Aanesland, Elvemo Sebastian
Tolnes, Andreas

Development of a system for retrieving vessel data for SINTEF Ocean

Retrieval of time series data

Bachelor's project in Software Engineering
Supervisor: Alexander Holt

May 2021

Bachelor's project

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Kunnskap for en bedre verden

Aanesland, Elvemo Sebastian
Tolnes, Andreas

Development of a system for retrieving vessel data for SINTEF Ocean

Retrieval of time series data

Bachelor’s project in Software Engineering
Supervisor: Alexander Holt
May 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Kunnskap for en bedre verden

Preface

This bachelor thesis was completed during the winter and spring of 2021 for NTNU in collaboration with SINTEF Ocean and follows a standard structure set by NTNU.
As more companies and nation states realize that the ocean is a treasure trove of data and unexploited resources, the need to collect and transfer vast amounts of data far from developed telecommunication infrastructure will be increasing in the next couple of decades. This was what led to interest into how to efficiently provide data services while in areas with connectivity and bandwidth issues. As new needs for stored data expands, the need to use it even in remote areas such as the arctic, which is becoming more and more relevant for global players for each passing year, go with it. In the future this need could expand from the oceans and frontiers of our planet, to that of the solar system and beyond. Knowing more about how to optimize this now could save time and resources where they are lacking the most in the future. The team also appreciates how this look into compression and transfer efficiency in systems could help better areas often overlooked by corporate interests as they are not user heavy. Whether this be a backwater in terms of infrastructure and wealth, or an area sparsely populated, the users need for data transmission won't change based on their location.

We wish to thank our contacts at SINTEF for their support and their enthusiasm in supporting our research. NTNU, and our councillor Alexander Holt in helping us keeping our research on track and focused. We would also like to thank our fellow student Odd-Erik Frantzen for his help in the early stages of the project with planning and writing up documentation.

Sebastian A. Elvemo:

____

Andreas Tolnes:

____

Problem Statement

Below is a translation of information taken from the original problem statement provided to NTNU by SINTEF[1]:

SINTEF Marine Data center collects data from various sources, including data logged from vessels. This data could be valuable for research and for shipping companies. We require a solution which provides such data through an API.

Tasks:
-Set up server functionality via Flask or similar.
-Functionality which allows for reading NetCDF files.
-Create functionality for creating data-sets with lower time resolutions.
-Create an API for easy access of data and metadata.

Possible additions:
-Create functionality for pre-processing and filter data.
-Handle data across multiple vessels.
-Access control functionality.

SINTEF can set up a service for testing actual data, and provide data sets for testing purposes.

Abstract

Large amounts of data is logged on sensors each year which SINTEF receives. But no system that SINTEF currently has can fulfill the task of storing, analyzing and and retrieving this data efficiently. This means that currently the data is of little value, as a technician would have to dig in the stored files themselves to extract the data they want. The data is often spread over multiple files, which is labour intensive and time could be spent better elsewhere.

To solve this, a system for handling uploads of NetCDF data and queries for this data was designed. Thus, whenever needed, the uploaded data could be accessed by a user of the system. Since the amount of data being logged by SINTEF could be enormous, a priority was placed on efficiency.

The chosen solution is a FastAPI server connected to an Influx database. The API then allows for querying against this database to get the data required. The application is written in Python and designed as a REST API with inspirations from GraphQL. To improve the efficiency of the queries, multiple compression techniques were added, including reformatting the response, using lower data resolutions, GZip, and more. As the size of the data transferred scales upwards, the gains from implementing such solutions will result in larger returns on performance and perceived speed. The application can eliminate 98% of superfluous data transferred through the compression methods used.

This report goes in detail through technology choices and methodology, as well as detailing the results of the project split into categories for easier reading.

Table of Contents

Preface i

Problem Statement ii

Summary iii

Definitions and Acronyms viii

1 Introduction and relevance 1

1.1 Background 1
1.2 Research Question 1
1.3 Goals 2
1.4 Report Structure 2

2 Theory 3

2.1 Previous research 3
2.1.1 Automating data pipelines for analysis of wave data . . . 3
2.2 Compression 3
2.2.1 Caching 3
2.2.2 Lossless Compression 4
2.2.3 Resolution Scaling 4
2.2.4 Response Time Analysis 4
2.3 Networking 5
2.3.1 Bandwidth 5
2.3.2 Packets 5
2.3.3 Packet Loss 5
2.3.4 Query 6
2.3.5 Response 6
2.4 System Architecture 6
2.4.1 Agile Development 6
2.4.2 Collaboration Tools 6
2.4.3 Version Control 6

3 Technology and method 7

3.1 Technology 7
3.1.1 Discord 7
3.1.2 FastAPI 7

3.1.3 Git 8
3.1.4 InfluxDB 9
3.1.5 Server-side caching 9
3.1.6 Jira 10
3.1.7 LaTeX 10
3.1.8 Microsoft Teams 11
3.1.9 Python 11
3.1.10 REST 12
3.2 Method 13
3.2.1 Development Team Roles 13
3.2.2 Kanban 13
3.2.3 Response data structure 14
3.2.4 Gzip 14
3.2.5 Delta encoding 15
3.2.6 Delta-of-delta encoding 15
3.2.7 Caching 16

4 Results **17**
4.1 Scientific Results 17
4.1.1 Gzip 17
4.1.2 ISO vs Unix epoch 18
4.1.3 Response data structure 20
4.1.4 Delta and Delta-delta encoding 24
4.1.5 Base case vs best case 25
4.2 Engineering Results 26
4.2.1 Goals 26
4.2.2 User Needs 27
4.3 Administrative Results 28
4.3.1 Project Handbook 28
4.3.2 Time Management 28
4.3.3 Development Process 29

5 Discussion **30**
5.1 Design Choices 30
5.1.1 Database solution 30
5.1.2 Programming languages 30
5.1.3 Flask and FastAPI vs Django 31

5.1.4 REST vs GraphQL 31
5.2 Scientific Results 32
5.2.1 Encoding 32
5.2.2 Compression 32
5.2.3 Summary 32
5.3 Engineering Results 33
5.3.1 Goals 33
5.3.2 User Needs 34
5.4 Administrative Results 35
5.4.1 Time Management 35
5.4.2 Development Process 36
5.5 Teamwork 36

6 Conclusion and Analysis **37**
6.1 Future work 38

7 References **39**

8 Attachments **42**
Problem Statement 56

List of Figures

List of Figures **vii**

1 Using RTA to find the bottleneck in response efficiency. 4
2 An example of a networking packet 5
3 Showing branching and merging in git[18] 8
4 Issue tracking on the project as shown in Jira 10
5 Example makeup of a REST URL.[28] 12
6 Percentage reduction between Gzipped and non-Gzipped requests. 17
7 Percentage reduction from responses using ISO vs Unix epoch without GZip. 18
8 Percentage reduction from responses using ISO vs Unix epoch with GZip. 19
9 List format vs key-value format reduction with GZip 22
10 List format vs key-value format reduction without GZip 23
11 Delta vs Delta-delta vs No encoding 24
12 Reduction from base case to best case. 25
13 Showing total hours spent for the different team members over time. 28
14 Showing time allocation based on activity. 35

Definitions and Acronyms

Backend	the part of software used by the frontend to store data, run calculations and acquire data.
CRC	Cyclic Redundancy Check is a type of error checking where the number of bytes is checked against received data to make sure no data has been lost.
DBMS	Database management system.
Devops	Development Operations, issue-tracking, planning, etc.
Frontend	the part of software which displays information to the user.
Git Branch	parallel sub-repository that allows for testing and alternate development.
HTTP Etag	Hash that enables cache verification to the web server letting the server refrain from sending the full response to the client.
InfluxQL	Influx Query Language, used by InfluxDB and is built on a modified version of SQL.
Kanban	an agile development method focusing on small teams and flexibility.
Mbps	Megabytes per second, a transfer speed descriptor.
NDA	Non-disclosure agreement.
NetCDF	file-type used to store sensor data with timestamps.
NTNU	Norwegian University of Science and Technology.
Packet Header	contains control data for the packet such as source and recipient.
Packet Trailer	contains data signifying the end of the packet and error checking such as CRC.
Python	a multi-purpose, high-level programming language.
REST	Representational state transfer, an architectural style in software used to create hierarchical web services.
RTA	Response time analysis, a method for finding bottlenecks in database query time.
Scrum	an early agile development method focusing on teams with clear leadership structure and development sprints.
SQL	Structured Query Language, a programming language for handling database operations in relational databases
Unix epoch	Amount of seconds since 1st of January 1970. Used in Unix systems to denote date and time.

1 Introduction and relevance

1.1 Background

In the modern day the amount of data transferred over networks has never been higher. And with the trend showing no signs of slowing down, some industries already face issues with poor network infrastructure where they operate. Among these industries are maritime and space-based industries.[2]

Maritime industries have multiple needs for data transferred from networks, be they navigational for their computer systems, sensor data uploaded to storage, or other communication systems.

The possibility then to lower the amount of data sent over these struggling networks should be of the highest priority, especially given that this information can be life-saving. Because of this, looking into such possibilities is the purpose of the research done in this thesis alongside the creation of an application to fit the needs given by SINTEF in their problem statement.

1.2 Research Question

To pinpoint the exact focus for the the thesis, the following research question was chosen:

"How can a system more efficiently provide data from large data-sets over low bandwidths"

To give some insight on this in a succinct way, reducing the resolution of the data, data-restructuring, and compression will be at the forefront of the development approach of the project.

1.3 Goals

Bullet-pointed below are the goals as found in the vision document for the project. These goals describe expected results in terms of features of the finished product.[3]

- *Create an API which can be used to obtain ship data in a usable form.*
- *Implement a GraphQL solution for requests up against a database.*
- *Deal with authentication and security of access of data.*

1.4 Report Structure

The thesis report will be split into sections detailing different aspects of the project. Firstly, an overview of relevant theory for the thesis will be covered. This will be followed by a description of the technology used, and the method used to collect data on which to base the conclusions. Thirdly, a discussion of the results found will be detailed along with possible improvements that or flaws in the project. This will be followed by a conclusion and an analysis of further research that could expand upon the findings.

2 Theory

The main focus of this thesis is on lowering the amount of data requirement to be transmitted, and as such, terminology related to compression and networking is included below. Included is also theory related to specific tools and methods used during the development of the application. The section will start with a look into previous research done in related fields.

2.1 Previous research

2.1.1 Automating data pipelines for analysis of wave data

This bachelor thesis from NTNU written in 2019 was also concerning streamlining data analysis pipelines using caching, REST, and a database. The system they created also used NetCDF data for the research. The database in question was not a time-series database, but was instead a standardized relational database built on SQL, and other than caching, nothing was done to reformat the response or otherwise compress the output as that was outside the scope of their thesis.

The research done here however does clearly show that many different sectors can benefit from streamlining their data storage, and analysis systems. In addition, it allows for comparison in terms of systems created for storing and analyzing NetCDF data. [4]

2.2 Compression

2.2.1 Caching

Within computing, caching details a storing method for data that is general and might be requested multiple times in the future. This is done in a way that another request for this information can be rerouted, skipping the entire need for another query against the database. This is useful for data that is transient, or might otherwise be asked for multiple times without the data changing in any way between these requests.[5]

2.2.2 Lossless Compression

Whenever data is compressed through a method so that it can be reconstructed back into the original data sequence, this is called lossless compression. Most of the compression methods used in this thesis are lossless. [6]

2.2.3 Resolution Scaling

Another method of compression is lowering the resolution of the data. In a time-series, this would mean aggregating multiple points into one, making the data less precise, but depending on the scale of the data, the amount of loss would differ. An example of this would be to scale a time-series from records of every second, to every third second.

2.2.4 Response Time Analysis

RTA is a method of evaluating database query efficiency, by looking for where the biggest bottlenecks in querying happens, which then makes it easier to focus on these parts to get the greatest rewards early on when developing solutions, before moving on to smaller bottlenecks once the large ones have been resolved. A diagram showing an example of discovering the bottleneck to work on next, is shown below: [7]

Figure 1: Using RTA to find the bottleneck in response efficiency.

2.3 Networking

2.3.1 Bandwidth

The maximum data transfer rate of an internet connection is called the bandwidth of that connection. Bandwidth is measured in Mbps, and although it is used to describe transfer speed, in reality it describes the amount of data that can be transferred at any one time, as travel-time of the data itself tends to be negligible due to the speed of fiber-optic communication systems. [8]

2.3.2 Packets

Whenever data is transmitted online, it is done so through sending collections of bytes called packets. The entire packet is made up of a series of bytes, with the header starting it, followed by the payload, and ending with the trailer. Packet bytes are split into two parts, the control information, and the payload. The control information consists of directions for delivering the data, including the recipient address, source, and error codes. This information is usually found in either the header, or trailer of the message. The payload consists of the information that the packet is delivering.[9]

Packet - E-mail Example

Header	Sender's IP address Receiver's IP address Protocol Packet number	96 bits
Payload	Data	896 bits
Trailer	Data to show end of packet Error correction	32 bits

©2000 How Stuff Works

Figure 2: An example of a networking packet

2.3.3 Packet Loss

When accessing a network or the internet, packets will be transferred. These packets can fail to reach their destination and this is called packet loss. Whenever this happens, the receiver of the data will have to be resent the information, and if this happens frequently it will manifest as the system hanging while it waits for the data.[10]

2.3.4 Query

Whenever a user requests data to be sent from a database this will be done through sending in a query to this database. The query will differ depending on the Query Language of the DBMS, the most common being SQL. Based on the instructions in the query, the database will then respond by sending out a response, which is sent back to the user.[11]

2.3.5 Response

The data sent back from a database after having received a query. This data will usually be organized into rows and columns, with all the data from the relevant tables and fields required by the query.

2.4 System Architecture

2.4.1 Agile Development

This describes a method of development where the goal is to be able to quickly respond to change. Agile development therefore puts priority on frequent client interactions and rapid deployment and iterations during development. Another big departure from previous waterfall style development methods is taking focus away from superfluous paperwork, only creating the most needed such as a vision document or a domain diagram.[12]

2.4.2 Collaboration Tools

Tools that serve to aid communication and cooperative work are called collaboration tools. These can include programs that allow for file-sharing, video conferences, conference calls, cooperative writing of documents and more. Examples of collaboration tools include software as disparate as Zoom, Overleaf, Discord, and Google documents.[13]

2.4.3 Version Control

Version control systems allow multiple developers to work alongside each other without them risking overwriting the work done by others while keeping records of all changes made to the project. This is done through maintenance of a central repository of files, which can only be updated upon the system granting the user access.[14]

3 Technology and method

This chapter will go over the technology and method used during the development of the project. This will be broken down into a subsection going over the technology, followed by a section going over the development methods used.

3.1 Technology

3.1.1 Discord

Discord is a collaboration tool offering both audio/video calls, and private servers with a combination of audio channels, and text and image channels.[15]

Discord was the main method of communication within the team during the project. Whereas most of the communication with SINTEF was done through email or Microsoft Teams.

3.1.2 FastAPI

FastAPI is a micro framework for developing web applications. Micro in this case meaning that it does not contain its own database abstraction layer, or any components that might be better handled by other third-party libraries. In this way, FastAPI is created to be useful as a base framework to build upon, while being very flexible in the way that this is done.[16]

FastAPI was chosen early on as the framework to use for the project. This came about as it seemed highly malleable, and because of the various development approaches that would be tested out for this project, this was on the forefront of the list of requirements the team considered in choosing a framework.

3.1.3 Git

Git is an open-source version control system, designed for software developments of all sizes. A useful feature of Git is branching out different 'sub-repositories', where the developers can work independently from updates in the central repository, and then later attempt to merge it back into the main branch. Branching out from the central 'master' branch, and only merging back finished or polished releases is standard practice and something that was followed during development.[17]

Figure 3: Showing branching and merging in git[18]

Bitbucket is a repository-hosting service run by Atlassian. It offers an easy to use UI overview of the repository, as well as options for CI, integration with other software such as Jira and Trello, and strong cloud security.[19]

In the early stages of the project a git repository was set up, using a private Bitbucket repository. This was owned by SINTEF, which meant that through all stages of development, SINTEF had access to the project to prepare for meetings or to ask questions. Development of features were also worked on in branch offshoots of the /dev branch of the project repository, which were only added to the /dev branch after having been reviewed by another member in the development team. As such other developers were kept in the loop about what had been worked on recently, as well as giving an extra layer of protection against buggy implementations making it into the /dev branch.

3.1.4 InfluxDB

InfluxDB is a an open-source database system specializing in handling time series data. It uses a modified query language built upon the SQL, named InfluxQL.

The data analyzed in the project were logs of different sensors from watercraft. Such data falls under the category of time-series data as it is logged periodically based on time. Usually such data would be simple to store, as sensors log a consistent number of points over a certain time span creating a series. But the removal and addition of sensors could happen at any time on a vessel, so the need for something that could dynamically handle such scenarios was of upmost importance. InfluxDB was suggested as a fitting candidate, given that it's measurement system is dynamic, and would allow for ease of inputting new data between already existing time-series data in the database. [20]

3.1.5 Server-side caching

The choice how to cache server-side data was a relatively simple one. The requirements were ease of use and the ability to store and retrieve data quickly. Since these are rather basic requirements, the choice became between Redis and memcached. As memcached is the simpler of the two with a straightforward Python library to interface with, and easy setup through Docker, it became a suitable choice for this use case.[21]

3.1.6 Jira

Jira is a collaboration tool that covers issue tracking on projects. It is owned and developed by Atlassian, which means it integrates well with the rest of their software suite such as Bitbucket. [22]

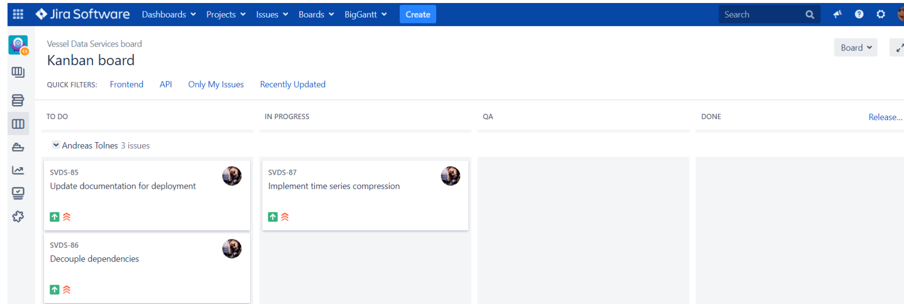


Figure 4: Issue tracking on the project as shown in Jira

In the very beginning of the project, the development team used Azure, but finding that SINTEF had existing Jira infrastructure set up integrated to their Bitbucket services, the project got moved over to Jira and Bitbucket. During development, issue tracking on Jira was used for big developments, using the issue tracking, which was divided into a front-end and back-end compartment as the team was also working in tandem, albeit separately, from another team developing a UI for the application.

3.1.7 LaTeX

A typesetting system used for writing documents with a high academic standard. Overleaf is an online collaboration tool for writing LaTeX documents in a team.[23]

LaTeX and Overleaf were used by the team in creating this thesis paper in an organized way while working physically apart from each other.

3.1.8 Microsoft Teams

Teams is a collaboration tool focusing on providing video conferencing to professional and academic settings. The service also offers a chat feature within groups called 'teams', where files can also be shared.[24]

Teams was used to share files with SINTEF, particularly documentation such as NDA's. It was also used to communicate with SINTEF about meetings, but since they had an internal version of the application, that made it harder to use appropriately. In the end, most of the communication ended up happening over email.

3.1.9 Python

A high-level interpreted general-purpose programming language. It is multi-paradigm-ed, and has thousands of modules to build on its framework. It fits well for projects both small and large in scope.[25]

Early on in the planning stage the team was contemplating whether to use Python or C#, and went for Python as it had better integrations with previous tools used by SINTEF, had more use to them for further development, and its modularity served well for interfacing with InfluxDB.[26]

3.1.10 REST

REST is an architectural style for web services, which uses a layered system for collecting data. To be considered RESTful, a service must fulfill the following:

1. Client and server separation.
2. The service must be considered stateless.
3. The service must be cache-able.
4. The service must have a uniform interface.
5. The service must be layered.

A signature feature of REST is that it uses URL input to send requests to the server, an example is shown below:[27]

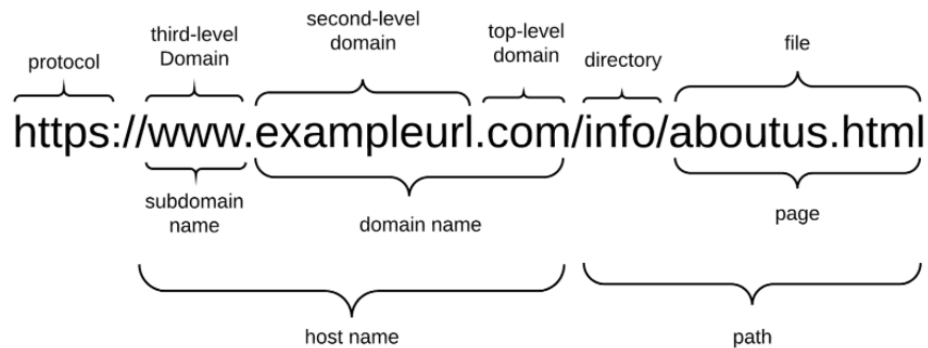


Figure 5: Example makeup of a REST URL.[28]

Initially during development, GraphQL was chosen rather than REST due to the system being more ideal for creating precise requests. But issues apparent with the GraphQL schema system for the intended purpose was discovered during development. The development team then pivoted to using a modified version of REST.[29]

3.2 Method

3.2.1 Development Team Roles

Early on during development the team established main fields of development. The team was using Kanban however, so there were no hard splits in team members only working on one aspect, but rather a system where one person had the main responsibility of one development branch. Meanwhile the others assisted where necessary with expounding on the frameworks. As development went further along, this became less and less relevant, and toward the end there was no such split at all. All documentation such as the vision document was done together in collaboration in the introductory stages of the project.

Sebastian did the main work early on doing NetCDF imports into InfluxDB, and upload improvements and testing.

Andreas set up the flask and Influxdb systems, as well as testing with GraphQL. During meetings Andreas also kept records, and that was one role that stayed throughout the project.

Odd-Erik did a basic framework of authentication systems before he left the project.

3.2.2 Kanban

Kanban is an agile development method focusing on transparency and maximizing inter-team communication to increase productivity. During the development process, issues are tracked and moved along a Kanban-board showing their state of development.[30]

During the development process the team used Kanban to develop the product, finding the issue-board useful to track issues still needing to be done, and to track the progress of the other members of the team. The Kanban board was on Jira, and most of the communication given the shutdowns due to Covid was done through Discord, and Teams.

3.2.3 Response data structure

One way to serve data is forwarding the raw query object from the database. In some instances this might lead to the data being formatted in a character heavy format, such as key-value pairs. Time series lend themselves useful to be represented as a list of points at a given time, where the first entry of the list contains metadata, and the subsequent entries only contain the actual data the client requested.

If a character heavy format is used, there can be significant gains to be made by restructuring the data before serving it across the web.

3.2.4 Gzip

Gzip is the most widely supported compression algorithm for HTTP content across the web with built in functionality in most modern web browsers and server frameworks. The most common version is the GNU Project's LZ77 compression, and compresses the content type of the request. It obtains a good balance between extra processing for client and server, and reduction in package sizes.[31]

As Gzip is implemented in a large majority of modern web browsers and web frameworks, this is decisively the easiest measure available to reduce package sizes. Since Gzip is built into most web frameworks, it is very likely to be available to most API developers and API consumers. As such it is highly relevant to test.

3.2.5 Delta encoding

Delta encoding, or delta compression, is a method for transmitting or storing sequential data in the form of differences between subsequent data points, also known as data differencing. In the context of time series, delta encoding performs greatly when applied to repetitive data such as time stamps and continuous data streams that changes over time. This works by keeping the difference between the previous and current value if the two values are subsequent, thus removing the need to store the actual value.

For example a given series of bytes 6, 2, 4, 2, 2, 3. This can be represented as 6, -4, 2, 0, 0, 1. This reduces the variance which allows the use of fewer bits to represent the same data.

If the data or time stamps can be encoded efficiently, then relative changes can be transferred instead of absolute values, as much of the interest of time series data are changes recorded from one value to another. And if the absolute values are needed, these can be easily reconstructed from the relative.

3.2.6 Delta-of-delta encoding

Delta-of-delta encoding (or delta-delta encoding) is delta encoding applied twice to the given data. This method removes even more redundant information by storing the difference of the previous difference. Time stamps is a prime candidate for this method, as normally one would record data at a given interval, and as long as that interval is largely consistent, then that data is superfluous.

For example a given series simplified Unix epoch timestamps 16015, 16016, 16017, 16018, 16020. This series of timestamps can be represented with 16015, 1, 0, 0, 1. Assuming the recorded timestamp is consistent, the representation can be made up of smaller integer numbers.

3.2.7 Caching

A cache is a middle layer between the server requesting data, and the database serving the data. The cache is usually smaller in size than the database, but a lot faster or closer to the server to access the data stored on the cache quicker than the database. When data is requested several times by the server, it can choose to store some of the data on the cache before it sends it out to the client. Next time the client asks for the same data again, the server can ask the cache for the same data back, which then enables a quicker response time.

As time series data in essence is immutable, caching of parameterized queries is an excellent way of improving the query time of the API. The client is very likely to request the same data multiple times for different display or analysis purposes. Creating a hash of the query that includes all parameters, ensures that every unique query has it's own hash and can be referenced in the cache. As the database or the web server might apply several heavy aggregations or mutations to the result before sending it out to the client, the subsequent calls to the expensive query can reduce the call time by several factors.

By also offering the same or a different hash based on the same parameters as an HTTP Etag, the client can choose to cache the data on the client machine, effectively reducing the call to a simple exchange of header information between the server and client.[32]

4 Results

This chapter will go over the findings of the project and will be broken up into sections. The first subsection will cover scientific results, the second will cover the engineering results, and the last the administrative results.

4.1 Scientific Results

Below are the scientific results of the project; the results dealing with answering the research question outlined in the first chapter of the thesis. The results are broken down into subsections covering the various methods for clarity.

4.1.1 Gzip

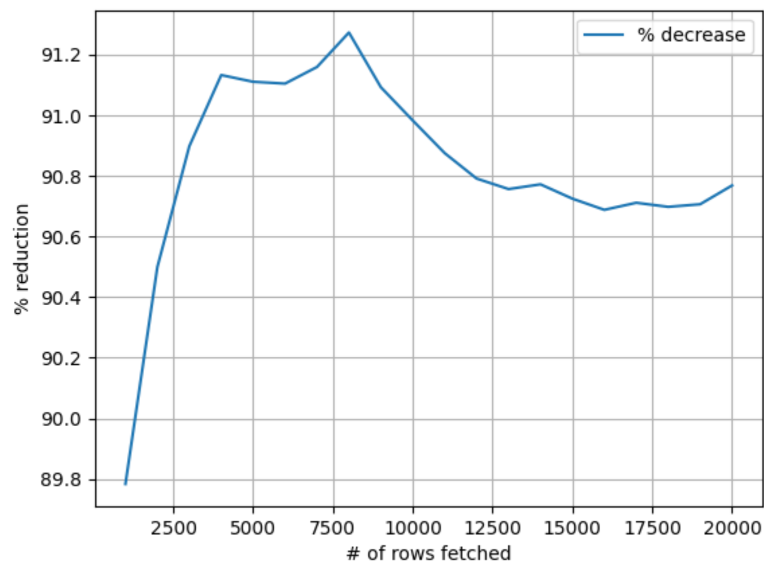


Figure 6: Percentage reduction between Gzipped and non-Gzipped requests.

As observed by the graph above, GZip is incredibly efficient and can easily reduce the package size of repetitive data such as time series by over 90 % without loss. As such it fulfills the necessary prerequisites for being called lossless compression.

4.1.2 ISO vs Unix epoch

Another way of reducing package size is to use Unix epochs instead of ISO for time stamps, as epochs can be represented with integers (Or decimals for higher resolution). Unlike ISO which is represented by a string that includes the traditional symbols humans use to distinguish different time sections such as years and months, and hours and minutes, epochs are made to be read by machines and don't need superfluous characters to distinguish units of time. Changing from ISO to Unix epoch proved itself necessary in order to implement other encoding techniques, but converting from ISO to epoch does have value in and of itself.

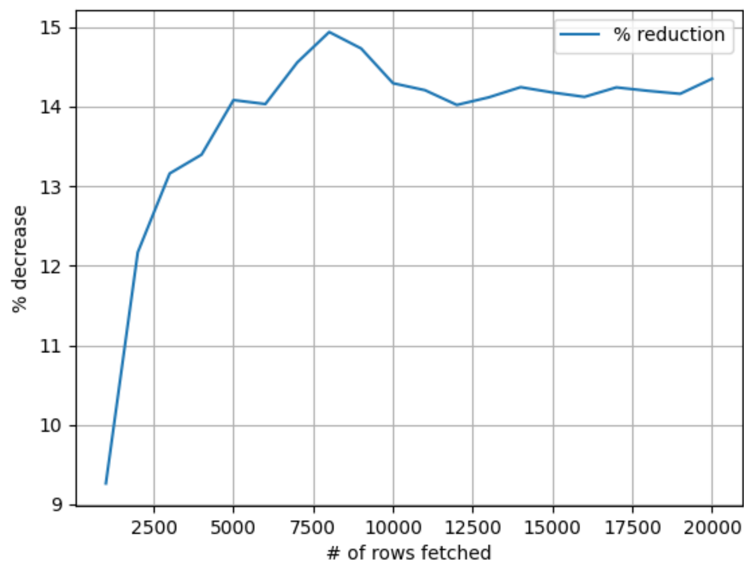


Figure 7: Percentage reduction from responses using ISO vs Unix epoch without GZip.

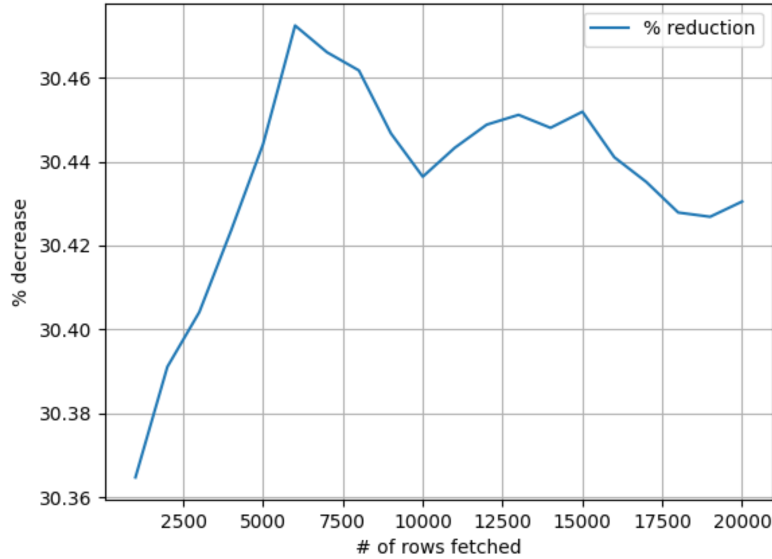


Figure 8: Percentage reduction from responses using ISO vs Unix epoch with GZip.

When paired with Gzip, the package size reduction almost more than doubles compared to the package that hasn't been zipped. This is a recurring effect when combining different encoding techniques and compression algorithms that are based on the *sliding window protocol.*[33] As epochs contains more repeating characters for each time stamp than the ISO format, the algorithm will have more compression potential.

4.1.3 Response data structure

With InfluxDB as the database, a raw unaltered response from a query containing three different data points, would look like the following piece of example code:

Listing 1: Time series as key-value pairs (207 Bytes)

```
1  {
2      "measurement": {
3          "data1": {
4              "2020-10-01T00:37:04": 123.5,
5              "2020-10-01T00:37:05": 124.4
6          },
7          "data2": {
8              "2020-10-01T00:37:04": 12345.2,
9              "2020-10-01T00:37:05": 2368.6
10         },
11         "data3": {
12             "2020-10-01T00:37:04": 10.2,
13             "2020-10-01T00:37:05": 10.1
14         }
15     }
16 }
```

In this example, each data point is represented as a key-value pair in a bigger parent dictionary. The values of each time stamp is accessed by using the time stamp as the key for each of the data points. For this particular response the size is 207 Bytes for 3 x 2 rows of floating point data.

If the same data were to be conveyed in a list format, where the actual wanted values are contained in sub-lists in a parent list, the amount of bytes transferred can be reduced drastically.

Listing 2: Time series as list of points (129 Bytes)

```
1  {
2      "result": [
3          [
4              "timestamp",
5              "data1",
6              "data2",
7              "data3"
8          ],
9          [
10             "2020-10-01T00:37:04",
11             123.5,
12             12345.2,
13             10.2
14         ],
15         [
16             "2020-10-01T00:37:05",
17             120.4,
18             2368.6,
19             10.1
20         ]
21     ]
22 }
```

The overall data was reduced from 207 bytes to 129 bytes, a total reduction of 78 bytes, or 37.68 % of the total data transmitted. This shows a very clear improvement as no non-duplicate data was lost during the transmission of the newer format. This means that the method satisfies the conditions for lossless compression.

With larger data sets, the difference starts to become significantly more noticeable.

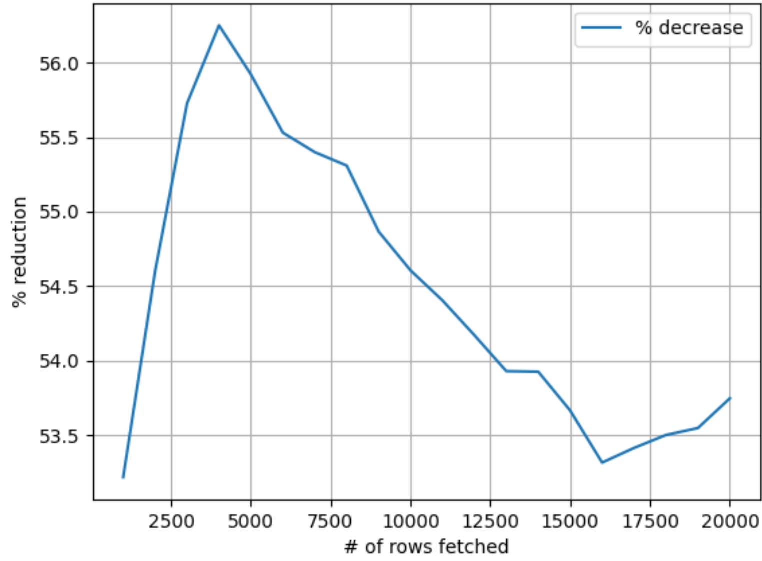


Figure 9: List format vs key-value format reduction with GZip

When run through a script that called the end point for an increasing amount of data, from 1 000 to 20 000 rows incremented by 1 000 for each request, a reduction up to 56.25 % was observed, with an average of 54.45%.

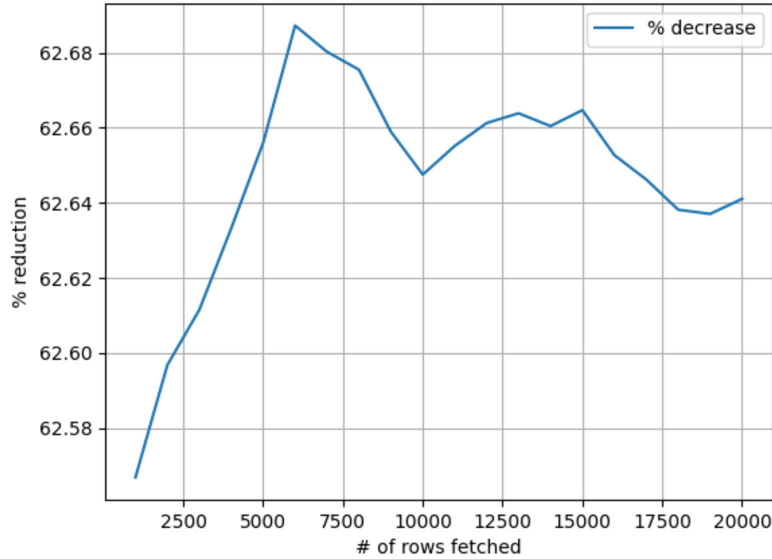


Figure 10: List format vs key-value format reduction without GZip

If Gzip compression is not applied, the difference between the key-value format and list format is even greater, with an average of 62.64%. This is probably because the contents of the reformatted package contains less repetitive data that can be compressed by GZip, which then lowers the observed yield from having a more efficient response format and GZip compression applied at the same time.

4.1.4 Delta and Delta-delta encoding

Delta encoding of the time stamps demonstrates a clear improvement of the package size with a consistent reduction of just shy of 16.5%. As delta encoding is simple to implement and requires very little processing, it shows itself to be a viable measure to take when trying to reduce the total load transferred.

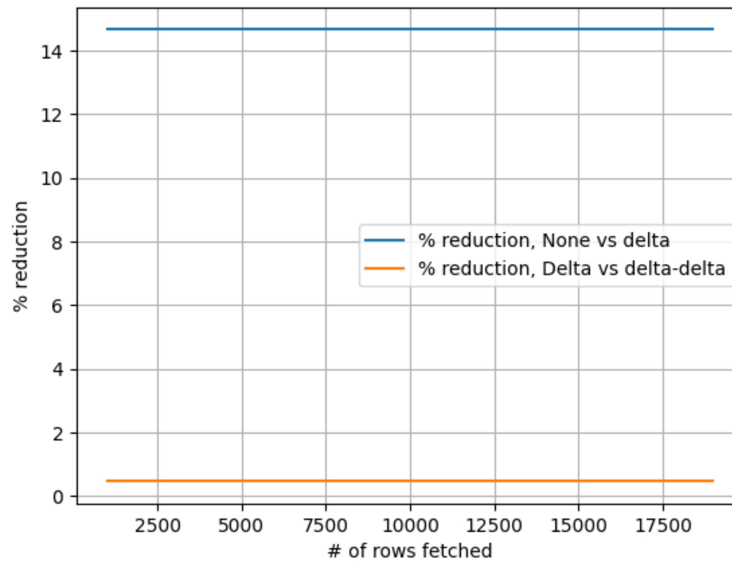


Figure 11: Delta vs Delta-delta vs No encoding

Delta-of-delta encoding is almost as simple as delta encoding, but in theory should yield greater gains when used on values with higher variance or when data is recorded sparsely. As time stamps usually is recorded at a consistent interval, the variance of the difference between each time stamp is close to, if not, flat out zero. When the data set is comprised of measurements taken each second, the extra encoding makes little difference. If the sampled data is recorded every hour, delta-delta encoding gave a reduction of 0.46% from standard delta encoding.

4.1.5 Base case vs best case

When the base case object representation from the database is used as the final response format, lots of redundant data is included in the response. The best case scenario uses a more efficient list format, Gzip compression, Unix epoch for timestamps, and Delta-of-Delta encoding. When pitched against each other, the observed reduction reached a max of 98.4% with an average of 98.23%.

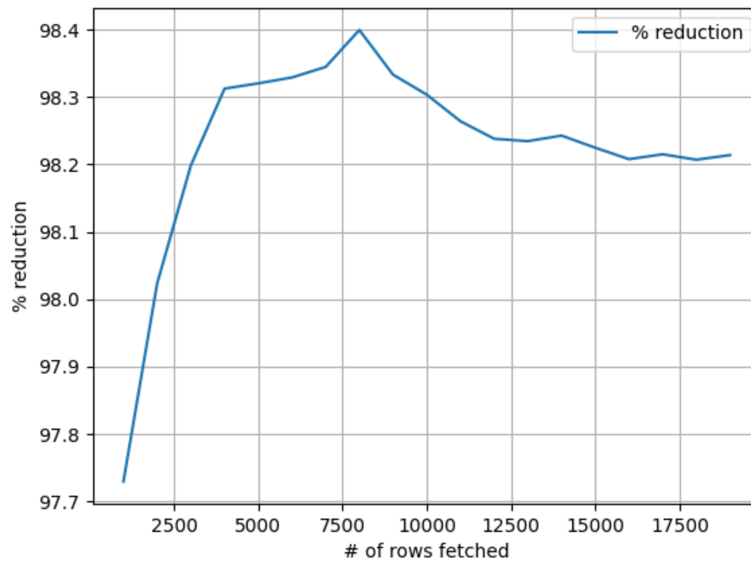


Figure 12: Reduction from base case to best case.

4.2 Engineering Results

Below the results compared to the goals and requirements set in the vision document for the project will be outlined, with special focus on the most relevant during development coming first.

4.2.1 Goals

- Create an API which can be used to obtain ship data in a usable form

The web-application created can both serve data, and data can be added to it, both fulfilling the requirements set up in the vision document. When data is to be uploaded, the data should be in the form of a NetCDF file, which can be read and uploaded to the database. Queries made against the system can be made for single, multiple, or all fields. The time-scope specifications for the queries are also malleable. Multiple compression methods also make the serving of data less network intensive.

- Implement a GraphQL solution for requests up against a database.

During development this was switched to a REST implementation, and as such the goal is fulfilled partly, although not by a GraphQL implementation. Rather a pseudo-GraphQL solution that uses POST requests with a body that contains the fields to be fetched, similarly to GraphQL. The solution is still mostly based on basic REST principles.

- Deal with authentication and security of access of data

Because of the time constraints due to the sudden absence of one of the team-members, this goal was not fulfilled in a satisfying manner. In the final product, no authentication system, nor way of isolating sensitive data from other users exists.

4.2.2 User Needs

• Access data from NetCDF files
Done through the netcdf_handler.py script in the application. The netcdf_to_dataframe function accepts a file, which is then made into a dataframe and uploaded to the database.

• Reorganize Data
Data requested is represented in the UI created alongside the request, and reorganization of the data can be done by requesting specific data and make that into new dataframes.

• Return relevant data
Queries can be sent to the application through the interface created, and it will return the relevant data to the user.

• Access control
Access control was planned, but not completed due to an unforeseen absence in the team late in development.

• Indepth security
Like access control, the focus on this was scrapped when it became apparent that the scope was bigger than could be reasonably accomplished after the departure of one of the team-members.

• Aggregation and calculation of sensitive data
Lowering the resolution of the data requested allows for aggregation and calculation of said interpolated results. On the security side nothing was finished, given the reasons stated above.

• Fast responses
Multiple compression and aggregation techniques are employed to provide faster responses. Among them, the response is made more efficient before it is sent to the user.

4.3 Administrative Results

This subsection will go over the administrative results, which are a subset of results related to time and overall project management.

4.3.1 Project Handbook

Gannt Diagram
In the planning phases of the project a Gannt Diagram for development, breaking the project up into weeks was set up. Although not highly specific, it provided an outline for what would be acceptable progress during the project. Although the development did not include design sprints as found in Scrum, the diagram helped the team keep an overall gauge of progress. Multiple deviations were eventually made from the Gannt-Diagram, mainly dealing with authentication and security.[34]

4.3.2 Time Management

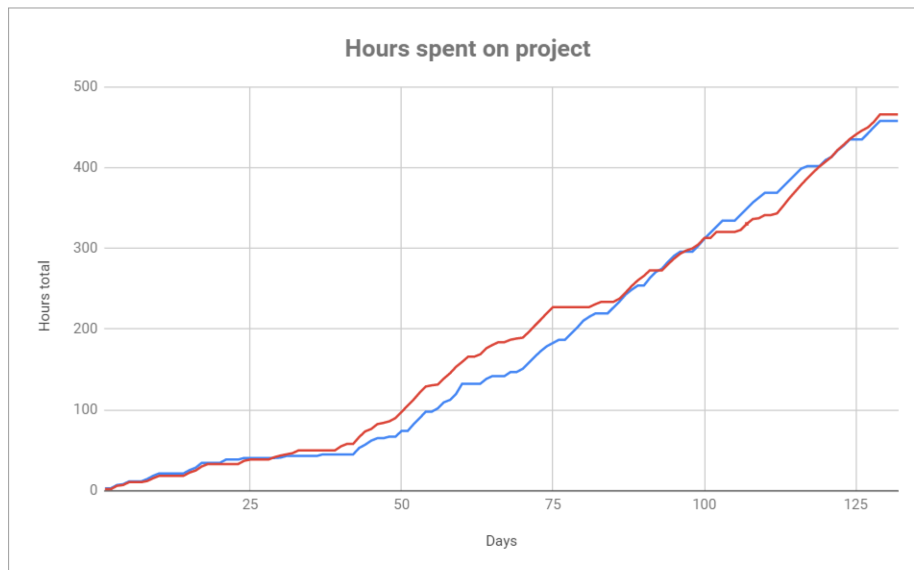


Figure 13: Showing total hours spent for the different team members over time.

From the chart above the hours spent working on the project can be seen over the weeks the project was active. For a more detailed analysis of the individual days and weeks, this can be found in the project handbook.

4.3.3 Development Process

Throughout the development, the team used Kanban as the development methodology, using Jira and the as an issue-tracker. The team also used the Gannt Diagram as a gauge of how development was going on a larger scale.

Git branching helped the individual developers work separately and test features without impeding the progress of others, and a system of needing approval of another team-member before merging back into the main development branch helped make sure that no sudden bugs emerged during development.

Daily descriptors of the work done that day was also written with the time-sheet submitted, making it easier to compare the actual results to the planned results. Weekly notes on what was completed that week was also made, and all of this documentation is put in the project handbook in the attachment section of the thesis.

5 Discussion

This chapter will discuss the findings in the results chapter and go over possible changes that could be done to achieve better results.

5.1 Design Choices

This subsection will cover alternative design choices, and their possible benefits over the chosen technologies.

5.1.1 Database solution

Postgres with the TimescaleDB plugin was an alternative that enables normal relational tables in Postgres to coexist with a specialized time series tables on the same instance. As the need for traditional relational databases are not present, and the advantages of a schema-free dedicated time series database, made InfluxDB a natural choice for this use case, and remains the better option for other projects like it.

5.1.2 Programming languages

There were two main choices discussed early during the planning, Python and C#. They both have robust libraries and frameworks for web API development and are fairly easy to work with given large amounts of troubleshooting sites and wide use.

C# with its good relationship between performance, development speed and access to third- and first- party libraries is a good choice for any web API and is a strong contender.

Python as a popular data processing platform with multiple open-source libraries is a fitting candidate for this kind of API as well. Even though Python is a less performant language, it still manages to keep up with heavy loads as demonstrated with its use in popular websites like Instagram.[35] Python with its data processing capabilities and quick development speed, as well as all parties involved being more familiar with the language and surrounding libraries, made it a natural choice for this project.

5.1.3 Flask and FastAPI vs Django

The two most popular web server frameworks for Python, are Flask and Django. Flask is lightweight and extension based, while Django is a full-stack aimed at delivering websites.[36]

Since Flask is very flexible and enforces little-to-no rules, it suits itself well to experimentation and creativity, rather than Django that is strict in how it wants the application to behave. Another new library that is a contender for Flask's position, is FastAPI by Sebastián Ramírez. It shares very similar syntax with Flask, and its main selling point being its ease-of-use when it comes to asynchronous operations, better performance, and self-documentation with the usage of strong typing and models.

As performant requests was a priority in this case, FastAPI was chosen for its similarity with Flask, with the extra performance and features that can improve the result of processing queries.

5.1.4 REST vs GraphQL

GraphQL at the onset of the project seemed like a natural choice, as the user can pick and choose what kind of data it wants, which falls in line with the need for smaller responses. As neither the database solution, nor GraphQL is in widespread use, there are currently no good libraries available to help with the mapping of models and the contents of the database. And the uncertain nature of the data, means that models and access methods would have to be generated dynamically, which proved itself to be unnecessarily complicated, creating a performance hit.

It was then decided to use a basic REST architecture as the base of the project, and then develop a more ad-hoc GraphQL solution that suited the projects needs more. Similarly to GraphQL, the main way of fetching data is to send a POST request with a body that specifies which data the user wants extracted.

Other more standard REST-endpoints are made available to document what is available on the database in place of GraphQL's schema system.

5.2 Scientific Results

Within this subsection an analysis of the results gathered to answer the research question will be contained.

5.2.1 Encoding

Delta encoding of time stamps showed a clear improvement over having now encoding at all, while being easy to implement the encoding server side, and decoding client side. Delta-of-delta is not much more complicated to implement, but didn't yield noticeable improvement. This is because of the consistency of the data used. It might play a bigger role if the data is inconsistently recorded and when combined with other encoding or compression methods such as simple-8b or run-length encoding which can encode repeated binary values. This was not experimented or tested with in this paper.

5.2.2 Compression

This paper experimented mostly with the standard Gzip implementation built into most web frameworks today. The result showed a drastic reduction in package size with minimal processing time.It was also observed that other methods of compression or encoding impacts the efficiency of Gzip. But this is mostly due to other methods reducing the amount of repetitive data, as such since Gzip is a lossless compression algorithm, it has a higher percentage of unique data patterns to conserve. Rather than lots of repetitive data that can be compressed away. Removing repetitious data was also experimented with. In one method rows containing only a timestamp change, without any other field change would be removed from the data. In this way the timestamps would only show the times data was changing. The system created for this however did not work as expected and was cut due to it using processing time, but it did not remove any rows in data experimented with given the very large number of fields in the data. The chance that all of them would remain static apart from the timestamp was so small that this did not yield much in terms of results.

5.2.3 Summary

All of the methods that were experimented on in this paper except Delta-delta encoding, showed a clear reduction of package size when applied to a real world data set. As such, when trying to compress data using RTA, the focus should be in the future on implementing all the methods with the biggest changes first. When all of the tested methods were applied at once, the observed reduction reached up to 98.4%, with an average of 98.23%. Easily implemented, the gains to be made when done so, are significant with very little overhead from processing on the server.

5.3 Engineering Results

This subsection will be evaluating certain results dealing with the vision document goals and user needs.

5.3.1 Goals

• Create an API which can be used to obtain ship data in a usable form

During the development, this goal was found to be too vague to be very useful in guiding development decision other than the most rudimentary choices of features. A possible way of improving it would be to more clearly convey what a usable form would entail, and more precise language dealing with functionality than 'obtain ship data'.

• Implement a GraphQL solution for requests up against a database.

Many difficulties were faced by the team throughout the development, particularly dealing with the GraphQL schemas and how to accurately fetch the data needed in a way that wouldn't result in many unnecessary database requests, and piecing the results together to one cohesive HTTP response.

• Deal with authentication and security of access of data

An important aspect to the lack of a security and authentication system is of course an ethical one. If software cannot ensure the privacy and security of information handed over to it, the chances for the loss and theft of this information is increased.

If the team had known from the beginning that a member would drop out half-way through the project, the focus on security and authentication would have been dropped completely. This would have come to pass as SINTEF had informed the team during a preliminary meeting that if security and authentication features were to be implemented, they needed to be implemented well.[26] Otherwise they could be dropped entirely, and be the focus of future development by SINTEF after the end of the project instead. As such, when faced with the decision of what to cut, the security features were chosen.

Implementing a system for web-tokens and using that for authentication would be a natural implementation that could be done later on, which, along with a login system on the front-end could allow the system to easily interpret which users were allowed what data.

5.3.2 User Needs

• Aggregation and calculation of sensitive data
The need for isolating sensitive data in particular was not met, as this would be a part of the authentication service that was cut to save on time after the departure of Odd-Erik Frantzen. This would have been a valuable addition because of ethical concerns considering the security of private corporate information, but as the application will not be going live until further work on these systems is done, the team feels certain that this issue will be resolved in a satisfying manner.

• Reorganize Data
More work could have been done to merge the application with the front-end development, thereby creating a better user experience when uploading and querying for data. As that is outside of the scope of the development however, and as time was of limited supply, this was cut to focus on more pressing features.

• Fast responses
High focus was put on this need as it served both the creation of the application and the answering of the research question for the thesis. As such, this is the need best satisfied by the end product.

5.4 Administrative Results

An analysis of the results dealing with the project handbook, development method, and time-budgeting will be contained in this subsection.

5.4.1 Time Management

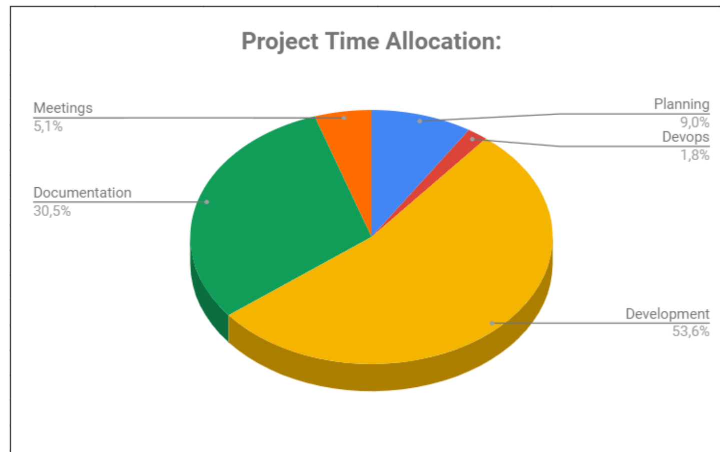


Figure 14: Showing time allocation based on activity.

The figure above shows the time allocation spent on various branches of development during the project from data seen in the project handbook attachment in the report. From this, some facts are striking.

As can be seen in figure 13, development started slowly, ramping up in the second half of March, around day 45. This is due to other intensive courses the team-members were working on simultaneously which ended in March. The graph therefore shows a steady workflow after that point, with deviations, but holding a general trend-line to the target hours worked.

Compared to initial time budgeting in the vision document, more time was allocated in the end to documentation and planning. The planning part can however be seen as time given to development, atleast in part, as the research into compression algorithms and technology to be used for development takes up the majority of that section if analyzed closer.

5.4.2 Development Process

Using Kanban presented many issues, the majority of them due to lack of previous experience with the process in the team. Issue tracking became too vague and included issues with too large of a scope early in the development. This made multiple developers cover the same functionality in different fashions as they fell under the topic of the larger issue. The time spent on this work could have been better spent at polish, or implementing features that were in the end cut from the product to save on time after the departure of one team-member.

In addition, Jira was not integrated well into the schedule of some team members, ending up with making the issue-tracking only useful to outside parties wanting oversight of development, rather than internally. This can be clearly seen in the time used for devops in the project handbook or in figure 14

5.5 Teamwork

The different team-members had varying degrees of skill with both Python and the various disciplines like networking and compression. Because of this, teamwork and the ability to communicate clearly was very important, and for the most part the team-work was steady and productive.

A major set-back came when just a month away from the project being delivered, the team received notice that Odd-Erik had pulled out of the project due to personal reasons. As he had been in charge of setting up authentication for the project, the remaining team-members decided to skip out on those features completely because of the time-crunch now placed on getting the other features done by the deadline.

Otherwise, disputes were solved through discussion and reasoned argument, and if this project was to be done again, more focus should have been placed on making sure that all team-members were more aware of personal issues right when they happened, rather than finding out later on. This would be especially helpful as the longer it takes to discover an issue, an exponential amount of time is needed to fix the defects stemming from it.[37]

6 Conclusion and Analysis

The research question covered throughout this thesis was:

"How can a system more efficiently provide data from large data-sets over low bandwidths"

Throughout the development, the biggest issue relating to answering the research question has been the staggering amount of different compression algorithms, techniques and different tools that were possible to integrate into the application. To deal with this, it was decided early on to focus on breaking the tools and methods into categories, and trying only the best fit within each category on the application, as the time constraints would otherwise ensure that the project would not be completed in a satisfactory manner.

With that stated however, the results clearly show that with restructuring and otherwise compressing the result before sending it back to the user, 98.23% of the data can be eliminated on average, ensuring that vital communication services, and sensor transmissions in places with limited telecommunication infrastructure can continue. As such, the performance of such systems can be improved in the inverse of this percentage, as transfer speeds are limited by poor bandwidth.

The web-application created for the purpose of researching this topic is fully functional and all code is currently in a repository run by SINTEF, which will use it to further develop it.

6.1 Future work

The completed API, although successful at its intended purpose has many possible additions that would be beneficial to reach an optimal satisfaction of the user needs described in the vision document.

The first important addition would be a better front-end service both for visualizing the data fetched from the server, as well as creating a more user friendly environment. This was being developed by another team as a project for NTNU, and as such combining it with the API would be a natural next step in the evolution of the product.

Additionally, security and authentication services are also a natural next step, as combining it up with a better front-end would lend it well to create a log-in system, which would be vital in maintaining customer confidence through ensuring the privacy of their vessel data from other users.

Looking further into removing data from fields that doesn't change between timestamps would also be a possible next step if the compression should be further improved. Such a solution should remove data from fields when they haven't changed from the previous timestamp, ensuring lossless compression.

Other methods of binary compression or encoding can further reduce the package size when used in conjunction with methods tested in this paper. Examples of methods to further test is simple-8b or XOR-based compression for floating point numbers. The methods tested in this paper can also be applied to more data fields if the data is known to be integers or floating point numbers, and might reduce the size even more.

7 References

[1] *Problem Statement, Attachment A.*

[2] *How much data is generated each day?* (Last Visited: May 19, 2016). URL: https://www.weforum.org/agenda/2019/04/how-much-data-is-generated-each-day-cf4bddf29f/.

[3] *Vision Document, see ZIP file with attachments.*

[4] Magnus Conrad Gjelseth-Borgen Trym Vegard; Hyll. *E39 Fjordkryssing: Automatisering av data-pipeline for analyse av bølgedata*. 2019.

[5] *Caching Overview*. (Last Visited: May 19, 2016). URL: https://aws.amazon.com/caching/.

[6] *Lossless Compression: An Overview*. (Last Visited: May 19, 2016). URL: https://cs.stanford.edu/people/eroberts/courses/soco/projects/data-compression/lossless/index.htm.

[7] *Response Time Analysis: How to Improve Database Performance by Measuring User Experience*. (Last Visited: May 19, 2016). URL: https://logicalread.com/response-time-analysis/#.YJNRELUzZPY.

[8] *Bandwidth*. (Last Visited: May 19, 2016). URL: https://techterms.com/definition/bandwidth.

[9] *What is a packet?* (Last Visited: May 19, 2016). URL: https://computer.howstuffworks.com/question525.htm.

[10] *What is Packet Loss?* (Last Visited: May 19, 2016). URL: https://www.forcepoint.com/cyber-edu/packet-loss.

[11] *Database Queries*. (Last Visited: May 19, 2016). URL: https://teachcomputerscience.com/database-queries/.

[12] *Agile 101*. (Last Visited: May 19, 2016). URL: https://www.agilealliance.org/agile101/.

[13] Michael; Page Carie L. Lomas Cyprien; Burke. *Collaboration Tools*. 2008.

[14] *What is Version Control?* (Last Visited: May 19, 2016). URL: https://www.perforce.com/blog/vcs/what-is-version-control.

[15] *Discord*. (Last Visited: May 19, 2016). URL: https://discord.com/.

[16] *FastAPI*. (Last Visited: May 19, 2016). URL: https://fastapi.tiangolo.com/.

[17] *Git*. (Last Visited: May 19, 2016). URL: https://git-scm.com/.

[18] *Branching figure git page*. (Last Visited: May 19, 2016). URL: https://git-scm.com/about.

[19] *Bitbucket*. (Last Visited: May 19, 2016). URL: https://bitbucket.org/.

[20] *InfluxDATA*. (Last Visited: May 19, 2016). URL: https://www.influxdata.com/.

[21] *What is Memcached?* (Last Visited: May 19, 2016). URL: https://memcached.org/.

[22] *Jira*. (Last Visited: May 19, 2016). URL: https://www.atlassian.com/software/jira.

[23] *The LaTeX Project*. (Last Visited: May 19, 2016). URL: https://www.latex-project.org/.

[24] *Microsoft Teams*. (Last Visited: May 19, 2016). URL: https://www.microsoft.com/en-us/microsoft-teams/group-chat-software.

[25] *Python*. (Last Visited: May 19, 2016). URL: https://www.python.org/about/.

[26] *Meeting with SINTEF 01.18.2021, see project handbook.*

[27] *What is REST*. (Last Visited: May 19, 2016). URL: https://restfulapi.net/.

[28] *REST example url*. (Last Visited: May 19, 2016). URL: https://frontend.turing.edu/lessons/module-3/rest-architecture-and-urls.html.

[29] *4 reasons why you should use GraphQL over REST APIs*. (Last Visited: May 19, 2016). URL: https://dev.to/blessingartcreator/stop-using-rest-for-apis-53n.

[30] *Kanban: What is it?* (Last Visited: May 19, 2016). URL: https://www.atlassian.com/agile/kanban.

[31] *Compression Tests: Results*. (Last Visited: May 19, 2016). URL: https://web.archive.org/web/20120321182910/http://www.vervestudios.co/projects/compression-tests/results.

[32] *FastAPI*. (Last Visited: May 19, 2016). URL: https://datatracker.ietf.org/doc/html/rfc7232#section-2.3.

[33] Philip Bille et al. "Lempel-Ziv Compression in a Sliding Window". In: *28th Annual Symposium on Combinatorial Pattern Matching (CPM 2017)*. Ed. by Juha Kärkkäinen, Jakub Radoszewski, and Wojciech Rytter. Vol. 78. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, 15:1–15:11. ISBN: 978-3-95977-039-2. DOI: 10.4230/LIPIcs.CPM.2017.15. URL: http://drops.dagstuhl.de/opus/volltexte/2017/7331.

[34] *Gannt Diagram, see project handbook in attachments.*

[35] *What Powers Instagram: Hundreds of Instances, Dozens of Technologies.* (Last Visited: May 19, 2016). URL: https://instagram-engineering.com/what-powers-instagram-hundreds-of-instances-dozens-of-technologies-adf2e22da2ad.

[36] *Why Django?*

[37] Maurice Dawson et al. "Integrating Software Assurance into the Software Development Life Cycle (SDLC)". In: *Journal of Information Systems Technology and Planning* 3 (Jan. 2010), pp. 49–53.

8 Attachments

A - Problem Statement (Norwegian)

B - Vision Document

C - Requirement Specification Document

D - System Documentation

Arbeidstittel:	SINTEF Marine Datasenter: API for fartøydata

Hensikten med oppgaven:
SINTEF Marine Datasenter samler data fra mange kilder, deriblant data som logges fra fartøy. Disse dataene kan være av verdi for rederi og for forskning. Vi ønsker utviklet en løsning som gjør slike data tilgjengelig via en egnet API.

Kort beskrivelse av oppgaveforslag:
Oppgaver:

Sette opp basis server funksjonalitet vha eksempelvis Flask.
Lage funksjonalitet for lesing av data fra NetCDF filer.
Lage funksjonalitet for å etablere datasett med lavere tidsoppløsning.
Lage API for å gjøre tilgjengelig data og metadata.
Opsjoner:
Lage funksjonalitet for å filtrere og evt preprosessere data.
Håndtering av data på tvers av mange fartøy.
Håndtering av tilgangskontroll (anta at autentisering er gjort).
SINTEF kan sette opp en egen tjeneste for testing mot virkelige data, samt stille data til rådighet for testing.
Oppgaven kan med fordel utføres av en gruppe og i kombinasjon med en eller flere av oppgavene "SINTEF Marine Datasenter: Etablere InfluxDB som lager av loggedata", "SINTEF Marine Datasenter: Innloggingsløsning", "SINTEF Marine Datasenter: Utvikle nettportal" og "SINTEF Marine Datasenter: Utvikle tjeneste for rederi basert på egne fartøydata".

Oppgaven passer for (kryss av de(t) som passer og skriv evt. en kommentar til oss):	- Bacheloroppgave - Masteroppgave
Kan oppgavestiller stille arbeidsplass med nødvendig utstyr og programvare:	Fysisk arbeidsplass avhengig av COVI-19.
Oppgaven passer best for, antall studenter:	- 1 - 2 - 3 - 4

Opplysninger om oppgavestiller

Er du fra bedrift/virksomhet eller er du student med en egendefinert/selvlaget oppgave?	- Bedrift/virksomhet
Navn på bedrift/organisasjon/student:	SINTEF Ocean
Addresse	Brattørkaia 17C
Postnummer	7010
Poststed	Trondheim
Navn på kontaktperson/veileder:	Karl-Johan Reite
Telefon:	99705393
Epost:	karlr@sintef.no

Ship data API for SINTEF
Vision Document

Project number 57

Version 1.1

Sebastian Aanesland Elvemo
Andreas Tolnes

2021

Revision History

Date	Version	Description	Author
19/01/21	1.0	Creation and drafting	Elvemo, Frantzen, Tolnes
05/05/21	1.1	Minor changes, revisions	Elvemo, Tolnes

Table of contents

1 Introduction **4**

2 Summary of problem and product **4**
2.1 Problem statement 4
2.2 Product statement 4

3 Project goals **5**
3.1 Result goals 5
3.2 Process goals 5

4 User Descriptions **5**
4.1 Stakeholders summary 5
4.2 User Summary 5
4.3 User Environment 5
4.4 User Needs 6

5 Product Overview **6**
5.1 Products role for users 6
5.2 Assumption and Dependencies 6
5.3 Risk analysis 6
5.4 Cost, pricing and benefits 6

6 Product Features **7**
6.1 Features 7

7 Requirements **7**
7.1 Operational requirements 7
7.2 Technical requirements 7
7.3 Process requirements 8
7.4 Usability requirements 8

1 Introduction

This document describes a project created for SINTEF Ocean as part of the subject "TDAT3001" at NTNU.
The project entails the creation of a web server API for use by SINTEF to make previously collected maritime data easily malleable for research purposes. The data is currently stored in NetCDF files, and the project will entail creating a product which can read and sort the data within these files, and give the users the data requested.

2 Summary of problem and product

2.1 Problem statement

Problem	Maritime data collected isn't easily accessible
Involves	Researchers and data-analysts
Resulting in	The data cannot be used for research without manual parsing
A better system will result in	Easier access to data for analysis and visualization.

Problem	Maritime data cannot easily be put into datasets.
Involves	Researchers and data-analysts.
Resulting in	Sorting data takes a lot of time.
A better system will result in	Time saved compiling datasets.

2.2 Product statement

For	SINTEF
Who	need to process and serve maritime data, from many sources at various resolutions
Our product	is a web server
Which	will provide an API for fetching data, and do the processing and handling needed
As opposed to	the current lack of accessibility to this data, which leads to SINTEF not exploiting this data to its full potential
Our product	will make any future uses of this data easier

3 Project goals

3.1 Result goals

- Create an API which can be used to obtain ship data in a usable form.
- Implement a GraphQL solution for requests up against a database.
- Deal with authentication and security of access of data

3.2 Process goals

The end result of the project will result in a well documented path of development, where all major and minor decisions are documented in a bachelors thesis. Project members will also have increased competency in lean development, development of APIs, and handling new technologies.

4 User Descriptions

4.1 Stakeholders summary

Name	Description	Requirements
Student team 57	Members developing the product	Planning, developing and documenting the development of the product.
Sintef Ocean	The client for the project.	Set development requirements and set product specifications.
NTNU	Supervising the Project	Set academic standards for the project, advise during development.
Front-end Developers	Student team creating related front-end	might use our API, pending communication

4.2 User Summary

Name	Description	Role
Developers	Internal SINTEF developers	Will further develop and use the API for data visualization and/or analysis.
Data Analysts	Workers within SINTEF that need parsed data	Use the system for acquiring data.

4.3 User Environment

The system will be deployed online, and as such will be available on a computer with an internet connection.

4.4 User Needs

Need	Priority	Related to	Proposed solution
Access data from NetCDF files	HIGH	Sysdev	Server handles parsing/interpretation & analysis
Reorganize Data	HIGH	Analysis	Server can create views for data, and manage data.
Return relevant data	HIGH	Sysdev	Server can be queried for specific data
Access control	HIGH	Authentication	Create an authentication system using web-tokens.
Indepth security	LOW	Authentication	Full infosec inspection.
Aggregation and calculation of sensitive data	MEDIUM	Analysis	Create system for back-end calculations to maintain the security of sensitive data.
Fast responses	MEDIUM	Sysdev	Caching of data, summaries

5 Product Overview

5.1 Products role for users

Will allow access to data through a GraphQL endpoint, allowing easier, faster, more centralized usage of data, with some level of access control.

5.2 Assumption and Dependencies

Time series database hosted in a Linux environment by SINTEF. Project period is between week 2 and 20, and the budgeted amount of hours for the entire project is 500 hours, but many of these are used on the thesis and other university work.

5.3 Risk analysis

Number	Risk	Probability (0-10)	Severity (0-10)
1	Serious Illness like Covid	3	8
2	Problems arising from using new technologies	8	3
3	Issues arising from the NetCDF data formatting	4	7

5.4 Cost, pricing and benefits

The development team consists of three members, who all have a max budgeted amount of hours for this project. Below is a general overview of the best and

worst case hours spent on product development for SINTEF for the completion of this project:

	Hours per person	Total Hours
Best Case:	300	900
Average:	350	1050
Worst Case:	400	1200

6 Product Features

6.1 Features

- Serve data from a GraphQL endpoint
- Limit data access based on authentication
- Read NetCDF files
- Read databases
- Compile data into a less fine-grained resolution before serving

If time allows:

- Filter and pre-process data
- Treat data across many vessels
- Add authentication and authorization
- Add robust security
- Cache data for speed of response and efficiency

7 Requirements

7.1 Operational requirements

The service must be hosted in a linux environment, preferably using Docker.

7.2 Technical requirements

- Read NetCDF files
- Use GraphQL or Rest implementation
- Use Python

7.3 Process requirements

Frequent meetings between product owner and project group will be held to ensure that the product will be delivered to the product owner's satisfaction. Kanban will be employed as the projects primary project management system hosted on Microsoft Azure.

7.4 Usability requirements

An important requirement of this project is that the finished product be easily iterated upon by outside developers and be integrated into a front-end solution created by another development team. As such, during development, extra care will be taken to avoid pitfalls for later developers such as messy code.

Software Requirements Specification
Ship data API for SINTEF

Version <1.0>

Table of Contents

Contents

1. Summary 3

2. User Stories 3

 2.1 Reading and Uploading data 3

  2.1.1 Decipher NetCDF files 3

  2.1.2 Upload 3

 2.2 Setting up a query 3

  2.2.1 Timescale 3

  2.2.2 Fields 3

  2.2.3 Changing resolution 3

 2.3 Querying for data 3

  2.3.1 General Query 3

 2.4 User Information 4

  2.4.1 Error reporting 4

3. Architecture model 4

Software Requirements Specification

1. Summary

This document is written as a part of the official documentation for a bachelor thesis project at NTNU during 2021. This documentation will specify the requirements for the finished application for ship data retrieval which will be delivered to SINTEF Ocean at the end of the development cycle. The application has a system for uploads of NetCDF file data, and methods for querying against a time-series database to retrieve the uploaded data. A frontend solution is also being developed separately from this project, but a simple UI will be delivered with the project.

2. User Stories

2.1 Reading and Uploading data

2.1.1 Decipher NetCDF files

As a	User
I want to	Read NetCDF files
So that	My data can then be uploaded to the database.

2.1.2 Upload

As a	User
I want to	Upload to the database
So that	My data is stored for easy access.

2.2 Setting up a query

2.2.1 Timescale

As a	User
I want to	Query within a given timeframe
So that	I only get the data relevant to me.

2.2.2 Fields

As a	User
I want to	Query within specific fields
So that	I only get the data relevant to me.

2.2.3 Changing resolution

As a	User
I want to	Change the resolution of the query
So that	I get aggregated data over large timescales quicker.

2.3 Querying for data

2.3.1 General Query

As a	User
I want to	Get specific information from the database.
So that	I can use that data for research and analysis.

2.4 User Information

2.4.1 Error reporting

As a	User
I want to	Know if and why a query has failed.
So that	I'm not wasting time making mistakes and waiting.

3. Architecture model

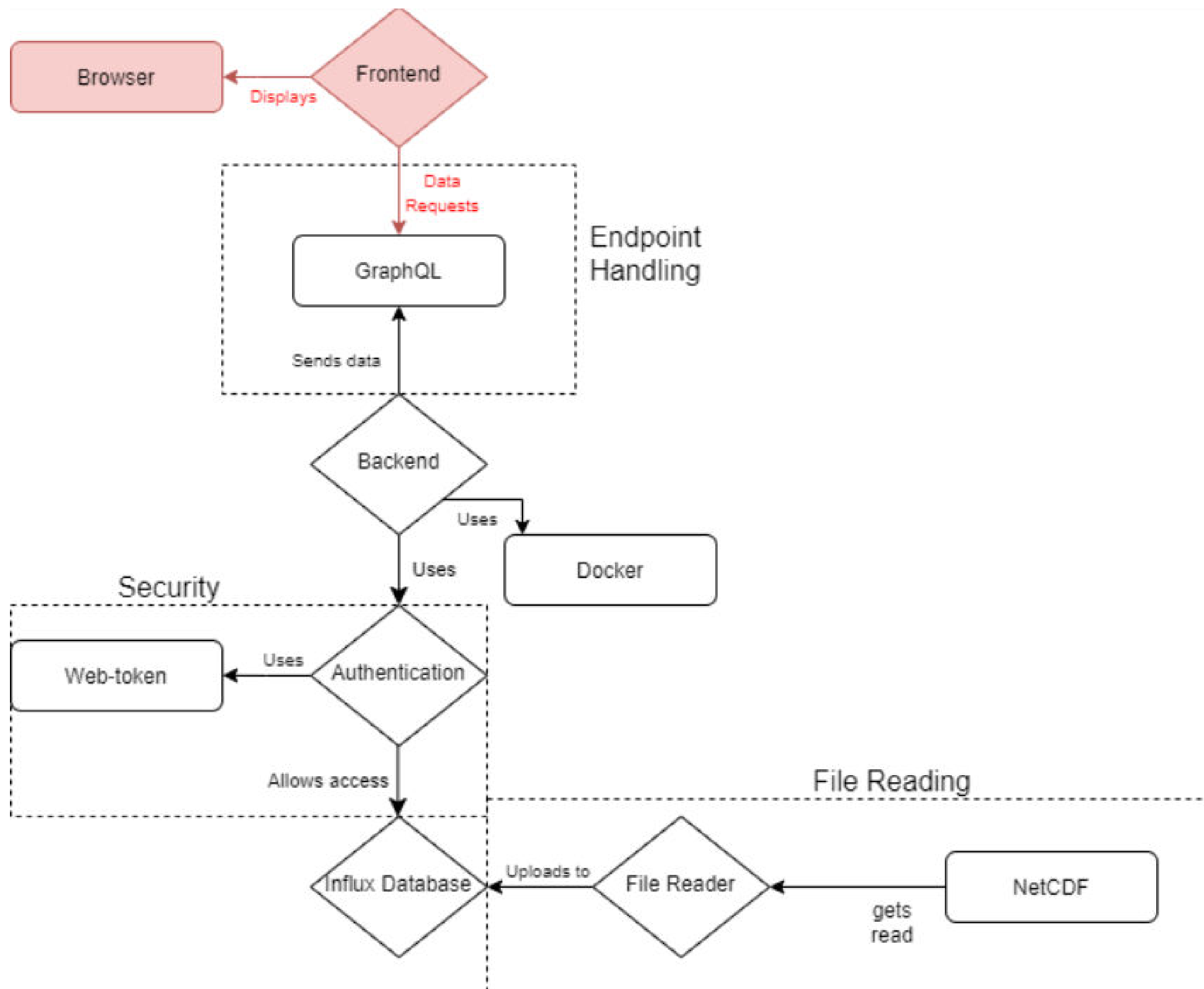


Figure 1: Figure showing a simplified architecture model, with the separately designed front-end in red.

System documentation

May 19, 2021

Table of Contents

1 Structure **1**
1.1 Data files . . . 1
1.2 Application flow . . . 1
1.3 Code standard . . . 2
1.4 Dependencies . . . 3
1.5 Security . . . 3

2 Server **4**
2.1 REST API . . . 4
2.1.1 Route description . . . 4
2.2 Environment . . . 4
2.3 Database . . . 4
2.4 Cache . . . 4

3 Mock clients **5**
3.1 OpenAPI . . . 5
3.2 Postman . . . 6

Installation **7**

List of Figures

1 Simple chart of the flow of the application 2
2 OpenAPI POST example. 5
3 Postman example query with headers. 6

1 Structure

As Python and FastAPI is limitless in which way to structure the application, a lot of freedom is granted to the developers. The system is structured with separation of concern in mind, with influences from object oriented programming practices such as dependency injection.

Source folder structure description.

/database	Contains connection creation to cache and database
/models	Different models for responses and requests as well as database and cache interfaces
/routers	Contains all routes that the API provides, as well as the header handling logic.
/services	Files handling the data fetching and formatting so it can be directly consumed by the routers

1.1 Data files

Data files are imported from time series saved in NetCDF files provided by SINTEF Ocean. The importing of data can be independent of the running system. For development purposes NetCDF files can be placed in the /netcdf folder in the root directory, and send a POST request to any of the seeding endpoints to import the data to InfluxDB. Data files are not provided as part of the repository.

1.2 Application flow

Sections and modules are split according to the separation of concern principle, and each handle a specific task unique to themselves. Dependency injection is implemented in the database and cache providers to enable easier mocking for testing pipelines.

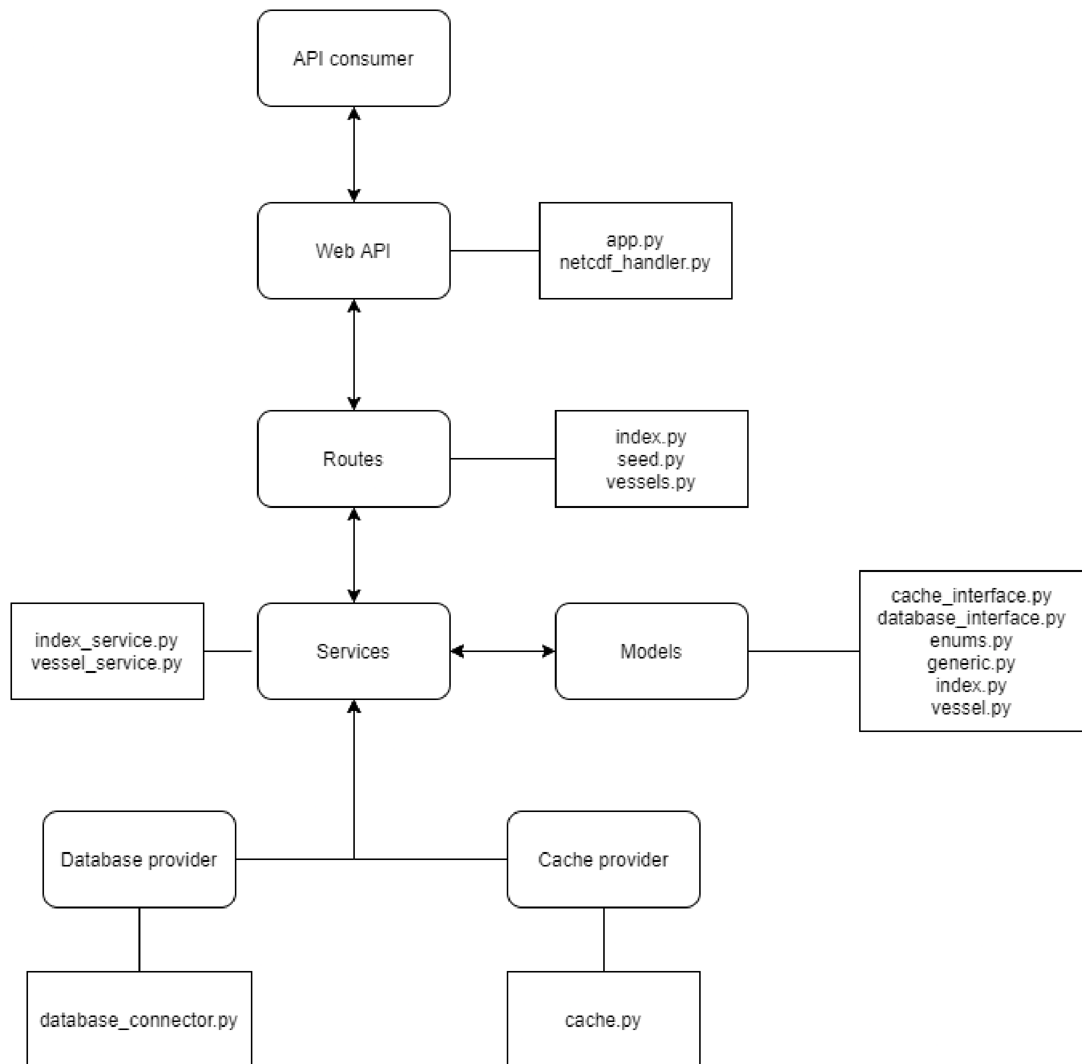


Figure 1: Simple chart of the flow of the application

1.3 Code standard

The code base follows the standardized PEP 8 style guide common for Python projects.

1.4 Dependencies

The dependencies constitute mostly of public libraries that interface with the different systems that the server communicates with and uses, such as the database, cache, and reading of NetCDF files. Dependencies are managed by the package and dependency manager Poetry.

- FastAPI
- Uvicorn
- Numpy
- Pandas
- Influxdb
- xarray
- netCDF4
- pydantic
- pymemcache
- python-dotenv

1.5 Security

Due to the flexibility requirements of the system to handle a number of unknown data types and several optional parameters, the query is highly dynamic based around the parameters from the client. This can pose a security risk in the form of SQL injections if not handled properly. To mitigate this risk all fields that the client requests are cross checked with the available fields on the database to make sure that no foreign or unexpected input is allowed. Frequent use and strict enforcement of different types of enumerators, primitives and complex data types ensures that all parameters in the query are checked and validated in one way or another before being queried to the database.

2 Server

2.1 REST API

2.1.1 Route description

/index	GET	Shows a list of available endpoints.
/docs	GET	OpenAPI documentation through swagger interface.
/redoc	GET	Redoc representation of the API documentation.
/vessels/values	POST	Returns a list of values with corresponding time stamps filtered by given parameters.
/vessels/aggregates	POST	Returns a list of aggregated values with corresponding time stamps filtered by given parameters.
/vessels/aggregates	GET	Returns a list of fields that are available on the server

2.2 Environment

Environment variables can be set by creating a .env file in the root directory. The file should contain database credentials as well as where to access the database and cache. See installation guide for further details.

2.3 Database

As InfluxDB is a dynamic database, no database structure or schema is needed and different columns of different data types can be added at any time without breaking the system. When compared to relational databases, the time stamp is always the primary key and tags can be added as something akin to foreign keys.

2.4 Cache

In memory cache memcached is available at port 11211, and is handled by cache.py which is called upon by any of the service modules that need it. It takes in JSON compliant data structures and stores them as bytes. When the data is fetched again it needs to be converted from binary to the original JSON structure.

3 Mock clients

Since no real client is consuming the API as of the moment of writing, requests needs to be made through specialized software for analyzing and testing APIs such as Postman or curl. Alternatively, interactive OpenAPI documentation can be used to test and explore each endpoint.

3.1 OpenAPI

With OpenAPI it is possible to see what parameters are obligatory to the request and what is optional, as well as example values for each. Real queries can be made through OpenAPI and is one of the easiest ways of getting to know what can and can't be done with the API.

to_date
string($date-time)
(query)
2021-05-19T13:04:00

from_date
string($date-time)
(query)
2019-05-20T13:04:00

limit
integer
(query)
100

offset
integer
(query)
offset

encoding_type
string
(query)
DELTA

etag
string
(header)
etag

Request body required
application/json

```
{
  "fields": [
    "field1",
    "field2",
    "field3"
  ]
}
```

Execute

Figure 2: OpenAPI POST example.

3.2 Postman

Postman is a specialized software for API development and includes several advanced feature sets. Queries made through Postman can be customized to a larger extent than through the OpenAPI documentation, with easy header manipulation and advanced HTTP settings to test different scenarios and configurations. Response time and size are broken down to make bottleneck analysis swift and easy.

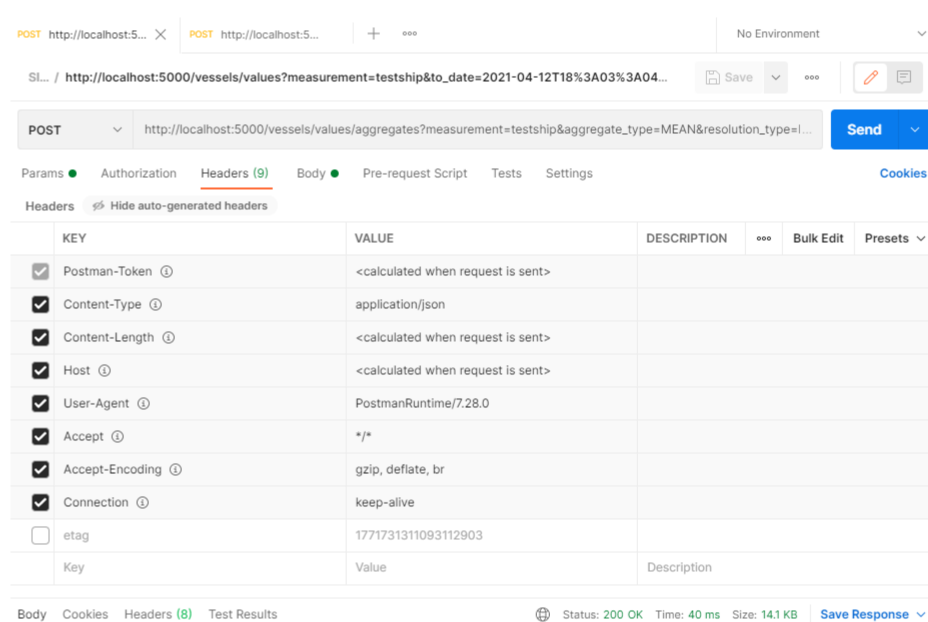


Figure 3: Postman example query with headers.

SINTEF API

How to run for development with Docker (Recommended)

> **Windows note**: Docker might not work with Windows 10 Home. Try using a Linux based distro or upgrade to Windows 10 Pro/Education/Enterprise

1. Make sure Docker and docker-compose is installed and working on your system.
2. Create a `.env` file in the root directory with the following example contents for Docker development. Replace with your own credentials if necessary.

```
HOST=db
PORT=8086
USERNAME=root
PASSWORD=root
DATABASE=wow

CACHE_HOST=cache
CACHE_PORT=11211
```

3. Run `docker-compose up -d --build` to build and launch web API, cache, influxdb and Grafana containers.

`--build` flag can be omitted after first run unless changes are made to docker files or python dependencies.

Containers should now be accessible at: - Web API: `localhost:5000` - memcached: `localhost:11211` - InfluxDB: `localhost:8086` - Grafana: `localhost:3000`

API documentation is available at `localhost:5000/docs` or `localhost:5000/redoc`

Stop containers

Use `docker-compose down` to stop running containers

Grafana

You can access Grafana to inspect data in the database by accessing `localhost:3000`. If you're prompted for credentials, enter standard credentials `admin/admin`.

If a data source is not already configured, you can configure it yourself with the following settings in `Configuration -> Data Sources -> Add data source` :- HTTP - URL: `http://db:8086/` - Access: `Server(default)` - Basic auth: `On` - Basic Auth Details - User: `root` - Password: `root` - InfluxDB Details - Database: `wow` - User: `root` - Password: `root` - HTTP Method: `GET` (Optional)

Run API outside of Docker (Not recommended)

1. Install InfluxDB and get it running on your computer.
2. Change the IP in `src/database/database_connector.py` to `localhost:8086` (Or desired IP).
3. Install python dependencies with Poetry `poetry install`.
4. Enter virtual environment with `poetry shell` OR run directly with `poetry run uvicorn --host 0.0.0.0 --port 5000 --root-path . src.app:app --reload`.
5. If you entered a new shell, use `uvicorn --host 0.0.0.0 --port 5000 --root-path . src.app:app --reload` to run the server locally outside of docker.

Web server should now be accessible from `localhost:5000`

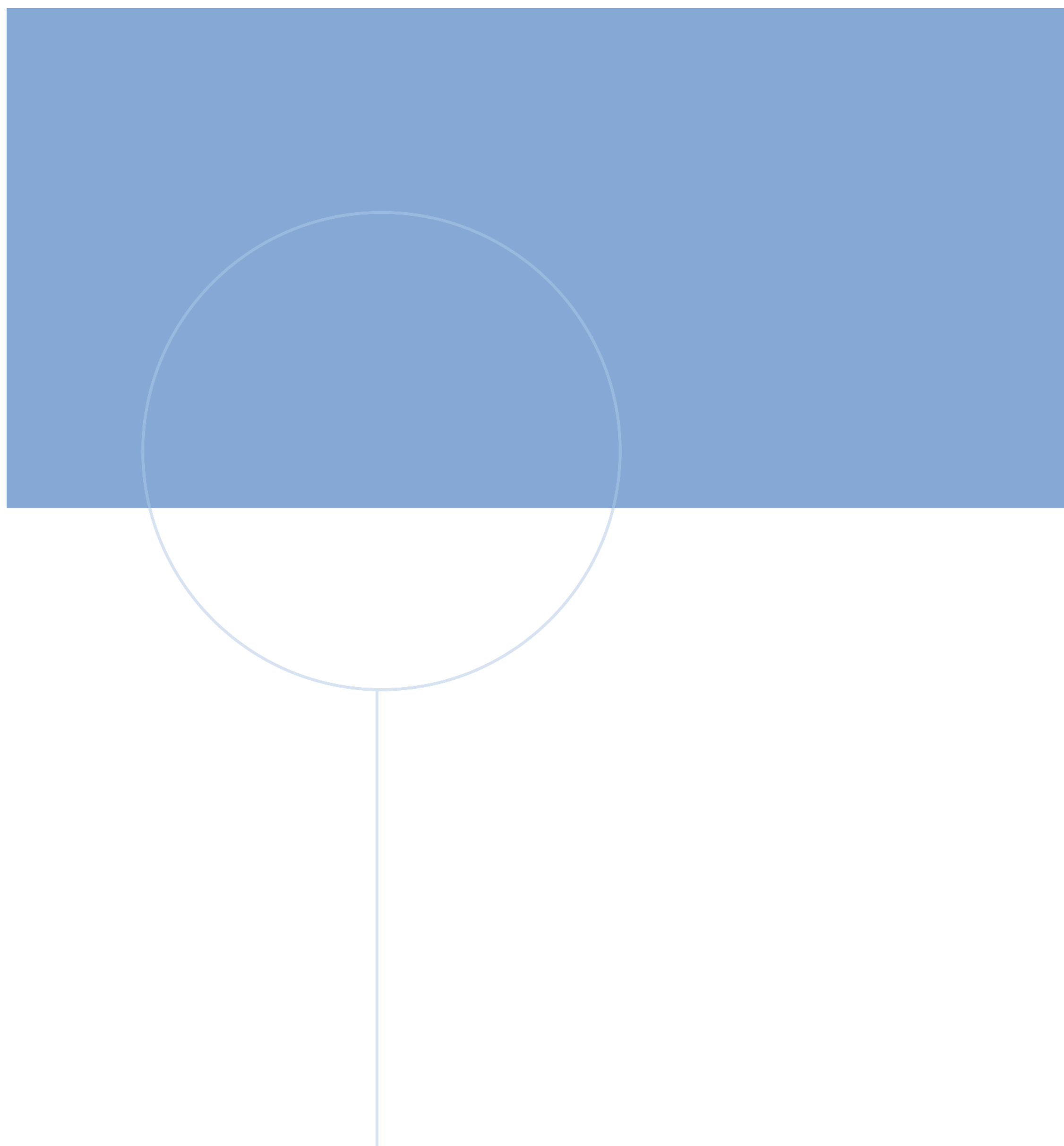
Importing test data

For development purposes, two endpoints for importing data found in `./netcdf` are available. See `localhost:5000/docs`. These are not meant for production and are subject to change/removal at any time. Data is not included in repository.

Who to contact?

Andreas Tolnes andrtoln@stud.ntnu.no

Sebastian Aanesland Elvemo sebastel@stud.ntnu.no



NTNU
Kunnskap for en bedre verden