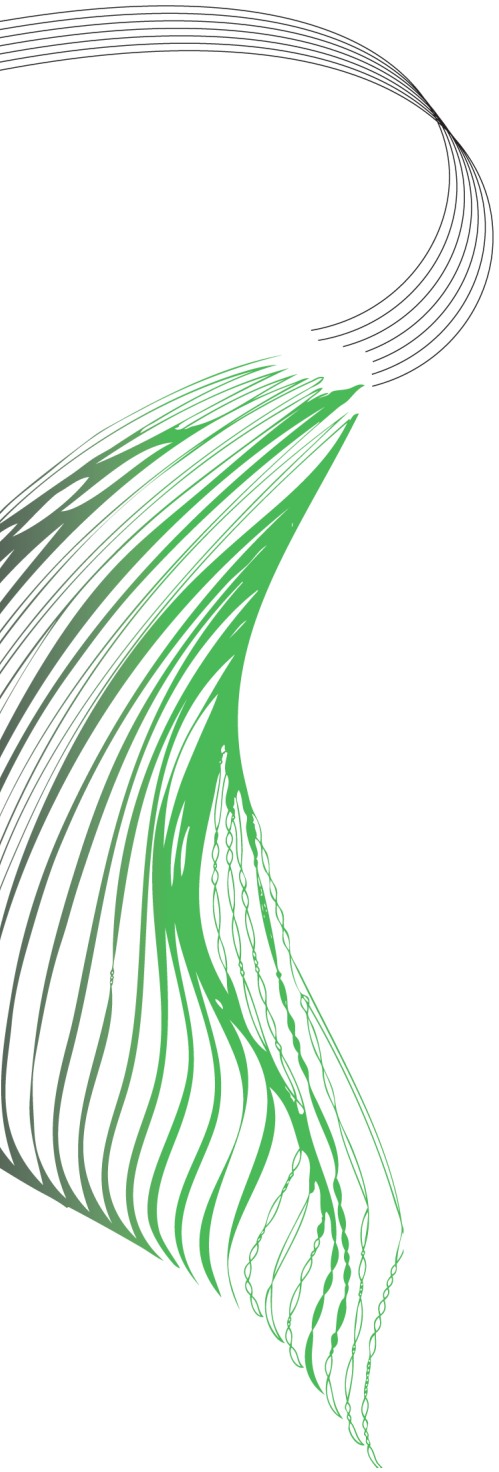
Design Project
BSc Technical Computer Science



Thales
Naval Collaboration Screen

by

Jordi Bals	s2253194
Jeroen van der Horst	s2389576
Hyeon Kyeong Kim	s2112019
Priya Naguine	s2315653
Puru Vaish	s2236141

Supervisor: Dr. Doina Bucur

April 22, 2022

Department of Computer Science
Faculty of Electrical Engineering,
Mathematics and Computer Science

UNIVERSITY OF TWENTE.

Abstract

This report details the entire trajectory of the Naval Collaboration Design Project supervised by Thales, containing details from the design phase, implementation phase and evaluation phase. Additionally, it encompasses the administrative process executed throughout the project together with the supervisor from the University of Twente and the external supervisors from Thales.

The Naval Collaboration system is a proof-of-concept project, consisting of a data-sharing architecture and a user interface for the collaboration screen used by the operators in the TACTICOS Combat Management System delivered by Thales. The project aims to build a prototype of an architecture that allows the operators and commanders of navy ships to communicate in a more efficient manner. Because it is a proof-of-concept project, the Naval Collaboration system will be used by Thales as an inspiration to fit their own system.

Contents

Chapter 1

Introduction

1.1 Assignment Description

About the Client Thales Group is a French multinational company that designs and builds electrical systems and provides services for the aerospace, defence, transportation and security markets.

Assignment

The TACTICOS system is one of the most well known naval combat systems of Thales. TACTICOS is made to support the crew of naval ships while on missions.

On naval ships the crew continuously tries to get an overview of the situation, so they can respond when this is necessary. The majority of this work is done from the command centre on the ship. In a command centre operators work on their individual screens to assess the situation around a ship, they work on information based on their roles and expertise. To get a more overall insight in what is going on, a collaboration screen needs to be developed. This screen allows for the combination of information from different sources and can be used by operators to display information from their screen on. Sources include video and data sources and the information displayed on the operators screen. The collaboration screen should be a true source of joined information.

Different operators and the commander have different needs of information, while still working together on the same collaboration screen.

The assignment as given by the client is the development of a proof of concept that will be used to develop a collaboration screen technology for their domain specifically. This proof of concept will then be used to demo the possible advantages of this technology in collaborative operations and for giving briefing on scenarios and then the client will use the results to make the decision of further investing in this technology.

Deliverable

The main deliverable of the assignment is the development of a data sharing architecture and a possible user interface for the collaboration screen/table. The user interface of the terminal, from which the operators on the ship connect to, will also need to mocked, since the client is unable to provide their existing terminals. In a similar case, all data Thales deals with is classified and can thus not be shared. For the purposes of testing the system and making the demo we will also need to mock data to provide a complete and cohesive summary of how the technology could incorporated.

Since the assignment given was to produce a proof of concept for a data sharing architecture, the program produced will not be expected to be integrated with the existing technology they currently have. But it is expected that our methodology takes into account their use cases and

constraints. Our implementation should also be well documented, such that the client could follow the approach to implement their own system should they choose to implement this technology.

Use Case

The following use cases were provided to us by the client to supplement the scenario in which the system will be deployed highlighting its functionality.

The following is a use case is about a scenario where a piece of data with information about a vessel or aircraft(a track) needs to be classified as hostile or non hostile.

1. Trigger:
   (a) The operators selects a track that is of interest on the system.
   (b) The operator sees a suspicious air track on the display
   (c) The operator wants to classify the entity as hostile and consequently notify the commander.
   (d) The operator is unsure about the track being hostile.
2. Consequence:
   (a) The operator requests to share track information on the collaboration table.
   (b) The operator using voice invites a colleague with expertise to the collaboration table.
   (c) The operators use the table and the context sensitive visualisation of the data on the collaboration table for coming to a conclusion.
   (d) The operator reports the decision to the commander.

The following use case is about a rescue operation in potentially hostile territory.

1. Trigger:
   (a) The fleet is on a rescue mission for some medical personal stranded in the sea with wreckage debris.
   (b) The operator wants to suggest a plan of approach to the stranded medical personal.
   (c) There are potentially hostiles in the area which the ship would like to avoid.
2. Consequence:
   (a) The operator requests to share track information on the collaboration table about the medical staff location and how they have drifted.
   (b) A colleague wants to overlay the hostiles' potential location on to the track information.
   (c) The combined share request is added to the collaboration table.
   (d) The operators use the collaboration table to decide the path to take around the debris towards the medical staff minimising the overlap with hostile locations.
   (e) The operator reports the path to the commander.

1.2 Domain Analysis

In this section we introduce the domain in which the assignment is situated in. We then further discuss the important technologies that exist in this field which reflect the design choices and trajectory that is taken by us.

Naval Operations

The system will be used on a naval boat. On such a boat there are many operators, staff of the navy and a commander of the ship. These ships carry out various missions, for instance plan rescue missions, search operations, and offensive missions as well. In these cases there is a requirement for collating many data sources from various sources, different data types and regular structure to produce knowledge. These include geographic graphs, location data with various attributes, time series data, live data that is updated in real time by a radar system for instance among others. Many of these data may be combined to produce a coherent explanation or reasoning in the mission situations and being able to visualise multiple sources in the same view becomes critical for this. Our project of the Thales: Naval Collaboration Screen deals exactly with creating an architecture for this using a collaboration screen.

Hierarchy in Data Access

In a Naval setting there is a hierarchy, in the sense of confidential and restricted data access, that is to be followed among the naval staff. In the system we design there is a cross cutting concern about officers without clearance seeing data that they do not have access to.

In the second meeting of the requirements gathering this was clarified that such access is given on verbal basis. Therefore it is not a requirement to include an authentication system in our system or include prompts for asking the user if the data is safe to be shown in an open collaboration table.

Data Format and Data Sources

There are many different types of data sources that exist in a real use case.

Data Sources

REST - API This is the simplest data sources that our client uses. This is an example of a static data source which only needs to be queried once and is not updated in real time. Examples include schematics of ships, and other factual data like ship weight, height, fuel capacity.

Event Driven Data Sources This is a real time data source that is continuously updated, for example due to a radar system or by a buoy when sea levels get too high. This data thus needs to updated in real time in the system. The radar system for instance may generate live location data, also named a track as mentioned before, for a flying object or another naval vessel.

Data Format

The data type that our client works with is a bespoke bit representation that is proprietary knowledge, but when the data is de-serialised it is similar to JSON serialisable data. For the project we were not provided the scheme Thales uses to be able to use the bit serialisation that they use. So, we were instructed to produce the prototype using JSON serialisable data.

Collaboration

For the purpose of this assignment the definition of collaboration is very important. Collaboration is, as defined by our client, the ability to work on a piece of data with multiple users without creating hindrances for others.

This idea of collaboration as given to us by our client is a central theme when developing our user interface. During our iterative reflection with the client we focused on achieving this for our system.

Stakeholders

Client The client for this project is the company Thales, specificaly the TACTICOS team from Thales. They want to produce this collaborations screen to optimise missions in naval defence and incorporate state of the art technological developments in the domain of Human Media Interactions and Informatics.

The Collaboration Screen that is developed will be used as a complement to the TACTICOS system, and works as a potential enhancement on its capabilities to collaborate.

Operators The users of the product are the Operators in the control room of the naval boats. These include the officers, commanders and other naval staff. These staff have been trained in the use of interactive systems, technical systems and interactive dashboards.

Development Environment

Software Environment The client uses a microservices architecture for their programs and so there is no restriction for the choice for the language of development.

Hardware Environment The client uses the TACTICOS terminals, however for the creation of the proof of concept, the client will not provide us with these terminals or a collaboration table (which in effect will be a large touch display), therefore the development has taken place entirely on personal computers and laptops.

Chapter 2

Implementation Trajectory

2.1 Requirements Discovery

2.1.1 Requirements Capturing

The first part of the project is requirements capturing. As noted in our planning in Chapter 4.2, the first sprint is concerned with requirements capturing. In the Section 4.2 we have already conducted and planned meetings with the clients to iteratively refine the requirements of our project and keep them up to date about our progress. This is also part of our risk mitigation plan as noted in Section 2.6.

Client Information

As mentioned before, it is important that the client is satisfied with the final product. When capturing requirements, it is therefore important to have a good understanding of what the client wants. Since the project proposal itself did not specify any direct requirements, a logical next step was to find a more concrete description of the project.

The weekly meetings that were discussed before serve as the main source for requirement capturing. Before these meetings, the developers prepare a list of questions and follow-up questions that will be asked to the client. This way, a list of hard requirements that satisfies the client can be constructed.

The client, however cannot provide all information needed to fully create the final list of requirements. The reason for this is that in the case of this project, the client is not the end-user of the product.

Other Stakeholders

For this project, we considered pivotal to talk to the end-users of the product. Simply put, there are two main groups of these stakeholders: Commanders and Operators. Both of these groups will have strong interactions with the final product.

The client has agreed on the possibility of meeting with these stakeholders for an interview. Such an interview would allow for a more in-depth description of the system requirements. Mainly, smaller design decisions (such as User Interface layout) could be left up to these stakeholders.

Requirements Formulation

After collecting all the necessary data from the stakeholders and client, we can assemble a final list of requirements. These can be divided in two groups.

Functional Requirements In this text, functional requirements are defined as those requirements that specify a certain function of the product. Presumably both the client and the other stakeholders (end-users) will provide functional requirements. These being functions that the product needs to have.

Non-Functional Requirements Non-functional requirements are those requirements that do not specifically allude to a function of the product, but rather a more abstract metric, such as performance or reliability. Whilst the end-users may have certain inputs here, it is most likely the case that the client will have the most input here. The client, after all, will be much more aware of the internal technical specifications of the product whilst the end-users just see the front-end of it.

Requirement Prioritisation

Not all the requirements we captured have the same priority. Since we still do not know if we can implement all of them, we will need to explicitly define these priorities. This process is called Requirement Prioritisation.

We conduct this phase in collaboration with our client. We ask the client for each requirement whether they consider it essential, important or an extra feature. This way we can ensure that our time is efficiently used to create a satisfying product.

2.1.2 Requirements Analysis

In the interviews with the client and the potential users of the system, we identified the user and quality requirements which after formulation were confirmed with the client. The requirements are listed using MoSCoW priority in the following list along with the client's acceptance criteria. The justification for our requirements can be found in Appendix B.

Requirement	Acceptance Criteria

Must The following user stories are considered to be essential.

No data to be modified on the source	There are no post or put requests initiated from the client or the server on to the data source
Handle track data type with location and a hostile, non hostile and neutral attribute for an entity	The front-end is able to parse a dictionary with enum data type and create a legend with the meaning and the designed application can transmit such data
Handle track data type with location and velocity attribute by showing an arrow	The front-end is able to draw an arrow in the correct direction with the speed annotated on the arrow and the designed application can transmit such data
Handle track data with a combination of multiple attributes	The front end is able to correctly parse a dictionary and draw an arrow with potentially a legend for any enum attribute and the designed application can transmit such data
Handle information data type defined by item name and text description only	The front-end is able to display the text in a human-readable format on to a partition and the designed application can transmit such data
Enter full-screen mode using a gesture for a partition	The user is able to use a three-finger pinching gesture within the span of a partition to make it full screen.

Exit full-screen mode using a gesture for a partition	The front-end is able to handle a three-finger pinching gesture to exit a partition
Resize the partitions on the screen	The user is able to pull on a handle to resize the screen
Dismiss a view from the screen	The front-end must have some button to dismiss the partition (e.g. x button)
Share a Data Stream on to the collaboration screen	The designed application is able to multicast a particular data view on the terminal to other terminals, collaboration wall or collaboration screen.

Should The following user stories are requirements that should be achieved aside from the essential requirements.

Extensible for more data types and describers	It is possible to enter a new datatype into the system without re-writing any code.
Draw on a particular partition on the collaboration screen	It is possible to draw on the collaboration screen by either touch or mouse input
Accept collaboration of another user to existing view	It is possible to accept a collaboration request from another user, this action will open the other user's view on this user's screen.
Add a name to a data view	It is possible to change and/or add the name of a particular data view. This name will then be visible on this data view object until it is removed or changed again.
Accept the display of a view onto the screen	It is possible to accept a view onto the collaboration screen. After this action, the view should be visible on the collaboration screen.

Could The following user stories are requirements that could be achieved, but not considered significant.

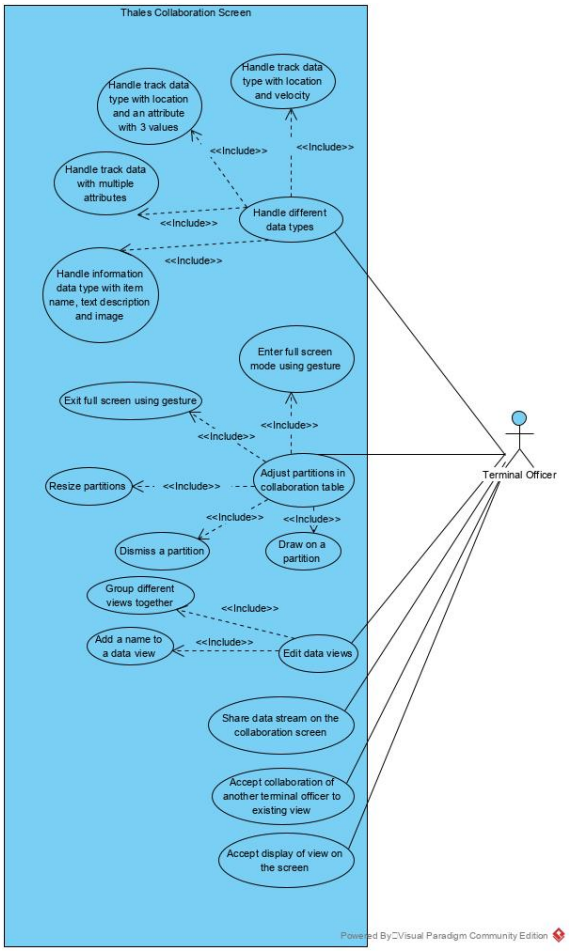
Group different view objects together	It is possible to group several view objects. After this action, these view objects will be either marked or surrounded by a visual indicator of their grouped status.
Handle information data type, item name and text description and image	The system is able to transmit such data types and the front end has a method which allows for parsing such data to show it on a partition.
Compatible with asychronous subscribe data sources	The system is able to gather tracking information solely from a asychronous data source database such as GraphQL database

Wont The following user stories are requirements that are not going to be achieved as it was considered beyond the scope of the project.

Incorporate Open Geo-Spatial Consortium conventions for the track data type	The system is fully designed in accordance with the Geo-Spatial Consortium conventions.

The actions that the terminal officer can perform can be seen in the use case diagram as shown in Fig 2.1.

Figure 2.1: Use Case Diagram



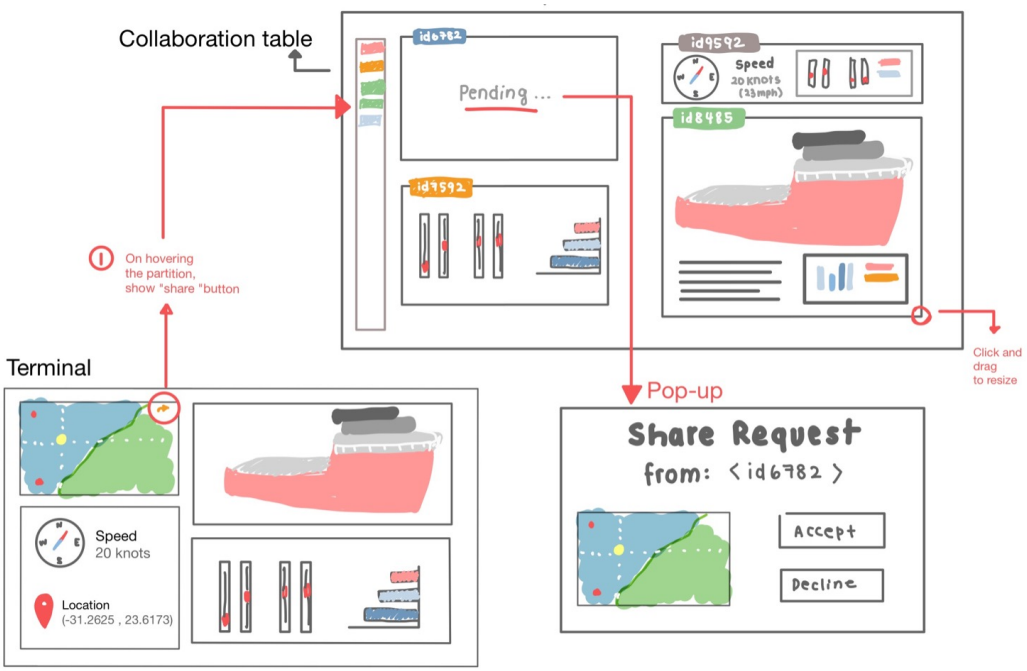
2.2 Design Phase

Design is at the core of any implementation trajectory. From experience, we have learned that creating a complex and realistic design is essential for an efficient implementation process. Without such a road map, it quickly becomes difficult to manage resources in larger projects such as this one. For the back-end design, we will mainly rely on UML (Unified Modelling Language)[1] diagrams and lo-fi and hi-fi prototypes for the front-end and user experience.

Low-fidelity prototype Before we can create a front-end, we create a prototype. The reason for this is that these are easy to create and adapt, whilst a full implementation is much more difficult to change. A prototype, however, still allows us to see what the design will look like. We have used a simple sketching software to create a lo-fi prototype (Fig A.1) that matches the requirements that we found. When we are satisfied with the result, we can implement it.

The current lo-fi prototype as shown in in Fig A.1 is due to the meetings we have had with our clients. We identified that there are many different data types hence each data type will have their own user interface descriptor function for consistent and contextual sensitive information visualisation and the way they need to be forwarded to a collaboration table is to add them to a queue (visible on the left side of the screen on the collaboration table) and when the operators are on the collaboration table they can choose to display it. This also prevents spamming or crowding of the collaboration table.

Figure 2.2: Lo-fi Prototype of product architecture



High-fidelity prototype After getting some feedback on the lo-fi prototype, we developed a hi-fi prototype (Fig 2.3) using Figma [1] in hopes of getting an opportunity to proceed with user testing (Link to the prototype [2]). This hi-fi prototype contains basic functionalities and interactions of the application such as sending a data request from the terminal and accepting the request in the collaboration table. Our initial plans were arranging a user testing session with one of the stakeholders in Thales. However, due to time constraints and logistical reasons with the terminals and availability of the officers, we were unable to perform them. Therefore, due to the limited time of the project we have chosen to forgo this evaluation which was a decision supported by our client as their key focus is the Data Sharing Architecture.

Data Sharing Architecture Since our back-end will be written in object-oriented Python, we have chosen to use class diagrams to visualise the flow of data. Another relevant aspect of the design of the back-end are the sequence of actions that can be executed on it. Sequence diagrams provide this functionality. Finally, use case diagrams are another form of diagrams that we will need. These will help us conceptualise the actors (users) and their specific requirements. The design phase of our project enables us to use the design by contract paradigm which ensures user requirements are met [2].

We also present a schematic diagram in Fig 2.4 to explain the data flow and to explain where exactly the Data Sharing Architecture is situated in the existing system.

[1] https://www.figma.com

[2] https://www.figma.com/proto/gf4YnF4XIQ68GlMDJ6Jzst/Naval-Collaboration?node-id=10%3A218&scaling=min-zoom&page-id=0%3A1&starting-point-node-id=10%3A218

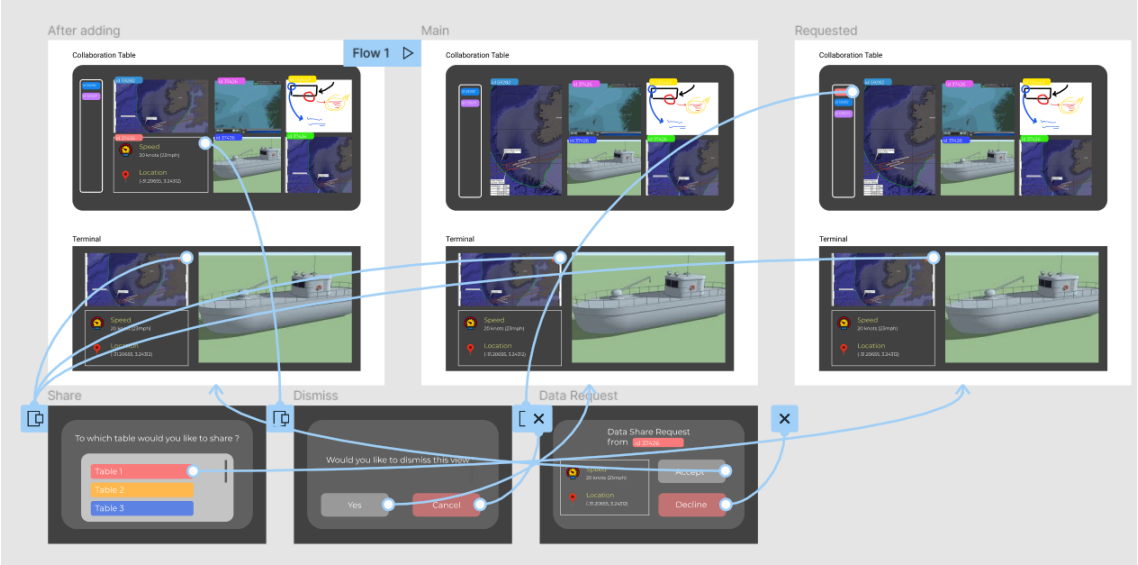
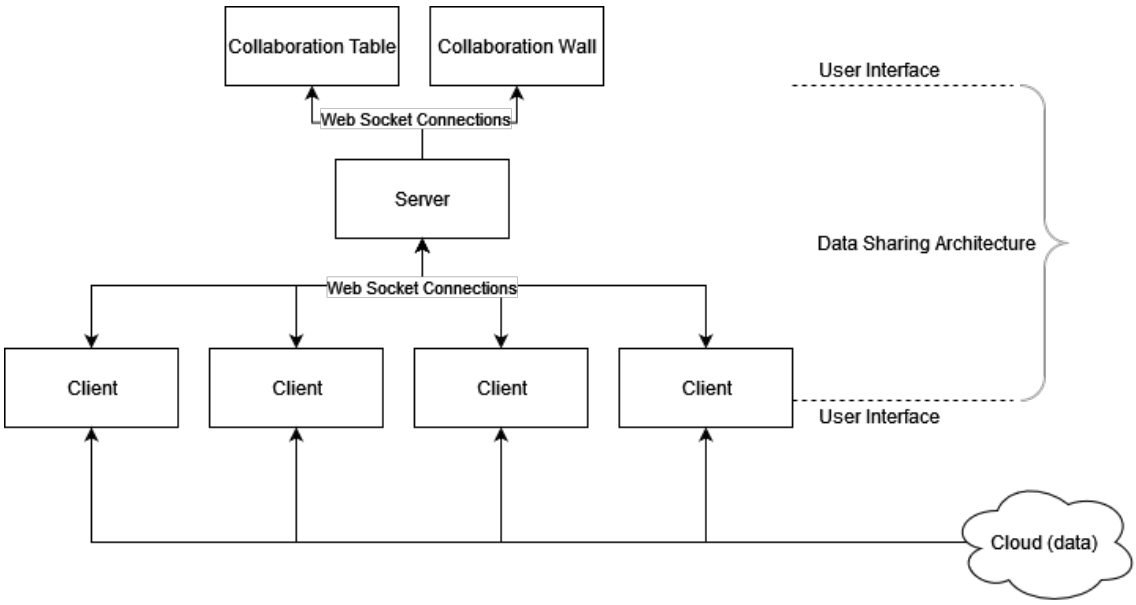


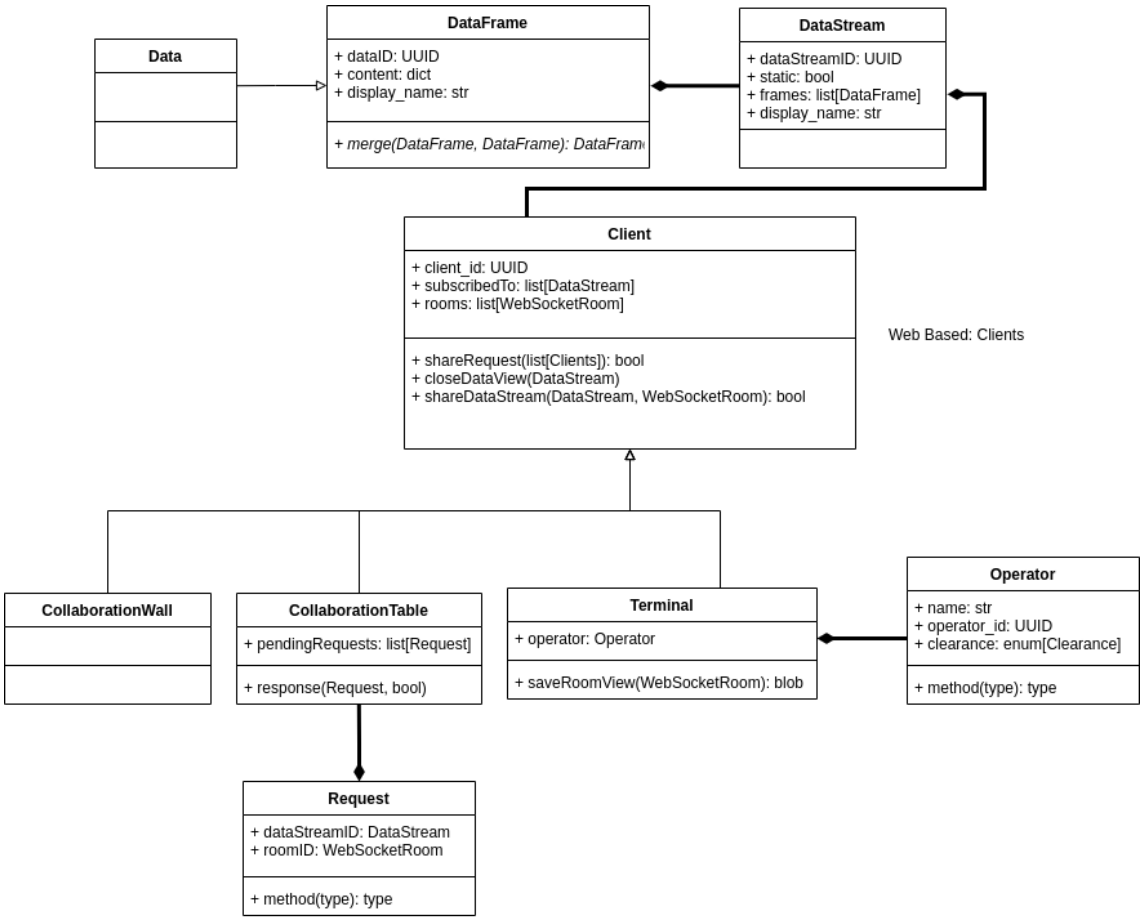
Figure 2.3: Overview of Hi-fi Prototype

Figure 2.4: Data Flow Schematic



We also have decided the abstractions for the data flow that are shown in the class diagram shown in Fig 2.5.

Figure 2.5: Class Diagram Back End



2.3 Test Planning

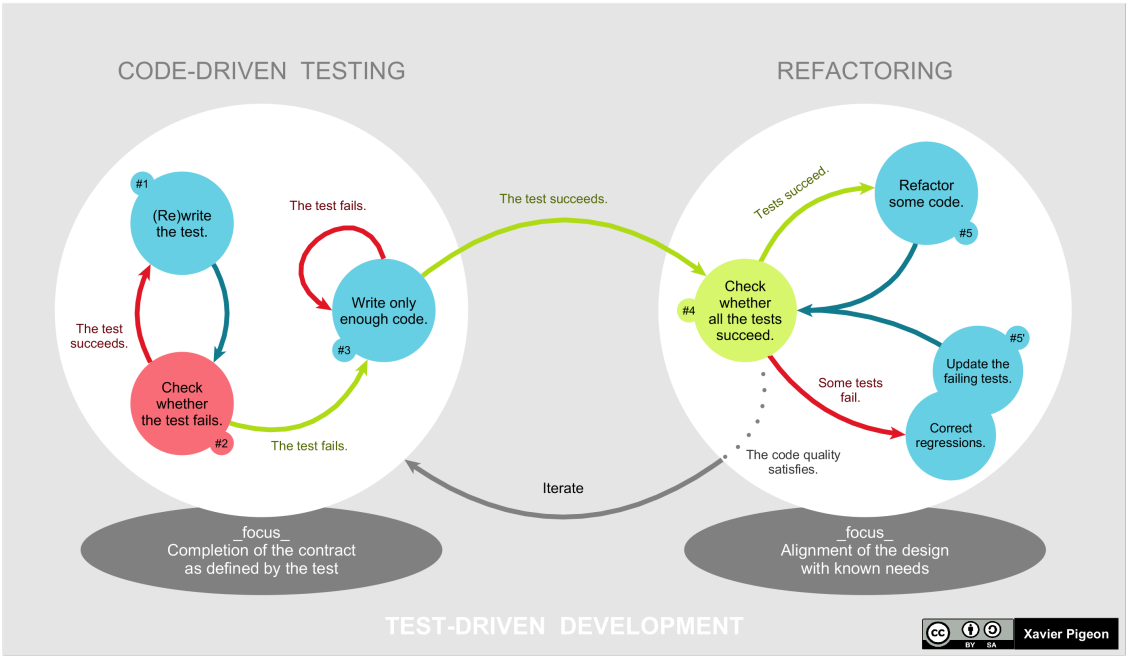
The proposed system essentially has two parts, the data sharing architecture that allows collaboration on the collaboration screen and the user interface used by the users on the collaboration screen. Keeping the proposed system in mind, we propose the following test plan.

2.3.1 Data Sharing Architecture

Unit Testing During the implementation of the data sharing architecture, we will use a test driven development approach [3]. The process is described in the Fig 2.6. It was shown that using TDD meant writing more tests and in turn programmers who wrote more tests tended to be more productive [4]. This also introduces the idea of applying design by contract [2] which ensures that requirements of the user are met.

Integration Testing Since the data sharing architecture is composed of multiple individual components, from gathering user input, transmitting it to the collaboration screen and gathering inputs from the collaboration screen again, it is essential that all the components are compatible with one another and work in harmony. The integration testing will be done using automated tests in a virtual environment which mimics the different interactions and will be performed when the data sharing architecture is complete and functional.

Figure 2.6: Test Driven Development



2.3.2 User Interface

End to End Testing This testing will be similar to the Integration Testing plan, but this time we will use the developed user interface instead of mocking the interactions with the program code. This will be done manually by the developers.

System Testing The program to be delivered will work on multiple systems: a console used by the operator and a central collaboration screen. The delivered programme therefore needs to be tested on these devices.

Physical Devices The terminals and collaboration screen that will be provided by the client or personal devices will be used for the final system test that will done at the end of the development phase.

Usability Testing We will have a meeting with product manager of the TACTICOS System of THALES, to generate usability test results. This will be done before the final product completion to ensure necessary changes and feedback can be incorporated.

2.4 Product Implementation

User Interface For our implementation of the user interface, we will produce a web-based interface. For our web-based interface we will employ a front-end framework, namely Svelte, since it allows for ease in development using its many utility features. It also enables us to rigorously test the front-end using automated tools. Using a web-based interface also allows seamless integration with any device such as a tablet or an operator terminal since the development of an Android, iOS, Linux or a Windows application is not required and is usable with any device which has Internet connectivity and access to any browser like application. For our test plan please refer to Chapter 2.3.

Data Sharing Architecture For the data sharing architecture, which is essentially the back-end of our web-based user interface, will be completed in Python language version 3.10. Python

was chosen due to the existence of many libraries that are relevant for our project and the general team experience. The Data Sharing Architecture which will constitute a server, which will be a Linux-based environment, as this allows for the use of Python hence it can be used without integration issues with other services. Our client has also indicated that since they use a micro-service paradigm for the many different services they have, the use of Python as a programming language is not an issue.

Using the design of the system made, the development of the data sharing architecture is done using a test driven development. This is covered in detail in Section 2.3.1.

General Activities over the Development Cycle During the development of the product, we will be documenting our project so as when the project is handed over to the client, their technical team have a clear idea on how to use the product and make the relevant changes to suit their production environment.

2.5 Technology Choices

In this section we cover the choice of technologies that we have chosen for the implementation of the different parts of the system and we justify the choices we make.

2.5.1 User Interface

We have the freedom of choice from our clients for how to implement the user interface and we have chosen to use a web based interface. We chose not to use other languages such as C#, Java or python due to restrictions on the hardware which will be used to run the user interface, it will need to work on tablet, terminals and televisions, but since they all have browser functionality this was the ideal choice. The technologies and frameworks that we will use and what role they play are given below:

1. Svelte [3]: This is a framework for creating the User Interface. It simplified away many tedious tasks that exist in plain HTML such as update on change, iteratively adding contents onto a an HTML page and most importantly eliminates code smell like Spaghetti Code for a more developer friendly experience. There are many such frameworks which achieve the same results with difference in syntax. Svelte was chosen due to the small learning curve that it exhibits.
2. Tailwind [4]: This is a Cascading Style Sheet add-on which provides many inbuilt CSS classes ready to use. This will save time in development and enables rapid prototyping which are not only functional but aesthetic.

2.5.2 Data Sharing Architecture

We were also given the freedom of choice from our clients on how to implement the data sharing architecture. Since our client uses a micro service paradigm the choice of language is free and does not need to be constrained by existing technologies. There are two parts to the Data Sharing Architecture the

1. The client that runs on the terminals and the collaboration wall and collaboration table.
2. The centralised server which handles the collaboration requests and produces views for the participants of a collaboration.

To implement both parts we have the following choices:

1. JavaScript ES6 [5]: The role of the web based programming JavaScript is to create the user interface interactions, such as displaying the tracks data, map data and most importantly it will play the role of the connection from the user interface to our data sharing architecture

[3] https://svelte.dev
[4] https://tailwindcss.com
[5] https://www.javascript.com

server. In the schematic diagram shown in Fig 2.4, this will exist in the clients, the Collaboration Table and the Collaboration Wall. The reason we choose JavaScript as opposed to WebAssembly [6] is the existence of other libraries that we will use in the project are also written in JavaScript. These libraries are listed below.

(a) Websocket: This is an inbuilt library in JavaScript ES6 which facilitates the creation of the web socket connection on the client side and separates the protocol concerns as written in RFC 6455 [5].

2. Python [7]: Our centralised server will be implemented in Python. We use python as opposed to other languages like Rust, Java, C# or dot Net programming languages is the familiarity with the programming languages and ease of incorporation with different hardware as python language can be used on any linux server with minimal installation steps.
There is also a technical reason why python was chosen, with python it is possible to define different data types dynamically, which essentially means that there is a higher flexibility in terms of the actual data that our server can handle without needing to program a new class from scratch when a new sub data type is introduced allowing for extensibility. This comes with the trade off of speed.

2.6 Risk Analysis

For this project, there are a couple of risks that could occur. In this chapter, we will go over the risks that can we can foresee as of now and how they can possibly be mitigated.

Risk sorted by likelihood and consequence

#	Risk	Likelihood	Consequence
1	The needs of the system are very complex	Medium	Medium
2	System is not integrable with the existing system	Low	Low
3	The product fails acceptance testing	Low	Low
4	System fails essential tests	Medium	High
5	Team has miscommunication	Medium	Medium
6	Refinement of the goals and needs later in the project	Medium	High

Risk Analysis and Mitigation

The best way to mitigate risk 1 would be to talk to clients, get their expectations then try to work with those expectations. Often try to get feedback from the clients, including what they like and do not like about the system. Upon receiving the feedback, alter the system if necessary, then repeat.

To mitigate risks 2 and 3, we will discuss this in the requirement exploration meeting about technical details and quality requirements. This will allow us to plan for this in the design sprint which will help us integrate it with their existing technologies. The consequence level is low however because the integrability of the system was not required by the client.

The way we would mitigate risk 4 is by adopting the test driven development. This is discussed in earlier sections [4.2, 2.3].

We think that the best way to mitigate risk 5 is by having daily meetings and follow up on important decisions that are made. We also plan to use Discord groups and plan to work at least once a week in person to ensure no one feels isolated from the others.

The way to prevent risk 6 was done by waiting for approval on the proposal. The approval on the project proposal was received in the third meeting with the clients and so we have an understanding on what is to be delivered and when. Obviously where in the proposal there is some room for interpretation these will be cleared when such issues arises should an issue arise.

[6] https://webassembly.org
[7] https://www.python.org

Chapter 3

Detailed System Implementation

3.1 Data Simulator

Choices

One of the main challenges we encountered in this project was the lack of data to test our system with. The simplest solution would be to write out some small sample data manually, but it was clear that this was unfeasible for larger, more realistic, tests. After some discussion with the client, it was decided that the most viable solution was to build a simulator.

Data Format JavaScript Object Notation, or JSON for short, was chosen for transmitting simulated data over the network. JSON was selected because of its speed, its ease of use with JavaScript, and its overall simplicity. The simulated data will be represented as a JSON list of JSON lists. As shown in Figure 3.1, the inner lists each represent one 'time frame'. This simply means that every inner list contains all relevant data at one point in simulated time. Each JSON object in the inner lists represents an object, like a ship, an plane or something else.

Isolation of the Simulator It is important to note that the simulator should, by design, be separated from the rest of the system. This separation simulates the way in which the operator terminals are also not directly connected to the sensor systems of the ship; the ship's cloud server is the only way in which these two components can share data with each other. Because of this separation, the implementation of the simulator was not bound to any of the other frameworks that were chosen. The language chosen for the development of the simulator was Python nonetheless. The Python programming language works well with JSON data.

Language The simulator is built in Python. Besides the aforementioned reasons for choosing Python, the reason to do so in the simulator was mostly related to the considerable number of good Python libraries that are available for such a purpose.

Generated Data

The structure of the JSON data was previously decided. The next step in the development of the simulator was determining what actual data had to be generated. Each time frame in the simulator will be represented by a JSON list of all the objects and their updated data at that time. An example of such a time frame can be found in 3.4

Position The simulator will simulate the movements of various objects around the ship. Position is a core property here. There were multiple options for storing such coordinates. The primary two that were considered were a grid system and latitude/longitude.

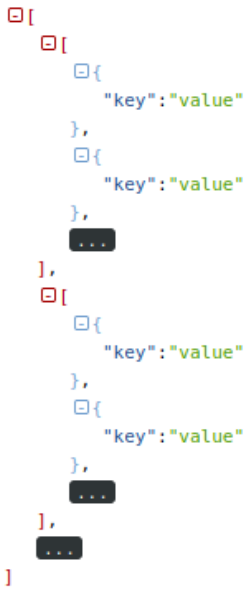


Figure 3.1: JSON format as a String Representation

Cartesian System In a Cartesian coordinate system, the coordinates of an object are determined by its x and y position in a grid. This means all positions are relative to the zero, or origin, of the grid. As the grid itself is by nature rectangular, such a system is indeed useful for representing objects on a rectangular screen. In fact, grid coordinates are already returned by the simulator itself.

It should also be noted that a Cartesian coordinate system is also preferable for calculating the movements of simulated objects. Simulated objects have a few attributes such as position, velocity, acceleration and a heading. All this data, as well as the grid, are represented in SI-units. This means calculating any kind of positional change is most easily done in the Cartesian system.

The problem with Cartesian coordinates however, lies exactly with their relative nature. In the context of a ship's bridge, this relative position means nothing. Although the simulator internally works with this system, it will have to output a more universally applicable type of data.

Lat/Long System A positional representation more commonly found on ships is the Lat/Long system. This system has the advantage being able to absolutely represent any point on earth. With the fact that this is the default system for representing position comes a compatibility with maps and many software programs that in some way make use of position.

Translating Coordinates It was briefly mentioned before that the simulator initially works with grid values. However, the Lat/Long coordinate system was chosen for the storage of actual data. This meant that the original grid data would have to be translated to Lat/Long coordinates. This was achieved with the simple code in. As shown in Figure 3.2 `cls.RADIUS` is the radius of the earth. The method `circum_of(r)` simply returns the circumference of a circle with radius r, and `circum_at(lat)` returns the circumference of the earth at a certain latitude.

Other Attributes

The rest of the attributes are indeed less complicated, and can be summed up in this section.

Name Simulated objects can have a name.

```
@classmethod
def geo_repr(cls, coord: Coord2D):
    """
    :return: Represents a certain 'meter' location
    :param coord: in lat,long format
    """
    m_zero = cls.meter_repr(cls.zero)  # Meter representation of zero
    lat = ((coord.x + m_zero.x) / cls.circum_of(cls.RADIUS)) * 360
    long = ((coord.y + m_zero.y) / cls.circum_at(lat)) * 360
    return GeoCoord(lat, long)
```

Figure 3.2: Method for translating Cartesian coordinates to their Lat/Long representation

Type Simulated objects always have a type. In the current implementation, this is the Python class to which they belong in the simulator. This can e.g. be:

```
"<class 'models.mobile.SeaShip'>"
```

according to Python's default class notation.

Speed The velocity of the simulated object.

Acceleration The acceleration of the simulated object.

Classification A ship can be classified as friendly, neutral, or enemy.

Angles Objects in the simulator move at a certain angle. It is important to note that this angle does not represent the actual alignment of the vessel in question, but rather the direction in which it moves. Ships are simplified here in the sense that they are represented by one angle. This means they cannot be angled vertically in any way. Airborne vessels, however, have access to an extra rotation variable. This way a movement in a three-dimensional space can be simulated.

Timestamp Timestamp at which this measurement was taken in simulated time

Graphical User Interface

Usage To facilitate the generation of data, a GUI was deemed useful. The GUI itself was kept simple to limit its development time. It offers the user the ability to create an arbitrary amount of objects in any medium (sea, land or air). Then the user is given the option to set various parameters. The speed, as well as the base acceleration, can be entered here. The object can also be given a name. All objects are represented as a blue square. Clicking on such a square enables the editing of the previously mentioned variables.

By right-clicking an object, the user can select a particular object. Then a path, consisting of green dots, can be formed by simply left clicking. Each green dot represents a way-point. The object will cross each way-point in its path.

When the user is satisfied with the scenario, it can be stored (ctrl + s) and run (ctrl+ r). This will generate a JSON file with the previously described data for all time frames.

A simple scenario might look like this. 3.3

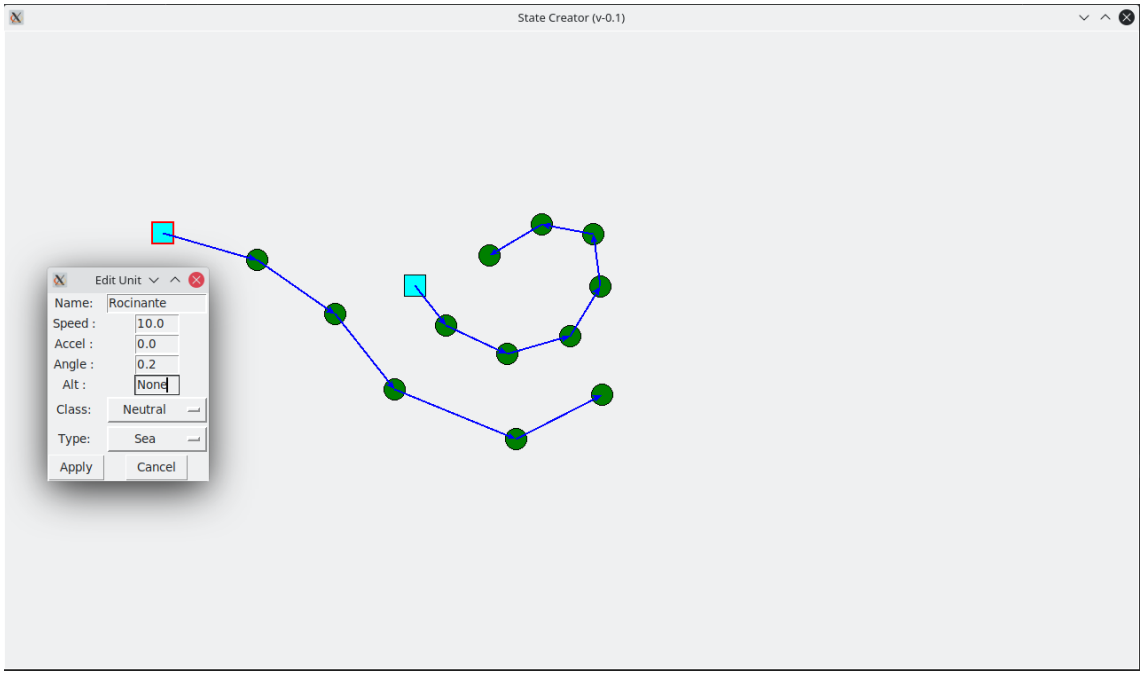


Figure 3.3: Simple GUI of the simulator

Possibilities Currently the main function of the GUI is to generate data and output it to a database. The simulator, however, can also be used in real-time. This means that instead of dumping the complete results in a database, it can instead take as long as the actual scenario would take. It can then periodically send updates. This way the simulator can act as a more realistic database on a ship, where the data is constantly updated as the situation changes.


```
[
  {
    "name":"Rocinante",
    "type":"<class
    'models.mobile.SeaShip'>",
    "speed":10.0,
    "acceleration":0.0,
    "classification":"Friendly",
    "coordinates":{
       "lat":0.002050453261494706,
       "long":0.0025396842167408084
    },
    "angles":{
       "xy":0.4510696559885235
    },
    "timestamp":0
  },
  {
    "name":"Medina",
    "type":"<class
    'models.mobile.SeaShip'>",
    "speed":4.0,
    "acceleration":0.0,
    "classification":"Neutral",
    "coordinates":{
       "lat":0.004835832139346198,
       "long":0.003686499140112562
    },
    "angles":{
       "xy":0.8231161158643125
    },
    "timestamp":0
  }
]
```


Figure 3.4: JSON representation of one simulated time frame

3.2 Data Sharing Architecture

Technology For the back-end of this project we have chosen to use python, we have chosen this language because in addition to python having extensive libraries and thus being flexible in its uses many of the group were most proficient in python.

In this project there are 3 different parts to the system: servers, terminals and screens. Terminals are the computers of the operators, screens are either a collaboration wall or a collaboration table and lastly servers are just devices that run as a hub for the system.

The connections between the server and terminals or screens are web socket connections, we have chosen web sockets to make it easier for Thales to adapt or change the front-end to something else while the server stays compatible.

3.2.1 Protocol Realisation

Protocol Schema

If a terminal or screen wants to join it sends a connect message to the server specified in the **proto_msg_fmt** file. In these connect messages the operator_id and operator_alias are included in case of the operator, this was done in order to mimic the TACTICOS system authentication system for operators. For the screens similar fields exist, these are for identification of a screen since in a meeting it was said a fleet or ship could have multiple walls and tables on which should be able to be collaborated on.

When a screen joins or disconnects from the server it will send out a **screen_broadcast_message** to all terminals connected, this message contains a list of all screens currently connected so the front-end knows which screens data can be sent to. The same message will also be sent to any new terminals connecting to the server. After a screen/terminal has joined any messages sent to their web sockets will be handled by their respective handler.

To start sharing data on the collaboration table/wall a terminal will send a **Colab_request_msg** to the server. The server will check the various fields and values of the message to ensure everything needed is in the message and all values are valid. After the request has been checked the a reply to the terminal with the room key for the collaboration, a similar message to the request, with the addition of the room key will be sent to the screen that should display the data.

When the terminal operator decides it no longer wants to share certain data or if the person at the screen has finished with a piece of data they can decide to close the collaboration. This is done by the screen or terminal sending a **colab_close_msg** to the server who will forward it to the other party in the collaboration.

When a screen or terminal disconnects the server will ensure that any collaborations that are still active will also be closed by sending a **colab_close_msg** to the other party for each collaboration still active. We implemented this to ensure that in case a screen or terminal abruptly disconnects due to crashing or losing power, the connection is cleaned up nicely without error messages being sent or received.

Extensible Protocol Ideology and Limitations

We had a clear ideology for the protocol, since our project is a proof of concept and not a full product our protocol needs to be extensible. Thales is able to easily add extra protocol messages to the server/client handlers. It is also possible to add additional data types to the system with one limitation, namely at the moment our system only works with data types that support base64 encoding. If other data types need to be added the data handler will need to be swapped out for a new one.

3.3 User Interface

As mentioned previously in section 2.5, our front-end work is built with Svelte. With this framework, we were able to implement a rough skeleton of the platform that handles basic interactions of the user. After various meetings with our client, we were able to make some design choices that fulfill the client's needs and potentially enhance the interaction between the users and the system.

General Theme - Dark Mode

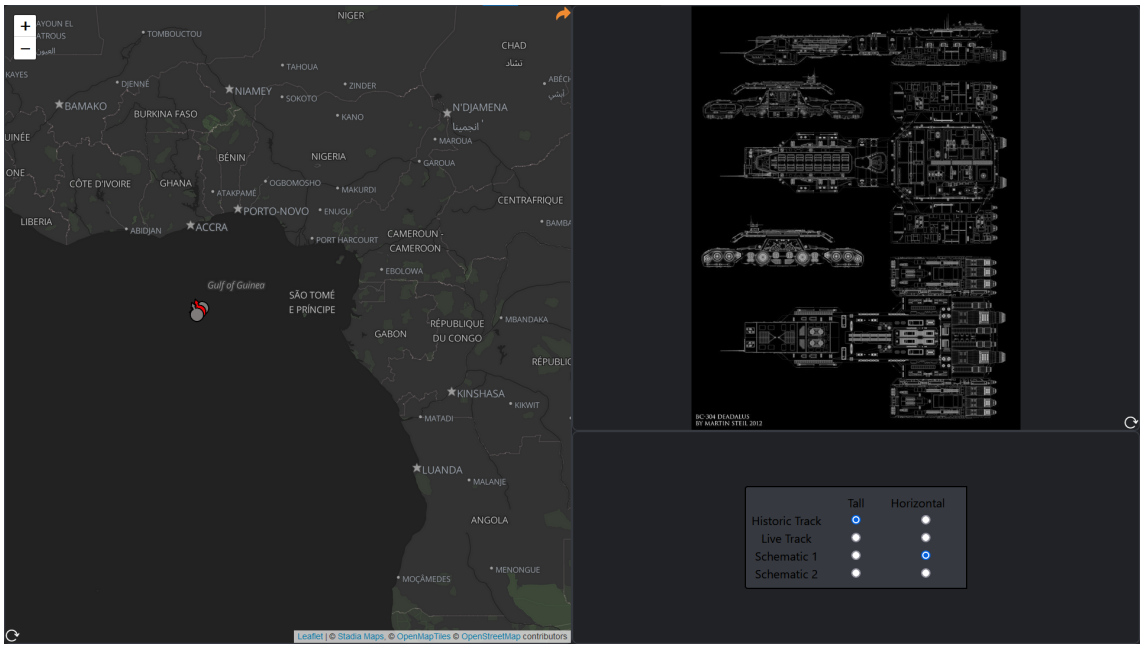
One of the most significant feature specifically required by our client was to have a dark theme. Since most of the operators and the commanders are likely to sit for several hours looking at the screen, it was crucial to design every component of our front-end application to fit low light settings to mitigate the irritation on the eyes. The inspiration of the design was taken from Discord's dark theme, accentuating several components such as buttons with more vibrant colours.

This is visible in the coming section which highlights each feature that was implemented.

3.3.1 Mocked TACTICOS Terminal

Mocked Chart Explorer In the TACTICOS system, one of the integral part is a chart explorer. This is mocked as well, seen in Figure 3.5 at the bottom right. A chart explorer is an explorer which allows an operator to choose a map, ship schematic or other data to show on the TACTICOS system. A track as defined as the live location on a map of singular or multiple objects moving or stand still on the map.

Figure 3.5: Mocked Chart Explorer



Sharing Request The operator can share their partition to the Collaboration table by clicking on the orange arrow located in the upper right corner of the partition (shown in Figure 3.5 in the map partition). This action will show a popup box element (also known as a modal) asking the operator to name the request and to indicate to which collaboration table they will share their partition to (shown in Figure 3.6. Once the partition is shared successfully, the operator will get a green (successful) toast notification as well as a red border on the partition (shown in Figure 3.8) indicating that the partition has been shared successfully.

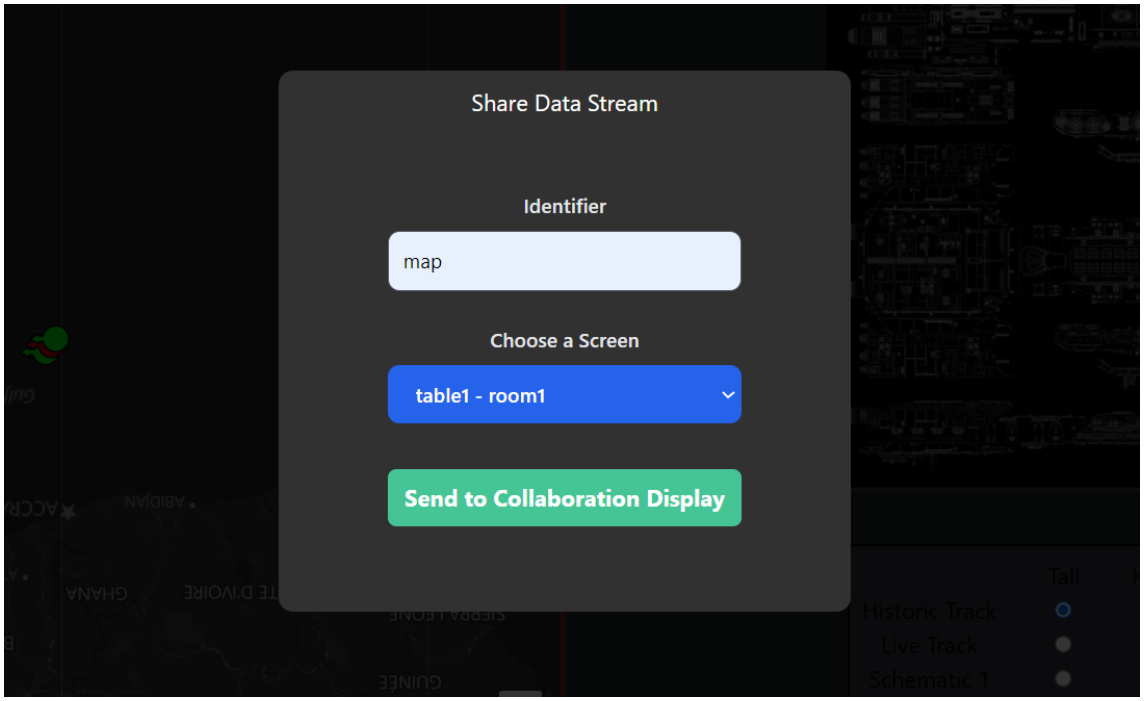


Figure 3.6: Modal to send request to collaboration table

Forwarding Data rather than Data Address Initially there were several options for sharing data from a terminal to a screen. The simplest method would involve sharing the location of the data on the central cloud server. The recipient should then be able to access this data. After more careful consideration, it was revealed that this structure has some severe downsides.

When sharing the source of the data, both the recipient and the sender implicitly need to have access to the same data object. This can result in concurrency issues in case the data is modified. In fact, modifying the data at all be a problem by itself. When one side makes changes, these can be confusing for all other users of the data.

Therefore another method of sharing data was elected. In the current system, not the location of the data, but a copy of the actual data is forwarded to the recipient. This way all recipients have an independent copy of the information, and can make adaptations without causing the previously mentioned concurrency problems.

3.3.2 Collaboration Table

Partition and Components

The partitions in the operators' screens allow the operators to visualize the different data types that they retrieve from the cloud storage. Since the client could not disclose the actual data / data types they store, we implemented a simulator (more on section 3.1) that mocks the possible data types such as location and track history of the ship. These data types are shown in a single map component (shown in Figure 3.8). Additionally, to allow transmission of a flexible range of data types, the front-end application supports types such as static images and text components to be communicated over web sockets to the server.

Modularity of Components

For the project it is important that more components can be added without entirely changing the code base. This ideology was incorporated in our program by design. Hence, the component that were added at later data for extending to systems that our client wishes to added can be done with ease.

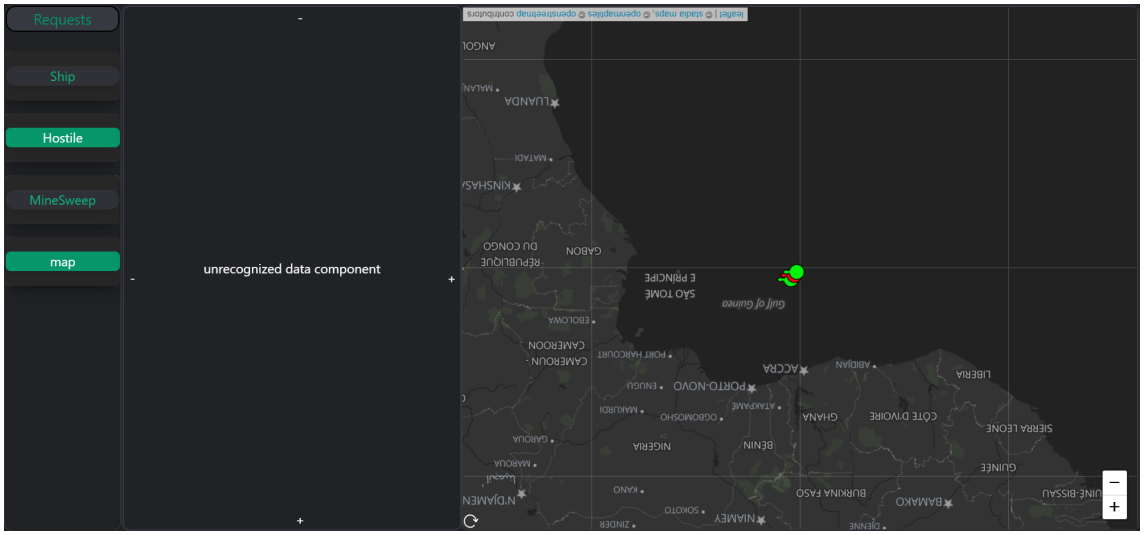


Figure 3.7: Collaboration Screen with rotated map component

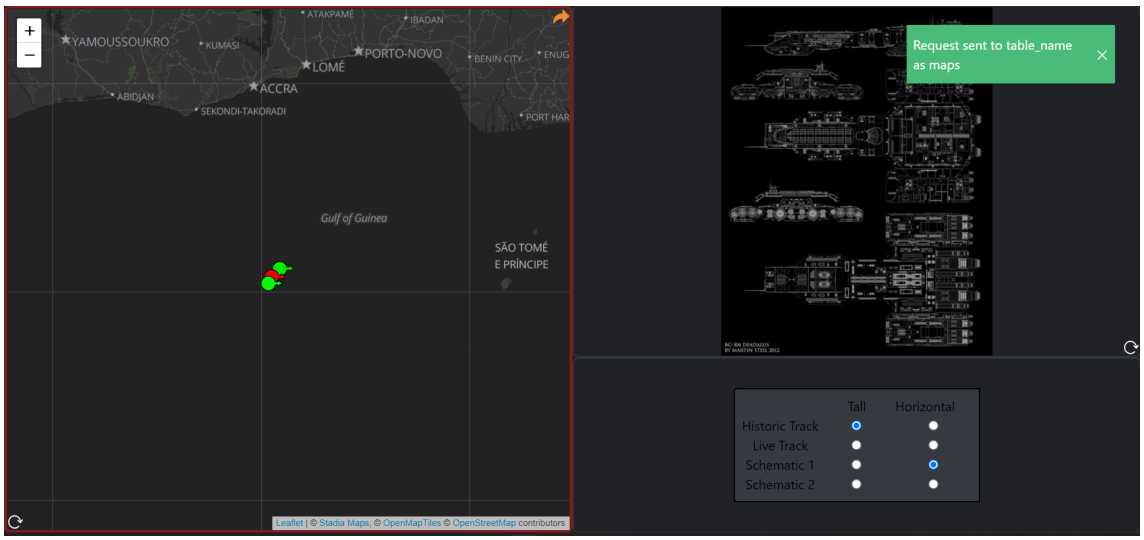


Figure 3.8: Operator's screen with map component shared to Collaboration Table

In our proof of concept, this was adopted using our choice of framework called a component selector. So once we can define a data type, we can then add it to this component selector framework. This is something that can be expected to be part of the final product if pur client chooses to implement it.

Rotation and Resizing of partitions

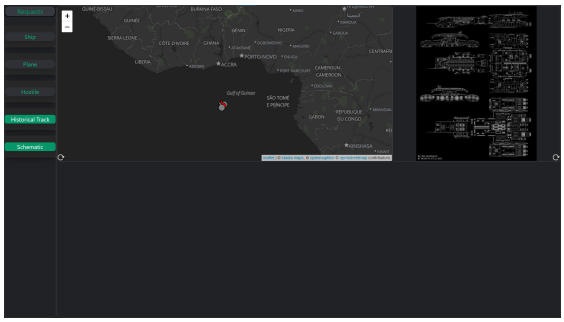
Another requirement given from the client was the rotation of the partitions. In order to collaborate efficiently, the operators should be physically able to see the same partition in different standing directions around the collaboration table. For this, a rotation button has been set in the lower left corner of each partition, allowing the partition to rotate 180 degrees. This feature allows the operators to collaborate without having to move around the table, but rather do a simple click motion to see the same partition.

Furthermore, the partitions have a resizing feature on each edge. The upper and left edges have a "-" sign indicating the partition to shrink in size in the respective directions, while the lower and right edges have "+" sign to do otherwise. With this feature the operator can rearrange their view

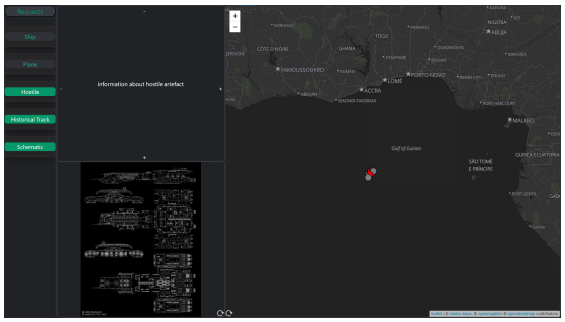
of the partitions. Note that while the initial design (Figure A.1) proposes the partitions to resize by dragging the lower right corner, but due to usability testing it was revealed that it is far more intuitive to have a grid like resize of the partitions to ensure visual bugs.

The aforementioned features are depicted in Figures 3.9.

(a) Components Initially in Smaller Size



(b) Components Resized



(c) Map Partition rotated only

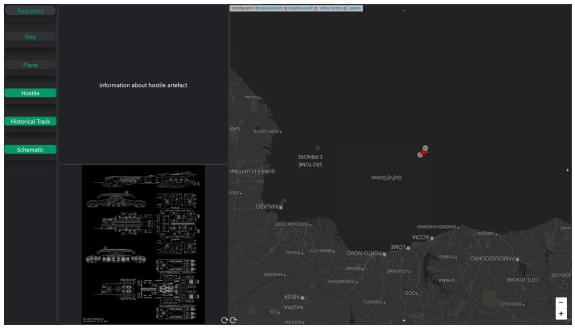


Figure 3.9: Tangible Partitions

Requests

Although the general structure of the operator's screen and the collaboration might look similar, there is an outstanding additional column found in the left side of the screen on the collaboration table: the request queue. The request queue contains all the different requests sent by different operators that wish to share their partitions to the collaboration table.

Toggling and Removing requests On the collaboration table, the requests of the partitions can be toggled on / off by simply clicking on the requests. When clicked (toggled on), the requests turn green, and otherwise remain dark grey (shown in Figure 3.7). If the operator/s no longer desire to share the partition, they can swipe the request (touchscreen friendly) from the queue to remove the collaboration from the table. This will be indicated in the operator's screen as well, with the disappearing of the red border indicator. The swiping is shown in Figure 3.10.

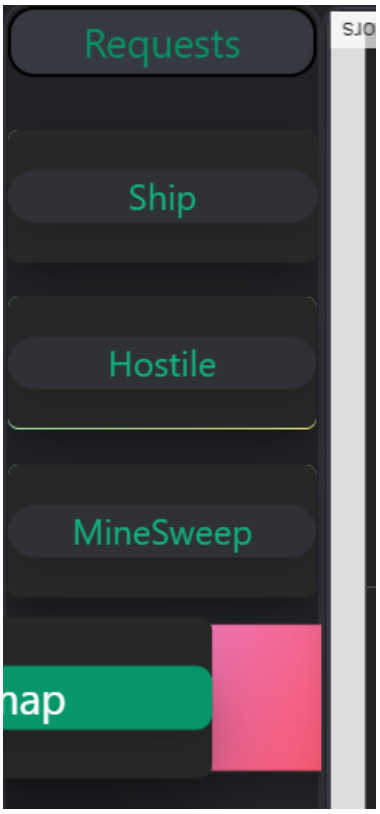
3.3.3 Real Time Data Updates

Svelte Stores

One of the most important methods to handle state management in the Svelte framework is the use of Svelte stores. Stores provide the ability to share data without explicitly passing it up or down components. [6] Besides making the Svelte framework more reactive, this feature can be used for ensuring the real-time updating of data.

A writable Svelte store provides an update and a subscribe method. The former can be used to modify the value held within the store, and the latter ensures that each subscriber is notified

Figure 3.10: Swiping to Dismiss a Collaboration



whenever the value is changed. This simple system ensures that all displayed data is updated in real-time as the raw data below it is updated.

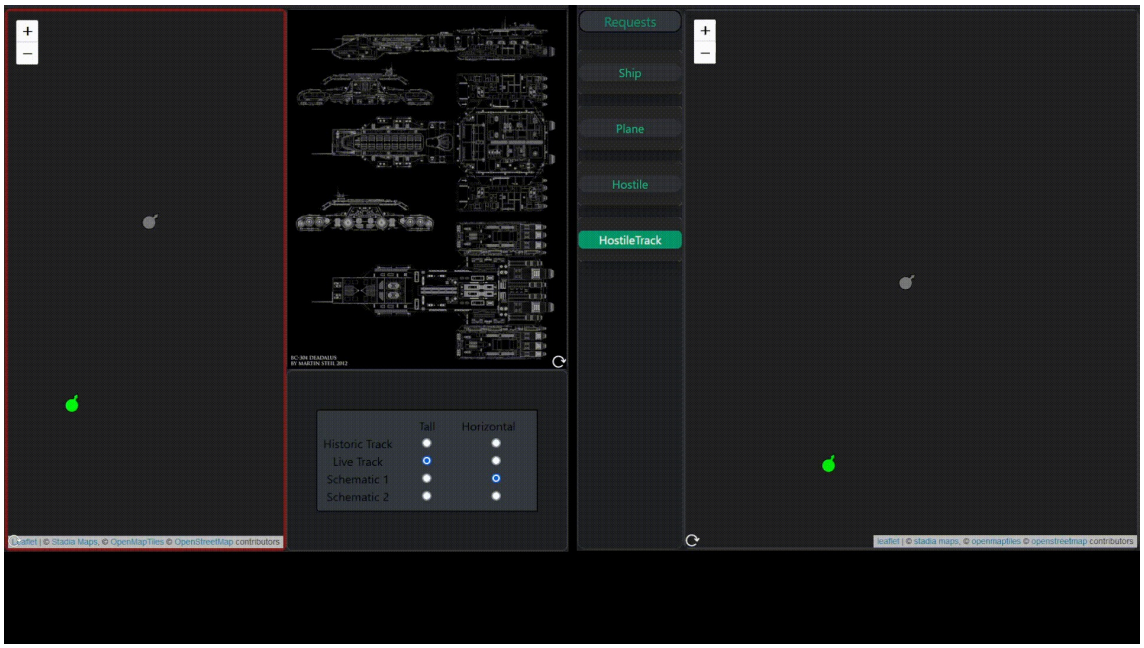
Websockets

Websockets were chosen to connect the various system components to the python server. This design decision was made because websockets provide a more efficient way of transmitting data than other architectures such as REST. The main reason for this is that websockets do not require the request overheads that would typically be present in other architectures.

Another reason for using websockets is that this architecture has equivalent implementations for non-web-based front-ends. This way the system is not tied to the web-based front-end that it currently has. This makes the back-end of the application, which is the most relevant part of the system, much more flexible.

Real Time Update was shown in the Demo and the Final Presentation (Figure 3.11).

Figure 3.11: Real Time Update (.gif file won't appear to move in .pdf file)



3.4 Integration

As shown in Figure 2.4, the connection between the clients (operator screen, collaboration table screen) are handled by the web socket connection. For each connection made between the server and the client, a new web socket object is created and stored in Svelte stores. Stores are "global data repositories that hold values" (MDN Web Docs, n.d.)[6]. Using this feature, we were able to store the web socket instances and manage the communication between operators and the server. In addition to sending messages across, the web sockets are also responsible to store the requests from the operators to svelte stores.

3.5 Testing Results

3.5.1 Approach

The test plan for our project was followed and the results and the details of the tests following the critical discussion of our implementation is given in this chapter.

3.5.2 Unit Testing

Unit Testing was carried out on the Data Sharing Architecture. The following test cases with the results are given in the following table.

Description	Registering a Collaboration Table with the Server
Acceptance Criteria	(a) Number of Displays on which to share content is incremented by one. (b) The Screen Name and Screen Location match as entered by the operator. (c) Missing any field sends a meaningful error message.
Completed	✓

Description	Registering an Operator Terminal with the Server
Acceptance Criteria	(a) Number of Operators connected to the server is incremented by one. (b) The Operator Id and Operator Alias for each connected operator is same as entered by the operator. (c) Missing any field sends a meaningful error message.
Completed	✓

Description	Registering a Display after an Operator registers
Acceptance Criteria	(a) The operator receives the new display that joined with display name and location as entered. (b) The display id that the server assigns to the display is also sent to the operator for future collaboration and this is the same as the id as registered in the server.
Completed	✓

Description	Registering an Operator after Registering Displays
Acceptance Criteria	(a) The operators receives all the displays that joined before it and when a new display joins.
Completed	✓

Description	Send a collaboration request from the operator terminal to the display
Acceptance Criteria	(a) The collaboration request has an identifier (the name which operator wants to use to show this), the screen and a data type including a unique data id with which the operator terminal can identify the request. (b) Any missing field sends back an error message with a meaningful error message. (c) A collaboration request is then forwarded to the display of choice with the room key, data type and the identifier. i. If this display id is incorrect, a meaningful error message is sent back to the operator. ii. If the display is no longer connected, a meaningful error message is sent back to the operator. iii. Unsupported Data Type or missing Data Type field sends a meaningful error message to the operator terminal. (d) A response is sent to the operator terminal with the data id and the room key which is the same as the one received by the collaboration table.
Completed	✓

Description	JSON serializable data can be sent once a collaboration is formed
Acceptance Criteria	(a) JSON data with and without lists can be sent. (b) Messages without a room key are rejected by the server and an error message is sent back to the operator terminal. (c) Messages with a wrong room key, a room key that does not exist (anymore) is rejected and a meaningful error message is sent back to the operator terminal. (d) Message that do not match the data type are rejected a meaningful error message is sent back to the terminal operator.
Completed	✓

Description	Closing a Collaboration on any display cleanly removes the collaboration
Acceptance Criteria	(a) closing the collaboration is sent by the display or operator and if the room is key is missing, an error message is sent to the display. (b) receiving a collaboration close message removes the collaboration from the server. (c) the operator receives the message collaboration is closed when the display sends the close for the collaboration.
Completed	✓

Description	(Random) Closing of Connection of Display Cleanly updates the server and operators
Acceptance Criteria	(a) All collaborations on the display are closed. (b) The operators are updated with the new available screens. (c) The server no longer has a screen with that web socket connection and the number of displays connected to the server decrements by one.
Completed	✓

Description	(Random) Closing of Connection of Operator Cleanly updates the server and display
Acceptance Criteria	(a) All collaborations the operator was part of, the operator is removed from those. (b) If there is a collaboration with no more operators, the collaboration is removed and the display is updated. (c) The server no longer has the web socket connection of this operator and the number operator terminals connected to the server decrements by one.
Completed	✓

3.5.3 Integration Testing

In this section we summarise the integration tests. The integration tests ran on our front end and included end to end testing but not user interface changes. Therefore the different data type components are not tested, or how they change.

Description	Operator Terminal after a display is connected
Acceptance Criteria	(a) Request to register as an operator after a screen was registered, updates the available displays list in the frontend.
Completed	✓

Description	Display connects after an operator connects to the server
Acceptance Criteria	(a) Once the screen is registered in the server, it is also broadcast to all operators connected and the list of available displays is updated.
Completed	✓

Description Acceptance Criteria	Starting a Collaboration (a) When starting a data in collaboration, the collaboration is added to the list of collaboration on the selected display. (b) If there is a missing identifier for the data being shared, a message notifying the operator is displayed by the user interface. (c) When a collaboration is sent, the operator gets a message from the server with a room to send messages to for this collaboration.
Completed	✓

Description Acceptance Criteria	Sending a message in Collaboration (a) Upon receiving the collaboration request response which has the original data id of which data to share, the data for that data stream are sent immediately. (b) If the data is updated, a new message is created and sent immediately if the data is still being collaborated on. (c) The collaboration table receives the message and updates the component depending on data type appropriately.
Completed	✓

Description Acceptance Criteria	Ending a Collaboration (a) When a collaboration ends, the data which was collaborated on is no longer subscribed to be sent if changed. (b) The list of collaborations on the collaboration table no longer has the collaboration associated to the room key associated with the collaboration which is closed.
Completed	✓

3.5.4 Usability Testing

For usability testing of the collaboration table, we contacted our client, and the client set up an interview with their product manager of their technical systems.

The following were the results of Usability Testing:

Dark Mode The user interface was required to follow a dark colour scheme as to reduce the strain on the operators eyes since they work long times on their screen in a low light level environment.

This criteria was fulfilled as all our components, including the different components, followed a dark mode colour scheme. The dark mode colour scheme was adopted from a very successful chat application which has the target audience who use their application in similar environment (at least in terms of ambience, lighting conditions and duration) Discord Dark Theme [1].

Mouse Mileage In User Interface Design where spontaneity is of huge importance, once example is a naval operations where different data may need to be shown very little delay, it is important any collaboration being shared or one that needs to be shared can be shown quickly on to the collaboration table. Initially we failed this usability requirement but then after receiving feedback and making relevant changes we completed this usability requirement.

In our initial design that was presented the mouse mileage was very high for the collaboration table since we included a pop up which asks the operator if a data stream could be shown on the collaboration table before actually showing it. This was changed later and it was changed to a

[1] `https://discord.com`

toggle system where data streams identifier could be clicked to show and hide the data stream associated to the identifier.

Touch Mode Intuitive gestures were required to be implemented for the collaboration table, since in practice the collaboration table would be a tablet.

For this we implemented swiping gestures for dismissing a collaboration from the collaboration table. This was said to be intuitive and quick in our usability testing as opposed to clicking an "x" button.

Overall our usability testing was successful and it addresses the main issues of the client for the collaboration table in the given domain.

3.6 Project Conclusion and Future

Conclusion

We completed all the main requirements and the deliverables required by the client, i.e., the demo and the technical design report. In our final demo, the client was satisfied with the proof of concept, including its ability to convey the purpose and how it could be useful to their system if they were to implement it.

Future

This project will not require further input from our group, since the deliverable is simply a proof of concept and is not going to be integrated into the existing system. The final deliverables, which are the demo, the source code and the technical design report were delivered.

Chapter 4

Administrative Project Details

4.1 Coordination With Client

As the system was proposed by a company, ensuring that the client is satisfied is a high priority. To ensure that the product works according to the specifications, it is essential that good information channels between the developers and the client exist.

Weekly Meetings

In order to improve consistency in meeting times, it was decided to schedule a weekly meeting with the client. This meeting takes place on Friday mornings. This moment was chosen after considering both the client's and the developers' schedules. Having such a default meeting time reduces the likelihood of either party missing meetings.

Because of the ongoing COVID-19 pandemic and the client's security concerns, it was decided to have these meetings online for the time being. This decision is subject to change.

Physical Meetings The client also expressed a willingness to meet physically with the developers. Before this can happen however, the client needs to verify the nationalities of all participants in this meeting.

Other Contact Options

Besides weekly meetings, there can be situations in which the developers need more information. For this reason, there is also a communication channel through e-mail. After discussion with the client, it was decided that e-mail should be the tool through which any additional and high priority information can be requested at any time during the week.

4.2 Planning of the Project

The planning of the project is done using the Agile methodology [7]. The details are as follows.

Sprints

- Sprint 1 (Week 1 + 2 + 3) → Requirements Gathering + Design (mockup, UML diagrams)
- Sprint 2 (Week 4 + 5) → Minimum Viable Product
- Sprint 3 (Week 6 + 7) → Refining Minimum Viable Product w/ Documentation
- Sprint 4 (Week 8 + 9) → Additional Features w/ System Testing
- Sprint 5 (Week 10) → Acceptance Testing and Wrap Up

Meetings

Meeting Dates with Thales	Meeting Dates with Doina
11th February, 2022	14th February, 2022
18th February, 2022	28th February, 2022
4th March, 2022	7th March, 2022
11th March, 2022	14th March, 2022
18th March, 2022	21st March, 2022
25th March, 2022	28th March, 2022
1st April, 2022	4th April, 2022
8th April, 2022	11th April, 2022

Platforms

- Meetings with supervisor: Discord
- Meetings with the Client Thales: Microsoft Teams
- Meetings with the Team: Discord
- Communication with the Team: Discord & WhatsApp
- Documents: OneDrive & Overle1af
- Project Version Control: Gitlab

Internal University deadlines

- Project Proposal: 28th February, 2022
- Poster, Design Report, Final Product and Manual: 22nd April, 2022

4.3 Future Planning

As mentioned in Chapter 1, our project was a proof of concept for a collaboration system to enhance TACTICOS, but due to the proof of concept nature it will not be able to be implemented as is. We will hand over the code written for the project in addition to the documentation in this report to Thales, so they can further improve on the system and possible add it to TACTICOS. Improving on this project should be easy due to the modular design of our system and protocol.

4.4 Group Self Evaluation

Communication

Our team communication was exceptionally satisfactory overall as we had a Discord server with respective channels to discuss over daily project matters. Additionally, we had occasional face-to-face meetings to have a more effective discussion.

Planning

As far as the planning of the project went, the group held weekly meetings to distribute tasks and set deadlines for the tasks. There were issues created in GitLab, corresponding to the tasks, and the issues were assigned to the responsible group members. If more meetings were required, these were set up depending on the schedule of the team members.

There were initially also weekly meetings with Thales, our client, as well as with Doina, our supervisor. Once we started the implementation stage of the project, these were changed to bi-weekly meetings as our sprints were two weeks long.

Responsibilities

In order to divide the responsibilities among the group, there were meetings among to group members to discuss each one's strengths and weaknesses. From there, we split the tasks based on

what each one was comfortable working with. The responsibilities were split as follows:

- Jordi Bals: Implementation of the Data Sharing Architecture and Unit Testing.
- Jeroen van der Horst: Simulator & Implementation of the User Interface of the Collaboration Table.
- Hyeon Kyeong Kim: Lofi and Hifi Mock-up, Connection between the Data Sharing Architecture with the User Interface & parts of the front-end, Poster creation.
- Priya Naguine: Meeting Minutes Taker, Parts of the User Interface, especially working on the Map Component.
- Puru Vaish: Implementation of the Data Sharing Architecture, Creation of the protocol logic, Unit and Integration Testing, VCS manager & Implementation of the User Interface, Terminal and Collaboration Table.

Other than that, all teams members worked on preparing the slides for the presentations and the deliverables for the design project such as the project proposal, the design report and the ethics report.

Bibliography

[1] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual*. Object Technology Series, Boston, MA: Addison-Wesley, 2 ed., 2004.

[2] R. Mitchell and J. McKim, *Design by Contract, by Example*. Boston, MA, USA: Addison Wesley, 2002.

[3] K. Beck, *Test Driven Development: By Example*. Boston, MA, USA: Addison-Wesley Professional, Nov 2002.

[4] H. Erdogmus, M. Morisio, I. C. Society, M. Torchiano, and I. C. Society, "On the effectiveness of the test-first approach to programming," *IEEE Transactions on Software Engineering*, vol. 31, pp. 226–237, 2005.

[5] "RFC 6455 - The WebSocket Protocol," Mar 2022. [Online; accessed 4. Mar. 2022].

[6] "Working with Svelte stores - Learn web development | MDN," Apr. 2022. [Online; accessed 18. Apr. 2022].

[7] Project Management Institute, *The Agile practice guide*. Newtown Square, Pennsylvania : The Project Management Institute, [2017], 2017.

Appendix A

Mock Ups

Figure A.1: Lo-fi Prototype

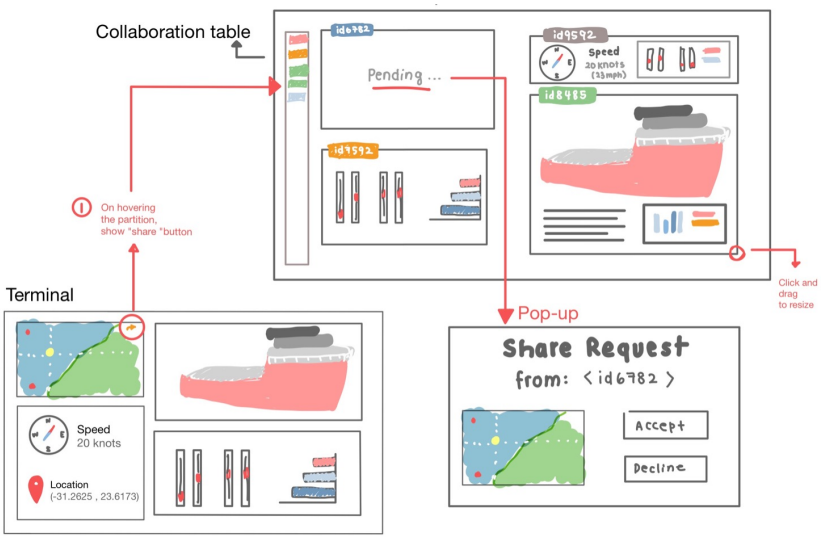
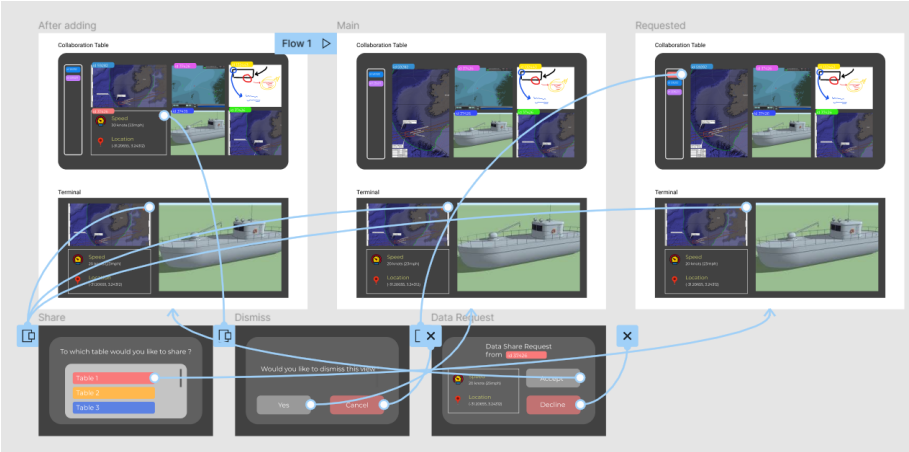


Figure A.2: Hi-fi Prototype



Appendix B

INVEST Specification

The requirements are stated below, along with their INVEST analysis:

1. No data to be modified on the source.
    - Independent: This requirement is independent.
    - Negotiable: This requirement is non-negotiable.
    - Valuable: This requirement is valuable as sensitive data is stored at the source and can have immeasurable consequences if modified incorrectly.
    - Estimable: This requirement is estimable. This is because we should only allow read access on the source.
    - Small: This requirement is small, as it cannot be broken down into smaller requirements.
    - Testable: This is testable. To do this, there should be a comparison of the data set before and after collaboration. There should be no difference on the data in the source.

2. Handle track data type with location and a hostile, non hostile and neutral attribute for an entity.
    - Independent: This requirement is independent. However, requirement 4 is dependent on it.
    - Negotiable: This requirement is non-negotiable.
    - Valuable: This requirement is valuable to the user as the user would want to know if they have to defend themselves against a hostile entity.
    - Estimable: This requirement is estimable. This is because once the data type is defined and there is data of that type, all that is left is to display it on the map.
    - Small: This requirement is small as this track data type cannot be simplified or broken down further.
    - Testable: This is testable. This can be done through end-to-end testing, as the user should be able to see a marker on the map to indicate location as well as different colours to indicate the classification of the ship, i.e., hostile, non-hostile or neutral.

3. Handle track data type with location and velocity attribute by showing an arrow.
    - Independent: This requirement is independent. However, requirement 4 is dependent on it.
    - Negotiable: This requirement is non-negotiable.
    - Valuable: This requirement is valuable to the user as the user would want to know in what direction a ship is moving.
    - Estimable: This requirement is estimable. This is because once the data type is defined and there is data of that type, all that is left is to display it on the map.
    - Small: This requirement is small as this track data type cannot be simplified or broken down further.
    - Testable: This is testable. This can be done through end-to-end testing, as the user should be able to see a marker on the map to indicate location as well as an arrow to indicate the direction of travel of the ship.

4. Handle track data with a combination of multiple attributes.
    - Independent: This requirement is not independent. It is dependent on requirements 2, 3, 5 and 17. It is necessary for the system to be able to handle the individual attributes first, before combining the attributes.
    - Negotiable: This requirement is negotiable. This is because the attributes to be combined need to be decided still.
    - Valuable: This requirement is valuable as the user might want to see all the track data at once instead of the individual track data types, i.e., look at whether the ship is hostile or not as well as the direction in which it is moving.
    - Estimable: This requirement is estimable. However, since it is dependent on requirements 2, 3, 5 and 17, it is necessary to take into account the estimation of the mentioned requirements.
    - Small: This requirement is small as this track data type makes use of existing track data attributes.
    - Testable: This is testable. This can be done through end-to-end testing, as the user should be able to see a marker on the map to indicate location, with an arrow to indicate direction, as well as different colours to indicate the classification of the ship, i.e., hostile, non-hostile or neutral, and the name of the ship.

5. Handle information data type defined by item name and text description only.
    - Independent: This requirement is independent. However, requirement 4 is dependent on it.
    - Negotiable: This requirement is non-negotiable.
    - Valuable: This requirement is valuable as the user might want to see an email which is simply text with a MIME format.
    - Estimable: This requirement is estimable. This is because once the data type is defined and there is data of that type, all that is left is to display it on the screen.
    - Small: This requirement is small as this data type cannot be simplified or broken down further.
    - Testable: This is testable. This can be done through end-to-end testing, as the user should be able to see an email on a partition on the screen.

6. Enter full-screen mode using a gesture for a partition.
    - Independent: This requirement is independent. However, requirement 7 is dependent on it.
    - Negotiable: This requirement is non-negotiable.
    - Valuable: This requirement is valuable to the user as the user might want to focus solely on a specific partition.
    - Estimable: This requirement is estimable. This is because only a gesture is required to show the particular partition.
    - Small: This requirement is small as it only requires a gesture to be defined.
    - Testable: This is testable. This can be done through end-to-end testing as the user should be able to view only the partition required.

7. Exit full-screen mode using a gesture for a partition.
    - Independent: This requirement is not independent. It is dependent on requirement 6.
    - Negotiable: This requirement is non-negotiable.
    - Valuable: This requirement is valuable to the user as the user might want to look at other data views again. In order for this to be done, the user would have to exit full screen so that the rest of the partitions can be seen.
    - Estimable: This requirement is estimable. This is because only a gesture is required to show the partitions that were previously hidden.
    - Small: This requirement is small as it only requires a gesture to be defined.
    - Testable: This is testable. This can be done through end-to-end testing as the user should be able to view all the partitions.

8. Resize the partitions on the screen.
    - Independent: This requirement is independent.
    - Negotiable: This requirement is non-negotiable.
    - Valuable: This requirement is valuable to the users as the view might not be the size required by the user.
    - Estimable: This requirement is not estimable. This is because the developers lack technical knowledge on this. Research has to be done first before working on this requirement.
    - Small: This requirement is small as it cannot be further simplified by breaking it down into smaller requirements.
    - Testable: This is testable. This can be done through end-to-end testing as the user should be able to change the size of the partitions to the predefined sizes.

9. Dismiss a view from the screen.
    - Independent: This requirement is independent.
    - Negotiable: This requirement is non-negotiable.
    - Valuable: This requirement is valuable as the user could be done analysing a view and would not like to see it anymore. This would allow more space for other views on the screen.
    - Estimable: This requirement is estimable. This is because only a gesture is used to remove the view.
    - Small: This requirement is small as it only requires a gesture to be defined.
    - Testable: This is testable. This can be done through end-to-end testing as the user should be able to swipe in order to remove the request from the queue and the relevant partition will be removed.

10. Share a Data Stream on to the collaboration screen.
    - Independent: This requirement is not independent. It is dependent on requirement 14, as the view should be named before it is displayed on the collaboration screen.
    - Negotiable: This requirement is negotiable. The types of data streams that can be shared can still be decided.
    - Valuable: This requirement is valuable as a terminal officer might want to discuss a situation with other terminal officers or the commander. For this to be done, the terminal officer would have to share the required data stream on the collaboration table.
    - Estimable: This is not completely estimable. This is because it depends on the connection between the front-end and the back-end, as the sharing can only be done once the connection is complete.
    - Small: This requirement is not small. This is because the different types of streams that can be shared have to be defined first, so it would be preferable to break it down into more small requirements.
    - Testable: This is testable. This can be done through end-to-end testing as the user should be able to view the data stream request in the requests queue on the screen.

11. Extensible for more data types and describers.
    - Independent: This requirement is not completely independent. This would require initial data types and describers to be defined, so it is dependent on requirements 2, 3, and 5.
    - Negotiable: This requirement is negotiable. This is because there has to be a decision made on the data types and describers to add.
    - Valuable: This requirement is valuable as the user might want more information to be displayed in the screen that was not available before.
    - Estimable: This requirement is not estimable. This is because it depends on the data types and describers to be added and this may be affected by the developer's domain knowledge.
    - Small: This requirement is not small. The size of this requirement is determined by the

number of data types and describers to be added. In this case, each new data type and describer would be a requirement on its own.
- Testable: This is not testable. This is because this is about the programming approach, rather than the program itself.

12. Draw on a particular partition on the collaboration screen.
    - Independent: This requirement is independent.
    - Negotiable: This requirement is negotiable. This is because there has to be a decision made on which type of partitions should allow for drawing and if there are any restrictions to what can be drawn.
    - Valuable: This requirement is valuable as the user want to annotate a map or a schematic during a debriefing session.
    - Estimable: This requirement is not estimable. This is because the developers lack the technical knowledge required. Research has to be done before adding this feature.
    - Small: This requirement is not small. This is because another program would have to be written with this drawing functionality to then be integrated with the existing system.
    - Testable: This is testable. This can be done through end-to-end testing as the user should be able to view the drawing on the specific partition.

13. Accept collaboration of another user to existing view.
    - Independent: This requirement is independent.
    - Negotiable: This requirement is negotiable. This is because there might not be a necessity to accept the collaboration. The collaboration could be done without having to explicitly accept it.
    - Valuable: This requirement is valuable as the user might want to discuss a particular situation with another user, so the collaboration should be accepted before he data is displayed on the screen.
    - Estimable: This requirement is estimable. This is because the request simply goes to the request queue and the user can choose whether or not to display it.
    - Small: This requirement is small. Since the request only has to be placed in the request queue, it cannot be broken down.
    - Testable: This is testable. This is a form of unit testing which tests the data-sharing architecture implemented in the Python server.

14. Add a name to a data view.
    - Independent: This requirement is independent. However, requirement 10 is dependent on it.
    - Negotiable: This requirement is non-negotiable.
    - Valuable: This requirement is valuable as the user should be able to identify the view before displaying it on the screen. Once the user knows the name of the view, they can click on it so that it appears on the screen.
    - Estimable: This requirement is estimable. This is because a name simply has to be given before sharing the view so that it can be identified in the collaboration screen.
    - Small: This requirement is small as it cannot be split into more requirements.
    - Testable: This is testable. This can be done using both unit and integration testing.

15. Accept the display of a view onto the screen.
    - Independent: This requirement is not independent. There has to be a share request first, so it is dependent on requirement 10.
    - Negotiable: This requirement is negotiable. This is because there might not be a necessity to accept the collaboration. The collaboration could be done without having to explicitly accept it.
    - Valuable: This is valuable as the user might want to decide on what to display and what not to display during a briefing session.
    - Estimable: This requirement is estimable. This is because the request simply goes to the request queue and the user can choose whether or not to display it.

- Small: This requirement is small. Since the request only has to be placed in the request queue, it cannot be broken down.
- Testable: This is testable. This can be done through end-to-end testing as the user should be able to view the data on a partition in the screen.

16. Group different view objects together.
    - Independent: This requirement is independent.
    - Negotiable: This requirement is negotiable. This is because there can be restrictions on which view objects can be grouped together.
    - Valuable: This requirement is valuable as the user might want to group together relevant information.
    - Estimable: This requirement is not estimable. This is due to the developer's lack of the technical knowledge required.
    - Small: This requirement is not small. This is because the positioning of the objects, their sizes, etc. have to be worked on. So, this can be broken down into simpler requirements.
    - Testable: This is testable. This can be done through end-to-end testing as the user should be able to put partitions together on the screen.

17. Handle information data type, item name and text description and image.
    - Independent: This requirement is independent. However, requirement 4 is dependent on it.
    - Negotiable: This requirement is non-negotiable.
    - Valuable: This requirement is valuable as this could be used to describe a ship. This could include the ship's name, description and image.
    - Estimable: This requirement is estimable. This is because once the data type is defined and there is data of that type, all that is left is to display it on the screen.
    - Small: This requirement is small as this data type cannot be simplified or broken down further.
    - Testable: This is testable. This can be done through end-to-end testing as the user should be able to view the ship's data on a partition in the screen.

18. Compatible with asynchronous subscribe data sources.
    - Independent: This requirement is independent.
    - Negotiable: This requirement is non-negotiable.
    - Valuable: This requirement is valuable because the current system in Thales works with real-time data coming from the cloud, and this is a feature they require from our proof of concept.
    - Estimable: This requirement is not estimable as the developers lack technical knowledge on this.
    - Small: This requirement is not small. This is because it can be broken down into smaller requirements such as dealing with reactive updates, updating the operator terminal based on the data received, etc.
    - Testable: This is testable. This can be done through integration testing as when there is a change in the asynchronous subscribe data source, there is a corresponding change in the receiver's data.

19. Incorporate Open Geo-Spatial Consortium conventions for the track data type.
    - Independent: This requirement is not independent. It is dependent on requirements 2 and 3.
    - Negotiable: This requirement is negotiable. This is because it is vague. There is not enough information on what parts of this convention have to be implemented.
    - Valuable: This requirement is not valuable as Thales has their own map API and mocking it would not add value for a proof of concept.
    - Estimable: This requirement is not estimable. This is due to the developer's lack of technical knowledge.

- Small: This requirement is not small. This is because these conventions can be broken down further into simpler requirements.
- Testable: This is testable. This is because the map API follows the Open Geo-Spatial Consortium conventions as specified.

Appendix C

Meetings with Thales

We had 7 meetings with Thales in total, excluding the physical visit to Thales.

Meeting 0

This was the introductory meeting with Thales. The purpose of this meeting was to get an idea of what Thales does and what they required from us. We got information regarding the domain setting, what their current technology is like and what are the improvements required on the current technology for us to have an idea of what we would be required to implement. We also got information on the programming languages used by Thales and also whom to contact in case interviews were to be conducted with Thales employees.

After the first meeting, an initial requirements specification was written and a lo-fi prototype created.

Meeting 1

The purpose of this meeting was to refine the functional requirements, show the lo-fi prototype that was created and get feedback on both. Also, quality requirements were discussed. Thales also gave us the use cases that we were supposed to work with.

After this meeting, we wrote the project proposal which included all the requirements, both functional and non-function, the lo-fi prototype, and the planning.

Meeting 2

The purpose of this meeting was to present the completed project proposal and get feedback on it. We also presented the initial design for the system. Another point that was discussed was the progress made by us on the meeting to be conducted with the potential users of the system.

After this meeting, we started working on the MVP.

Meeting 3

The purpose of this meeting was to present our MVP to Thales and get feedback on it, as well as tell them about our future plans towards the refinement of the MVP.

After this meeting, the team worked on the refinement of the MVP as well as the hi-fi prototype to present to the project manager.

Meeting 4

The purpose of this meeting was to conduct an interview with Franck Maarse, who is the project manager. We showed Franck our lo-fi prototype and current implementation. We asked him for feedback on our current implementation and if there was something else he would like to see.

After this meeting, we incorporated the feedback given by Franck into our project and continued working on the refinement of the MVP.

Meeting 5

The purpose of this meeting was to give an update on what we had been working on.

After this meeting, we continued working on the refinement of the MVP.

Meeting 6

The purpose of this meeting was to show the refinement of the MVP and get feedback on that.

After this meeting, the team worked on the finishing touches as well as on the Design Report.

Physical Visit to Thales

There is a physical visit to Thales planned for the 22nd of April, 2022, where acceptance testing will be conducted and a presentation given on the project. During this visit, the team was given a tour of the Thales building, as well as a presentation on the existing TACTICOS system. After that, the team gave a presentation on what was worked on and showed a demo of the current system.

Appendix D

Sprint Reviews

Sprint 1

Sprint 1 ran through weeks 1, 2 and 3. The requirements gathering was done in this sprint, as well as the creation of a mock-up, i.e., lo-fi and hi-fi prototypes and UML diagrams. The UML diagrams include a class diagram, use case diagram and sequence diagram. By the end of the sprint, the project proposal was created, which included the requirements specification, UML diagrams, lo-fi and hi-fi prototypes and the planning for the project.

Sprint 2

Sprint 2 ran through weeks 4 and 5. The team worked on developing the Minimum Viable Product during this time. The test-driven development approach was followed so tests were also written during the development of the Minimum Viable Product. Documentation on the code was also written during development.

Sprint 3

Sprint 3 ran through weeks 6 and 7. During this sprint, the team worked on the refinement of the MVP as well as on the technical documentation.

Sprint 4

Sprint 4 ran through weeks 8 and 9. During this sprint, the team continued to work on the refinement of the MVP, integration testing and the technical documentation. The team also started working on writing the design report.

Sprint 5

Sprint 5 ran through week 10. During this sprint, the team worked on the finishing touches, design report and the acceptance testing.