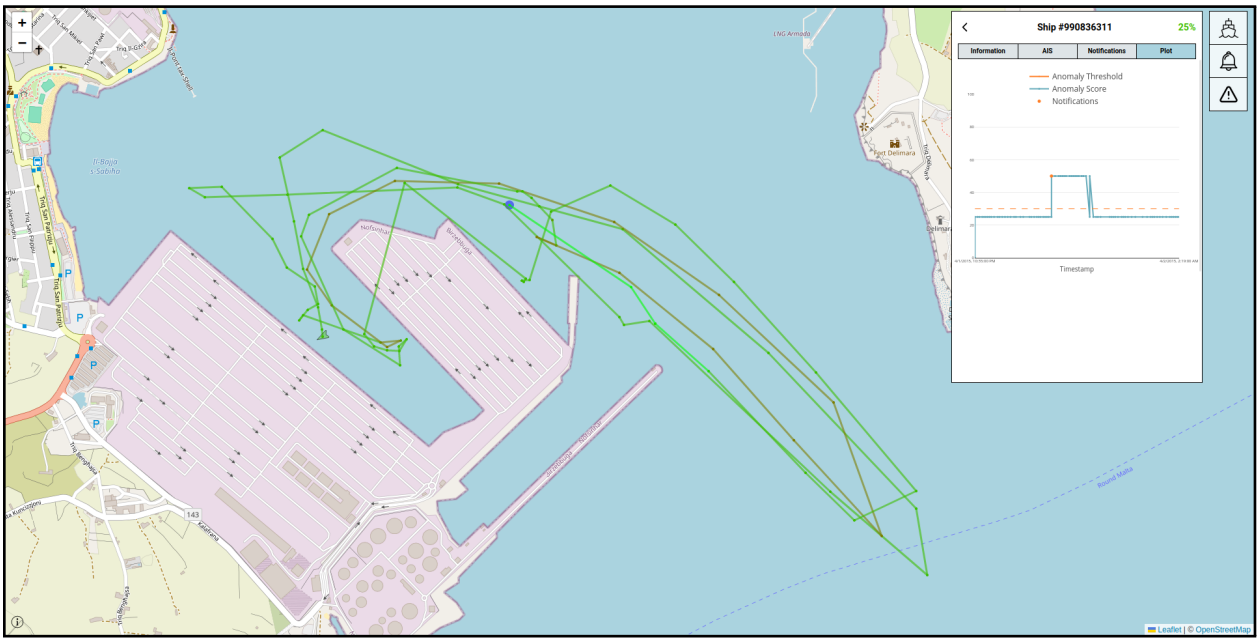
Track Anomaly Detection

Developing Software for Detecting Anomalous Vehicles in the Sea



Report submitted as the final delivarable for course:

Software Project (CSE2000)

Group 18A

Augustinas Jučas, *5742439*

Victor Purice, *5777216*

Justinas Jučas, *5739500*

Marius Frija, *5804132*

Aldas Lenkšas, *5714192*

Client: **Dutch Ministry of Defence**

Client Representative: **Sebastiaan Alvarez Rodriguez**

Teaching Assistant: **Nathalie van de Werken**, Coach: **Kubilay Atasu**

Date of submission: **21 June 2024**

TU Delft

Preface

This report was prepared by five second-year Computer Science students from TUDelft as part of the CSE2000 Software Project course[1].

We want to express our gratitude and appreciation to everyone who supported us throughout the project's timeline. In particular, we would like to thank:

- The client's representative, Sebastiaan Alvarez Rodriguez, for guiding us throughout the project, providing valuable feedback and organizing insightful meetings with responsible people from the Ministry of Defence.
- TU Delft support through our teaching assistant Nathalie van de Werken for providing constructive feedback and our coach Kubilay Atasu for valuable implementation suggestions and guidance.
- Lieutenant Martijn van Diessen for the generously offered time and for answering important questions regarding the problem of our application.
- Technical Writing and Teamwork lecturers from TU Delft for sharing their expertise and engaging lectures.
- The open source community for the remarkable available tools that aided the development process of our system.

The team had a lot of fun.

Sincerely,

The authors,
Delft, The Netherlands
June 2024.

The team during a trip to a navy military base in Den Helder.
From left to right: Marius Frija, Justinas Jučas, Victor Purice,
Aldas Lenkšas, Augustinas Jučas.

[1] **Total word count**: 12 907
Statement on the use of generative AI: In writing this text, we used generative AI to improve the grammar, style, and/or spelling of the text.

Summary

Detecting malicious ships in the sea represents a large problem for the Royal Netherlands Navy. This issue arises from the tens of thousands of vessels in the oceans, out of which only a small fraction may present security threats that concern the military. Furthermore, accurately identifying the few suspicious ships requires inspecting each watercraft's information separately. Currently, the identification of such vessels must be conducted manually by military personnel with little specialized software support. As a result, the process requires significant financial and human resources and, therefore, should be optimized.

The engineering challenge that needs to be addressed is as follows. Most ships are required by law to broadcast their AIS (Automatic Identification System used for tracking ships) information. Such broadcasted signals must be received by a specific software system and analyzed, and anomaly information for them must be derived, indicating the level of suspicion associated with a corresponding ship. Due to the large number of watercraft in the oceans, the system must be developed to be capable of managing thousands of unique data streams simultaneously. Additionally, the Navy operators, who are the main users of the application, should have access to an interface displaying a map with all known vessels, along with their current anomaly information.

This report presents our solution to the described anomaly detection problem and details the development process. It begins with an issue analysis, including an investigation of existing solutions and a list of the derived requirements. Afterwards, it describes the implemented system, covering the analysis of available technologies, the implemented anomaly detection algorithm and its alternatives, and screenshots of the user interface design process. Additionally, the report includes chapters on quality assurance techniques, ethical considerations, teamwork processes and considerations for future work.

The final solution to the ship anomaly detection problem required solving multiple technical challenges. Distributed systems such as Kafka, Flink, and Druid were used to ensure that the application could be easily scaled to handle large data loads. Furthermore, the client had a requirement to ensure that the anomaly scores assigned by the system had human-readable explanations of why a particular score was assigned. This requirement was crucial in selecting the anomaly detection techniques. In particular, a heuristic approach was chosen - multiple rules describing suspicious behaviour were derived from a provided dataset and a technical interview with a Navy officer. This rule-based approach ensured that the implemented anomaly detection algorithm was explainable.

At the end of the project, the main goals were accomplished - all *must* and *should* requirements were implemented. To be more precise, the final developed system is capable of handling the required high loads of incoming ship signals while also having an intuitive and fast user interface. Additionally, the software was written to ensure that the anomaly algorithms used were modular and thus easily extendable. Furthermore, the software was extensively tested, with unit, mutation, integration, stress, distributability and UI (user interface) tests, reaching high coverage in terms of code lines and functionalities. Also, the accuracy of the final anomaly detection algorithm was evaluated by comparing it to general machine learning techniques, such as isolation forests. Future work was identified, mainly including improvements to the anomaly detection algorithm, introducing a user system, and adding support for different types of vessels. Finally, ethical implications were considered both during and after the development of the application.

Contents

Chapter 1

Introduction

Piracy and maritime drug trafficking pose serious safety, economic, and social risks for global maritime traffic. In 2023, 120 pirate attacks were reported worldwide, having grave repercussions on the trade and passenger transportation sectors in certain regions of the globe [1]. Moreover, as stated in the UNODC World Drug Report [2], illicit drug smuggling by means of maritime transportation represents a key component of the global narcotics distribution networks. Detection of such malicious agents in global maritime traffic is difficult due to insufficient human and administrative resources for direct surveillance and the immense surface that must be constantly monitored [3]. Our client, the Netherlands Ministry of Defence, is aiming to find a solution to detecting suspicious maritime activities by attempting to distinguish ships that display anomalous behaviour as early as possible. Developing an innovative automated solution that would increase the current capabilities of detecting real-time traffic anomalies would not only ensure higher transportation safety but also help fight against international criminal organizations.

The objective of this report is to describe a software system we have developed to solve the problem of automating the detection of anomalous vessels. One of the primary system requirements was also to enable processing a substantial volume of streaming data and accommodate computationally complex algorithms. After researching the possible frameworks, the implemented architecture was made to be easily scalable and distributable using Kafka, Flink and Druid. As for the detection algorithms, using heuristics introduced more explainability [4] and robustness but less accuracy compared to machine learning methods that were less understandable to the operators of the naval ships.

The report is structured as follows. Chapter 2 provides the theoretical background for our problem. The general architecture of the software system is described in Chapter 3, which details the complete technology stack required for the implementation of the application, together with the process of researching the anomaly detection algorithms. A quality assurance analysis is provided in Chapter 4, stressing the performance and quality metrics achieved in the final application. The ethical considerations can be found in Chapter 5. Chapter 6 of the report provides insights into how the internal team development process was conducted. Future work is outlined in Chapter 7, whereas Chapter 8 describes the conclusions of this report and recommendations for the Navy.

Chapter 2

Problem Analysis

The following chapter presents a thorough analysis of the problem aimed to be solved, which was helping the Navy operators perform ship track anomaly detection. In particular, Section 2.1 includes a detailed description of the mentioned issue. Stakeholders are described in Section 2.2. Afterwards, Section 2.3 describes the currently existing solutions that could potentially tackle it. Furthermore, Section 2.4 details the requirements specified by the client for the desired solution of the problem. Finally, Section 2.5 and Section 2.6 outline the performed feasibility study and risk analysis respectively.

2.1 Problem Description

The large number of ships in the oceans presents great security concerns since vessels with malicious intent are hard to manually discern from loads of normal watercraft. The commercial fleet alone consists of at least 90 000 ships [5]. This problem of finding anomalous ships among thousands of ordinary ones is especially relevant to the Royal Netherlands Navy.

In particular, the Navy would like to find suspicious ships to be further inspected and possibly neutralized. Furthermore, at the present moment, this detection of suspicious vessels is done manually by Navy operators. The mentioned military specialists must track suspicious transport vehicles to prevent drug trafficking, piracy, infrastructure damage, and other security problems. According to our research (described in Section 3.2.1), at the present moment the operators rely on their expertise and the provided guidelines to detect vehicles which display anomalous behaviour, such as unusually fast speed or a suspicious trajectory.

The amount of vehicles that need to be checked manually makes this task difficult and hardly scalable. The challenge that needs to be addressed is automating the detection of anomalous vehicles, so that military officers can use this solution to make decisions faster and on a much larger scale. This would result in increased productivity of the military personnel and as a consequence, safer oceans.

2.2 Stakeholders

The first group of stakeholders are the Navy operators. They are the main users of the system, which means they are the most important stakeholders. Their main desires for the system are to have a clear and intuitive design, as well as to get explanations of the computed anomaly score. To collect their relevant domain knowledge and understand which anomaly detection techniques are preferred, our team talked with a Navy officer (refer to Section 3.2.1).

Another group of stakeholders are the Navy software engineers. They would be responsible for deploying and maintaining our implemented software on their servers. Thus, their main desires are: the system should be easily scalable and distributable, the system should be easy to use and deploy, and the code should be well-documented.

2.3 Existing Solutions

A solution to the problem described in the previous section would have to include two main parts. First, a user interface that allows military operators to inspect existing ships in real-time while also providing anomaly information for each vessel. Second, the solution would have to include a system that can handle receiving large loads of incoming data from thousands of ships at sea. It would process incoming signals, calculate their anomaly scores, and provide related data to the user interface.

There exist only partial solutions for this track anomaly detection problem. To the best of our knowledge, no applications that contain both crucial parts exist in the public domain. Even if, according to our client (the Dutch Ministry of Defense), the Navy already has some form of the described system, their solution is classified, and no useful information about it was provided to us. As a partial solution, there are concrete systems for handling and displaying high loads of ship data (see 2.3.1), but without any anomaly detection mechanism. Moreover, there also exist relevant anomaly detection algorithms that track traffic anomalies - it is a known issue that has research behind it (see 2.3.2). Combining these two partial solutions would result in a system that would be capable of solving the full problem. Therefore, these existing partial solutions are presented next.

2.3.1 Existing Systems for Processing High Numbers of Incoming Vehicle Signals

There is a large number of systems that can deal with processing a massive load of incoming signals. As for such systems that deal with data from transport vehicles, two major software systems exist: *marinetraffic* website[1] and *absexchange* website[2]. In general, both products offer similar functionalities, but they represent only partial solutions to the problem at hand since they do not include any anomaly-tracking capabilities.

Additionally, these two systems include a lot of the features that are also crucial for the track anomaly detection solution. For instance, the *marinetraffic* website displays an interactive map displaying global maritime traffic data currently obtained from AIS (Automatic Identification System) signals. Additionally, the *absexchange* website serves the same purpose but for air traffic. Both of these systems were used as an example in terms of their data visualization functionalities, such as increased navigability of the virtual map and handling of a large number of incoming signals both in terms of processing and visualization on the map. Moreover, they both provide a list in the user interface that allows for quick searching and filtering. A few of these features were used as a point of reference during the development process of the solution, tailored to the needs of the Navy.

2.3.2 Existing Track Anomaly Detection Algorithms

There are already related existing solutions for the anomaly detection problem. Our team has found three papers with state-of-the-art algorithms. All of them deal with finding anomalies in the traffic data which is also relevant to our problem (finding anomalies in ship traffic). The first one is "uTRAND: Unsupervised Anomaly Detection in Traffic Trajectories" [6]. This paper contains relevant insights for dealing with unlabeled data and for formulating explanations for human interpretation. The other two papers deal with slightly different anomaly problems than the problem that we have dealt with. However, both present related observations. One paper is "DeepFlow: Abnormal Traffic Flow Detection Using Siamese Networks" [7]. There a solution is proposed for detecting an anomalous traffic flow with only a small training dataset. Another one is "Anomaly Detection in Road Networks Using Sliding-Window Tensor Factorization" [8]. This paper suggests a solution to anomaly detection using graph and grid building and representing the traffic data as a third-order tensor, to which the tensor factorization could be used.

[1] `https://www.marinetraffic.com/en/ais/home/`

[2] `https://globe.adsbexchange.com/`

2.4 Requirement Analysis

Deriving a comprehensive list of requirements represents one of the most critical milestones in the life cycle of a software application. During the first week of the project timeline, collecting, analyzing, and formulating the application requirements represented our most important priorities and were, therefore, handled with high diligence. The main objective of the previously described process was to obtain a highly detailed picture of the future application, around which the entire development process could be effectively organized.

2.4.1 Requirement Elicitation Process

The requirement elicitation process comprised two techniques, which relied on direct feedback from the client representatives. The first technique was to derive the preliminary list of both functional and non-functional application requirements based on the initial understanding of the application concept. After that, the opinion was asked from the client to specify key aspects that need to be changed in the list of requirements. Once we obtained rich feedback, emphasizing the necessary conceptual changes, the requirements were changed accordingly. Another step in the requirement elicitation process was interviewing the Navy officer who provided relevant insights into how the system would be used and what the preferred requirements have (see 3.2.1).

2.4.2 System Performance Prioritization Over The Algorithm

During the initial weeks of the project, we had several discussions with the Project Coach and the Client regarding the main problem that our system should address. The original plan was to create a system that was both highly scalable and distributable, with a user-friendly interface, as well as incorporating a sophisticated explainable ML algorithm. However, it became clear that addressing both of these aspects within the given timeframe was not feasible. As a result, the Client expressed a preference for prioritizing the scalability and distributability of the system over the performance of the algorithm. The final requirements list was created to reflect this decision. However, as it is described in later sections (3.2 4.3), a great amount of effort was spent on research, development and analysis of the algorithm.

2.4.3 Key Requirements

There were clear key requirements for the system. From the functional requirements, it was important that the map is displayed and users are able to drag and zoom in it. On this map, the ships should be displayed and it should be possible to select them and see the relevant information. The key non-functional requirements, as already discussed in the previous sections, were to make the system scalable and distributable. The final list of the requirements was formulated according to the MoSCoW method [9] and are presented in Appendix A. The main requirements, which represent the Minimum Viable Product, are the Must requirements.

2.5 Feasibility Study

2.5.1 Feasibility

The allocated time for completing the following project was ten weeks, i.e. the duration of the CSE2000 course. Although a significant amount of time was dedicated to working on the final report and other related deliverables, our preliminary planning was optimistic, projecting a deadline for the MVP (minimum viable product) in the fourth week of the course and a deadline for the final version of the application in the eighth week of the course. The development process was highly dependent on external factors, such as the availability of the client representative and the coach to provide the necessary assistance or feedback or the availability of specific hardware resources required for testing. Despite the mentioned external factors, we managed to fulfil the expectations of achieving a high-quality application in the corresponding time constraints. Considering that the development process focused on three main components: backend, frontend, and anomaly detection algorithms, each with a high degree of implementation complexity, the project was of a tolerably large scale in the scope of the requirements of the course.

The team had a large degree of freedom regarding the frameworks and third-party services used to implement the application concept. Therefore, after a meticulous analysis of the application requirements, the necessary technology stack was derived together with the required resources, such as the domain-specific dataset.

For the initial phase of our project (developing the Minimum Viable Product), we possessed all the necessary resources. The ending phase of the project (scaling the system in a distributed fashion) required access to more hardware resources for the purpose of testing the system's performance. Due to the lack of the aforementioned resources, the team planned to acquire them using personal means. Additionally, in order to ensure the validity and accuracy of the devised anomaly detection algorithms, a labelled dataset containing sequences of AIS signals was required. At the time the feasibility study was conducted, the drafted solution to this problem was deriving a test dataset by manually labelling the behaviour of different ship instances.

2.5.2 Available Data

As the developed application represents a data-intensive system which processes maritime traffic data, it was required to leverage datasets containing real AIS information. Regarding the available data at the start of the project, we were restricted from accessing classified datasets hosted by the client. As suggested by the client, the system runs on publicly available data provided by MarineTraffic Research [10]. The corresponding data set consists of AIS signals emitted in the Mediterranean Sea 10 years ago, sufficient for developing this application. The data analysis is extensively presented in Appendix B.

2.5.3 Changes in Case the Project is Infeasible

Under the vision of the team in the first week of the project, we considered the possibility of not being able to implement the advanced features of the application, as these require access to a more complex hardware system, a larger dataset and a higher level of expertise in implementing a distributed system. We addressed this concern with our client and reached a compromise, which entails achieving an application that is capable of running the system in a "simulated" distributed fashion (i.e. utilizing the computational resources of a single machine rather than multiple machines).

2.6 Risk Analysis

Based on the extensive analysis and planning that was performed by the team during the first week of the course, we formulated the following potential risks to the successful completion of the project:

- **Problem:** Unavailability of the computational resources for scaling the system in a distributed fashion.
  Solution: As described in the above paragraphs, an agreement with the client has been reached, such that we can take an alternative direction in the development of the advanced features of the application. In addition, the client representative agreed to provide necessary technical guidance when needed.
- **Problem:** Insufficient domain-specific knowledge for developing a highly performant anomaly detection algorithm.
  Solution: As the task of detecting anomalous behaviour is partly based on specific heuristics used by the navy operators, which may not be publicly available, we have asked our client about the possibility of having a face-to-face meeting with the navy specialists to elicit the necessary information for developing the detection pipeline and our proposal was granted.

Chapter 3

Description of the Implemented Solution

The following chapter describes the key decisions taken concerning the implemented solution. The goal of this chapter is to provide all necessary information and reasoning behind the system so that it can be reproduced or extended by future developers. In Section 3.1, the key design choices for the system are presented, together with the motivation for the chosen technologies. This section also includes an overview of the final design of the system, together with a high-level diagram of the components. The second section (3.2) focuses on a single yet crucial part of the system - the anomaly detection algorithm. It outlines the choices made, together with the reasoning and considered alternatives. After that, how the visual user interface was developed is described in Section 3.3. Lastly, a brief description of the completed requirements can be found in Section 3.4.

3.1 Outline of The Architecture

The system that tackles the track anomaly detection issue has to solve all of the following problems:

- **Problem no. 1**. In a real-world scenario, multiple large streams of AIS (Automatic Identification System) signals would be sent into the application. The application must handle receiving and directing all of these streams in a persistent and scalable way.
- **Problem no. 2**. A stream of incoming AIS signals must be processed fast so that an anomaly score for the ship corresponding to the AIS signal can be updated in real time.
- **Problem no. 3**. The large load of processed information must be persistently stored in a database for quick retrieval in the future.
- **Problem no. 4**. The calculated anomaly scores and ship positions must be accessible at all times. I.e., the user interface must be able to communicate with the server to obtain anomaly scores of the transport vehicles and display the ships.

The existing technologies that could tackle these three large problems are discussed next. In Section 3.1.1, the possible solutions for the first two problems are discussed. Afterwards, Section 3.1.2 discusses the technologies chosen for tackling the fourth problem, from the perspective of the server side. Then, Section 3.1.3 discusses the technologies chosen for tackling the problem 3. The solutions to the problem 4 from the frontend side are discussed in Section 3.1.4. Finally, Section 3.1.5 draws the overall picture of how these previously described and other smaller components connect to form a coherent system architecture.

3.1.1 Selecting Technologies for Big Data Processing

The issue of handling a large number of incoming AIS signals and quickly calculating anomaly scores for them is exactly the issue of large stream processing. Therefore, stream handling and processing frameworks were analyzed first to decide what could be used to tackle the problem 1 and problem 2.

There is a large list of potential stream processing engines that could be used: Apache Kafka, Apache Spark, Apache Flink, Apache Storm, Apache Apex, and multiple others. Most of them are designed for scalability and distributability when processing large streams, which is exactly what the anomaly detection problem requires. Many of these frameworks were considered and thoroughly analyzed at the beginning of the project while planning. The extensive descriptions and comparisons that were made between popular streaming frameworks and libraries can be found in Appendix C. These comparisons served as an important research component when designing the application but are emitted from the main part of the report for readability purposes.

After analyzing the possible options for the existing stream processing frameworks (for more information, see Appendix C), three key possibilities for solving the problem 1 and the problem 2 were identified:

(a) **Kafka**. Apache Kafka framework could be used as a message queue to handle the large load of incoming messages (first problem), and the Kafka Streams library could be utilized to process them (second problem). This would have the advantages of perfect compatibility between components and being relatively easy to set up. However, such a configuration would not be ideal since it would have suboptimal performance and limited algorithm capabilities due to the limitations of Kafka Streams. For instance, parallelizing and distributing Kafka Streams applications is relatively more difficult compared to other frameworks such as Flink.

(b) **Kafka + Samza**. Apache Kafka could be used as a message queue and for persistence purposes (first problem), while Apache Samza would be used as a processing framework for anomaly score calculation (second problem). This would be almost ideal, but it would have the drawback of relatively little support from the Samza developers' community since the framework is relatively unpopular and, as a result, less documented.

(c) **Kafka + Flink**. Finally, the optimal option would be to use Kafka as a message queue and for persistence purposes (first problem) and to use Flink (consuming straight from Kafka) for most of the stream processing (second problem). This would result in the best of both worlds: reliable and scalable stream handling by Kafka, fast anomaly score calculation by Flink, and great support and documentation for these frameworks. The only drawback would be that such a configuration may be harder to set up initially.

As a result, option (c) was chosen, which uses Kafka and Flink to solve the first and second problems.

Note that for all three of the mentioned options, Kafka was the solution to the problem 1. The reason for such a choice was mainly due to how well the Kafka messaging queue integrates with stream processing engines, such as the mentioned Kafka Streams, Samza and Flink frameworks. Other alternatives to Kafka, such as Apache Pulsar or Liftbridge, were briefly considered, but due to how well Kafka fits the needs of the application, they were quickly dismissed.

3.1.2 Selecting Web Technology Stack

To address the back-end part of the problem 4, a framework for the web server had to be chosen. First, to arrive at the decision, Java was chosen as the main programming language for the backend for the following reasons:

- Both Kafka and Flink are written in Java and/or Scala.
- The entire developer team has experience in Java, having developed back-end applications in this language.
- Java is a very common choice for web server implementations.

Therefore, there would be a straightforward link between the application's streaming pipeline and the web server part.

Once the use of the Java language was decided, a framework for solving part of the problem 4 was selected. In particular, a framework for handling internet requests from the client application was chosen. That framework was selected to be Spring Boot. The main reasons for the choice were the following:

- Spring (Boot) is arguably the most popular web framework for Java.
- All team members have prior experience working with Spring.
- Spring Boot supports handling both HTTP requests and Web Sockets which are the two main features that are required for the application.
- Spring Boot is well documented and has high community support.
- Spring supports dependency injection, simplifying development significantly.

3.1.3 Selecting the Database

Solving the problem 3 required a database for storing vast amounts of data coming in and out of the system in stream. It must be distributed since it should support a high volume of writes in a short time. Additionally, since Kafka was selected as the messaging platform for handling the high streaming loads, it would be ideal if the database had support for injecting data straight from a Kafka source.

Therefore, the mentioned constraints narrowed the options to a few candidates. In particular, the following options were considered:

(a) **Apache Cassandra**. A distributed database for storing data quickly and reliably. It supports fast querying when the queries are simple but is less optimized for analytical querying. Additionally, it can handle fast injection of data from a Kafka source but is not specifically optimized for real-time processing. For instance, Cassandra primarily uses disk-based storage with only some in-memory caching, resulting in a safer durability configuration but also reduced performance.

(b) **Apache Druid**. A database for storing streaming data. It can handle hundreds of thousands of writes per second while enabling fast data analytic queries in real-time. Additionally, it supports SQL queries, is a column-based database and is easily distributable. Druid is being supervised the same corporation (Apache Software Foundation) as Kafka and Flink. As a result, it has built-in support for ingesting data from Kafka, which is very convenient for integration. It also supports high-performance querying, which could be very useful if the application was extended with advanced anomaly detection techniques. The main drawback of Apache Druid is that it does not support fast data updates. Fortunately, such capabilities are not needed for this system.

(c) **ksqlDB**. A streaming database that is built on top of Kafka. It is designed specifically for real-time processing of data, allowing users to perform queries on the streams. It is a fast framework that allows low-latency simple querying of data. On the other hand, just like Apache Cassandra, it is not designed for more complex analytical queries. Additionally, ksqlDB is optimized mostly for data processing and not storage. But in our case, the processing is done using Flink, and the purpose of a database is storage and possibly more complex querying. Therefore, ksqlDB may not be the perfect fit.

After considering the advantages and tradeoffs offered by the candidate databases, it was decided to use the Apache Druid database due to its specialization in real-time processing, its ability to quickly handle complex queries, and its very easy integration with Kafka.

3.1.4 Selecting Frontend Technology Stack

To solve the problem 3 mentioned in the chapter introduction, we have implemented an intermediate frontend server, which acts as a link between the client and the backend server. The most suitable environment to run

it on is NodeJS[1], which represents a top choice for generating dynamic web page content before the page is sent to the user's web browser. In addition to this, the Node's package manager *npm*[2] greatly simplifies the installation, update, and uninstallation of the used packages, which came in handy during the project.

Selecting Frontend Framework

Modern frontend frameworks aim to automate the overhead associated with common activities performed in web development. The most popular and widely supported front-end frameworks at present are React.js, Angular.js, and Vue.js [11]. The application's frontend was implemented using the React.js framework[3], due to several advantages that it provides compared to other popular frameworks:

- React.js adheres to the declarative programming paradigm by allowing the developers to design views for each state of the application, which makes the components easy to reuse and modular.
- React.js uses a virtual Document Object Model, an in-memory copy of the actual Document Object Model, which helps update only a part of the web page instead of the whole page.
- React.js itself does not include a lot of components such as routing, state management, server-side rendering, and others, thus making use of the tools which are provided by the open-source community that makes its usage very modular and very easy to adapt to the problems that the developer face.

Other popular frameworks, such as Angular.js and Vue.js, have some drawbacks compared to React.js. Angular.js is a more heavy-weight framework, has a steeper learning curve, and introduces some performance overhead. On the other hand, Vue.js is not as popular as React.js, and hence, it has a worse ecosystem of tools that support it.

React.js supports multiple frameworks, which, as mentioned, provide functionality for features that were useful throughout the course of the project. Some of them were considered:

- React.js Native represents a useful framework for building native mobile apps. Making the product work well on mobile devices is not a core requirement in the application, so this framework is not used.
- Next.js is a framework that solves two important web application problems: server-side rendering and search engine discoverability. Server-side rendering might help us achieve better application performance for the app's users, but it was decided not to use it, mainly because of the short timeframe of the project. Moreover, search engine discover-ability, which represents the likeliness of the users of a search engine like Google Search to find the application falls beyond the scope of the project.

Selecting Programming Language

TypeScript was chosen as the main programming language for the frontend. It is a widely supported programming language that provides type-checking and static typing functionality and transpiles directly to simple JavaScript code. Using TypeScript in the project made the team write higher standard code and be more *pragmatic* in implementing the frontend.

Selecting Map Library

The application map represents the core of our user interface. Two different map libraries were considered for this application. The first option is Google Maps JavaScript API, which provides a set of rules and protocols for interacting with Google's mapping services. The second option is the Leaflet library, which is similar in providing a set of tools for manipulating the rendered maps and an API for fetching the maps from tile providers like OpenStreetMap and MapBox.

[1] `https://nodejs.org/`

[2] `https://www.npmjs.com/`

[3] Technically speaking, React.js is not a framework for JavaScript, but rather a library, since it provides a collection of tools without enforcing strict rules or structures for building the entire application.

In terms of functionality, neither of these two map libraries is better than the other since both provide a similar set of extensive tools. However, it was decided to use Leaflet in the project due to the following reasons:

- It represents an open-source and lightweight library, hence providing more transparency in how it works under the hood and allowing us to achieve higher flexibility and a lower rendering time with it.
- It fetches tile data from multiple providers, hence allowing us to choose the one that better suits our needs.
- The *marinetraffic* website already uses this library for its UI, which can help us gain some inspiration for our project.

3.1.5 Final System Design

The mentioned technologies were connected to a coherent functional system. The design of it is presented in Figure 3.1. It can be split into the following parts:

- Simulator (Java)
- Event Messaging Queue (Kafka)
- Stream Processing and Anomaly Score Calculation (Flink)
- Web Server (Spring Boot)
- Database (Druid)
- Frontend (React.js)

The Simulator is the first part of the system. It is an application that simulates events that mimic historic AIS signals. Those signals are sent by ships in periodic intervals to the Kafka Messaging Queue. As the deployed system would real-time world AIS data, the Simulator is not run in the deployed system. Thus, it is a separate process not part of the web server.

Then, the Kafka Messaging Queue takes incoming produced data and stores it in topics (queues). Even though the only producer is the Simulator, Kafka Queue could handle more if needed. For instance, multiple military radars sending information from the ships could be added as separate producers to the Kafka Queue. Since Kafka needs to be started as a separate process, it is detached from the web server.

The next step for the data flow is the Flink stream processing. This part of the code is a consumer of the Kafka Queue, meaning it takes a stream of events from it. The taken stream is split according to the vessel ID, and then each ship's anomaly score is calculated. A past state is used to calculate the anomaly score for a new data entry, which is provided by the Flink stateful map functions. In addition, during the processing of AIS signals, notification data is calculated, and the current ship details objects are aggregated. The new details are written to the Kafka topic, and also saved to the Druid database. For a more detailed description of the steps done in the Flink processing, refer to the Appendix D.

The communication between the client and the server is done through a REST API and WebSockets requests. To handle these, a web server was implemented using the Spring Boot framework and structured according to the layered architecture: controllers, services, and core domain code. The controllers handle the API endpoints. When a request is received, the controller calls service classes, which call the internal classes to handle the logic of the request.

The Apache Druid database is used for storing ship signals and score histories. The database is set to supervise the Kafka topic to which the calculated anomaly scores, along with the corresponding AIS signals, are sent. This way, web servers are able to query the database when the ship's historical data is needed. For example, such a functionality is useful when services need to retrieve the ship's anomaly scores throughout the track so that the colourful trajectory can be displayed in the frontend.

The frontend is the final part of the design and is the only visible part to the system user. Its implementation relies on the React.js framework. As a simplification, in Figure 3.1, the backend part of the frontend is

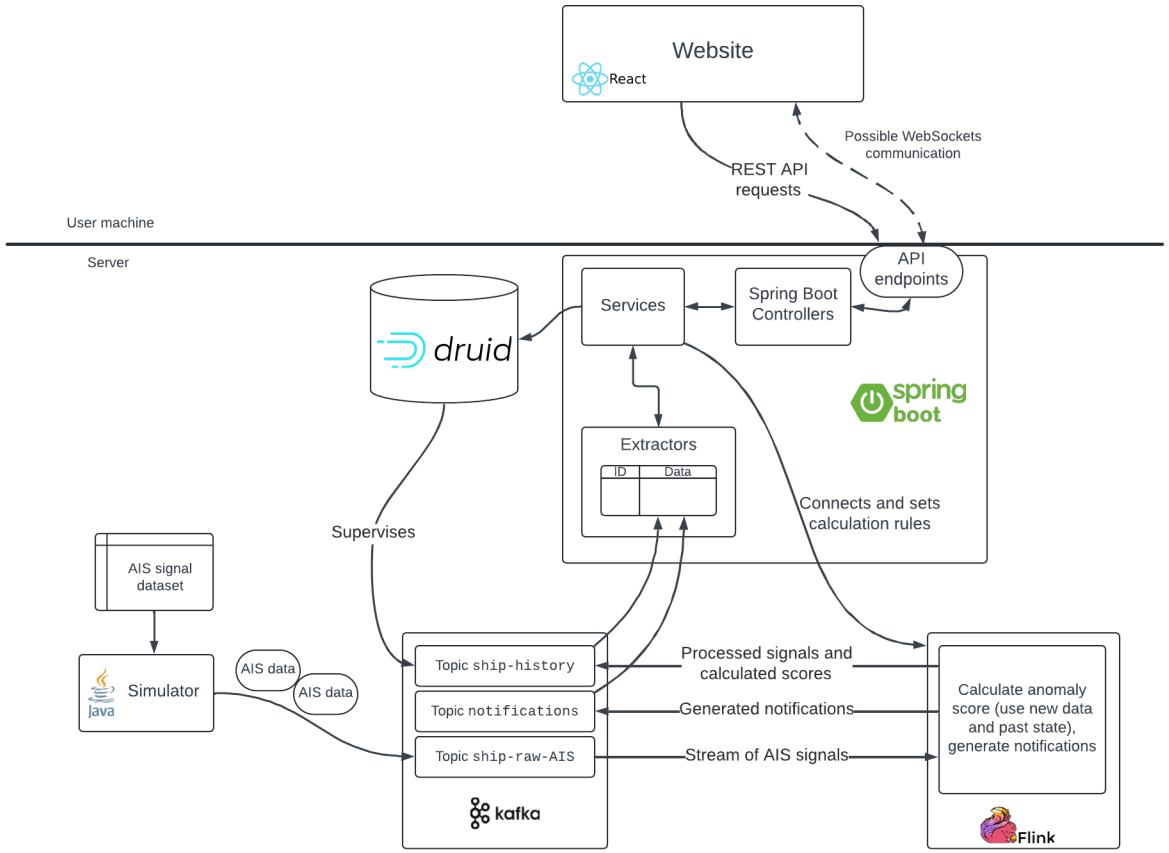


Figure 3.1: *System Design Layout for the whole Track Anomaly Detection application. Boxes represent separate steps and components in the system, while the arrows represent the data flow.*

skipped. However, when deploying, the React server should be started on the backend, so that the client computer can retrieve the resources, such as HTML pages, CSS, and JavaScript code.

3.2 Outline of the Anomaly Detection Algorithm

When designing the system's anomaly detection techniques, the team investigated what should be considered an anomaly and derived concrete implementation for the corresponding definition. The anomaly detection algorithm design is extensively presented in Appendix B.

3.2.1 Defining Anomalous Behaviour in Maritime Traffic

To properly understand what is referred to as an anomaly, the team met with Lieutenant Martijn van Diessen, who is involved in training future naval operators and has extensive expertise in this domain. This meeting was a crucial milestone in the planning of the project. The complete meeting overview is included in the team's documents repository[4].

The definition of an anomaly depends on the context of the corresponding mission executed by the naval operators. The context of the mission might include different parameters, such as the area being investigated, the corresponding weather and sea conditions, the surrounding grounds, the intended goal of the operation,

[4]`https://gitlab.ewi.tudelft.nl/cse2000-software-project/2023-2024/cluster-w/18a/documents/-/blob/main/navy/vanDiessen-minutes.md`

etc. Many extensive examples were given to support this claim; for instance, if the mission is to enforce an embargo, only merchant ships can be anomalous. In such a situation, the operator would focus not on behaviour but on where the ship came from. In a time-of-war scenario, the system should only consider warships. Furthermore, the anomaly definition is also location-dependent; a ship in the middle of the ocean is more likely to be considered an anomaly than a ship in the middle of a small sea. Another example would be the possession of weapons; a ship having weapons in the North Sea is a lot more suspicious than in the Caribbean, where having weapons is a normality. So, an anomaly does not have a clear definition; it depends on the mission's scope.

A particular mission scenario consists of several behaviour rules aggregated together. Implementing detection algorithms for full-scale missions such as those mentioned in the previous paragraph was not a requirement for our application. The team focused on implementing a few such rule components, which can be easily extended with new ones and included within the anomaly detection pipeline for particular missions executed by the naval operators. After the discussion with Lieutenant Martijn van Diessen, seven rules were extracted from the context of the available data set:

- Ships, except the high-speed ferries, exceeding 30 knots could be suspicious.
- A ship that is constantly manoeuvring and doing a lot of speed changes could be suspicious. This does not include fishing vessels, which could do a lot of manoeuvring without being suspicious.
- Contacts that do not provide AIS information but are present on radar and other modes are suspicious. That is because the vast majority of ships must have AIS transmitters turned on.
- Ships, excluding the fishing vessels, that do not go in the direction of their destination port could be suspicious.
- Ships that do not transmit their AIS signals for more than 15 minutes could be anomalous.
- A ship changing its AIS information (such as its length) in the middle of the journey should be considered anomalous. Generally, merchants or fishing ships are not allowed to do that.
- Russian/Iranian ships that move around naval bases could be suspicious.

Regarding the accuracy of the anomaly detection algorithm, it was emphasised that such a system should produce more false positives than false negatives. This is due to the fact that not detecting a potential anomalous ship can lead to severe consequences. In contrast, in the case of a false alarm, even though some resources might be spent to check the corresponding ship, the overall consequences are not severe.

For the explanations generated for the computed anomalies, not providing any explanations would likely result in the operators not trusting the system, especially in cases where the operator can not understand why a particular ship would be labelled as an anomaly. This would result in the operators not using the system (since it cannot be trusted). Therefore, making the computed explanation as detailed as possible represented a top priority throughout the course of the project.

3.2.2 Considered Alternatives

Even though making a system to be distributable was more important, the team still considered integrating AI-supporting frameworks into our application. In particular, as suggested by the team's coach, some options considered were Tensorflow Serving[5], TorchServe[6], and Ray[7]. As for providing explanations for the outcomes of the integrated Artificial Intelligence models, SHAP[8] was researched. These tools were not integrated within the system but represent a good option to enhance the application's anomaly detection capabilities. Please refer to Chapter 7 for further discussions on enhanced machine learning techniques.

[5] `https://www.tensorflow.org/tfx/guide/serving`
[6] `https://pytorch.org/serve/`
[7] `https://www.ray.io/`
[8] `https://shap.readthedocs.io/en/latest/`

3.2.3 Final Implemented Algorithm

The anomaly detection algorithm consists of four components. Each component is independent, accounting for 25% of the total anomaly score. If a particular component is not triggered, then it adds up 0% to the final anomaly score, and otherwise, it adds 25% with no intermediate state. The implemented components are the following:

- **Speed heuristic:** takes into account the reported speed and the speed computed by the system based on geographic location and time information. The component triggers if any of the following is true:
  - The computed speed reported from the AIS signals is greater than 30 knots.
  - The computed acceleration is greater than 300 knots/hour.
  - The absolute difference between the reported speed and the computed speed is greater than 10 knots.
- **Turning heuristic:** takes into account the reported course and heading values from our data set. The component if any of the following is true:
  - For consecutive AIS signals, the circular difference between heading values is greater than 40 degrees.
  - For consecutive AIS signals, the circular difference between course values is greater than 40 degrees.
  - No heading value was reported (heading is 511).
- **Signal timing heuristic:** takes into account the timing between various AIS signals. The component is triggered if the following is true:
  - The difference between the last two emitted AIS signals is greater than 10 minutes, and the computed average speed over this time frame is more than 3.2 knots.
- **Manoeuvering heuristic:** The component checks if a ship did a large amount of manoeuvers in a short interval. The component is triggered if and only if the following rule is true:
  - The number of turns of the ships that exceed 40 degrees exceeds 10 in the last hour.

3.3 User Interface

This section describes the planning, motivation, and final results of the developed User Interface. In particular, the first part (3.3.1) includes short drawn frames indicating the initial design that was decided upon at the beginning of the project. The next subsection (3.3.2) contains more sophisticated drawings derived later in the project. Finally, the last part of this section (3.3.3) includes screenshots of the final implemented user interface.

3.3.1 Initial User Interface Design

At the project's initial phase, it was decided to use a simplistic design while leaving more sophisticated options for later. In particular, we were choosing between the design depicted in Figure 3.2 and Figure 3.3, and decided first to implement the second option, which was more straightforward. At the same time, it was concluded that the second option, where all of the main application components are floating above the map, was the desired final outcome, which should be reached by the end of the project. That was partially impacted by the client's feedback, informing us that it is desired to see as much of the map as possible at a given time. As can be seen in further sections, the goal of implementing the more sophisticated design was achieved later in the project. The implemented initial version can be seen in Figure 3.4.

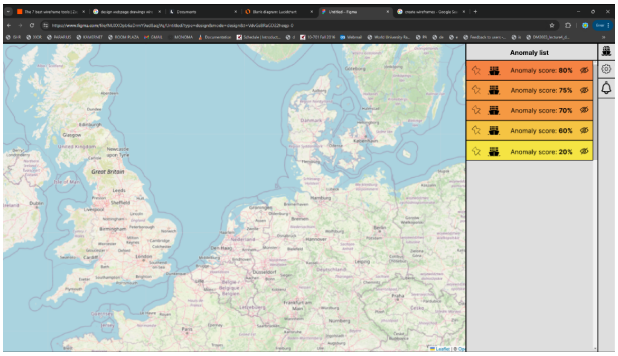


Figure 3.2: *Initial blueprint for the simplistic version of the interface design. Note that this is just a sophisticated drawing, not an implemented website.*

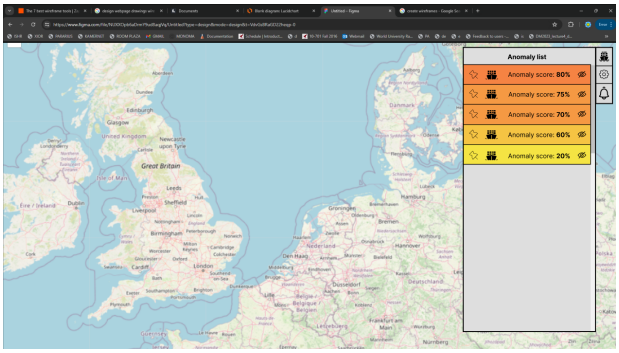


Figure 3.3: *Primary vision blueprint for the final design of the application. It includes most of the applications floating above a map in order to display as much of the map as possible. As shown in later sections, the final design, implemented two months later, matched this one very closely.*

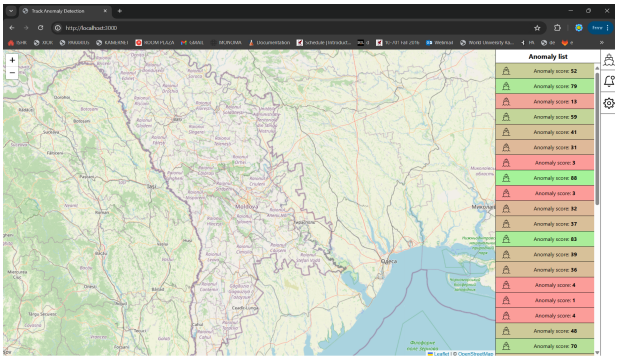


Figure 3.4: *The first version of the implemented design. In particular, this is a screenshot of a functioning application. It was implemented based on the wireframe from Figure 3.2.*

3.3.2 Improved User Interface Design

In the second half of the project, we got to redesign the user interface. In particular, as we added more features to the application, the design became more inconsistent and crowded. Since originally, it was intended to create a sophisticated layout, we decided to create new drawings for it and implement them.

After much discussion and iterative improvements, multiple wireframes were prepared for implementation. Some of them are described in the following figures. For instance, figure 3.5 denotes how the anomaly list was decided to be implemented. The core ideas of this design closely match the initially planned ideas from Figure 3.2. Certain other parts of the design are depicted in Figures 3.6, 3.7 and 3.8. Note that we had prepared nine wireframes in total for the vast majority of pages, but for brevity, only four of them are depicted in the report.

3.3.3 Final Implemented User Interface

After finalizing the drawings, the intended user interface had to be implemented. That was done relatively fast due to the high level of detail of the wireframes that were prepared. The final implementation was very close to the intended design, with a few differences with respect to the colour scheme. In particular, we used white colour instead of grey in all of the designs. Some screenshots of the final implemented design can be found in Figures 3.9, 3.10, 3.11 and 3.12.

3.4 Overview of the Completed Requirements

At the end of the project, most of the requirements were completed. To be more precise, all *Must* and *Should*, and several *Could* requirements were implemented. For a complete list of the requirements, refer to the Appendix A, where all the completed requirements are marked.

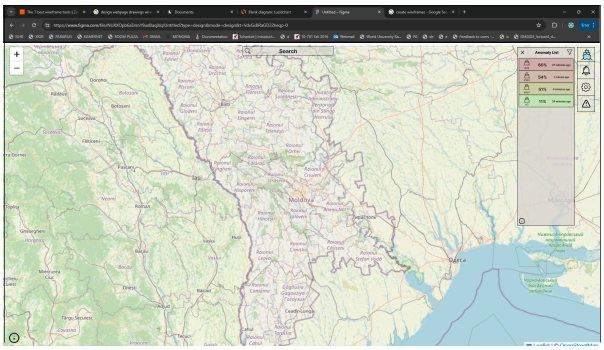


Figure 3.5: *A wireframe denoting how the final version of the anomaly list was supposed to be implemented. In particular, the anomaly list had to contain a list of ship entries, each colour-coded with respect to their anomaly scores. Every ship entry had to contain a shortened version of the ship's ID, its anomaly score and how long ago its last signal was received. Clicking on an entry in the list leads the application to the object details page, partially described in Figures 3.7 and 3.8. The drawing also depicts that the whole application had to be floating on top of the map and also be partly transparent.*

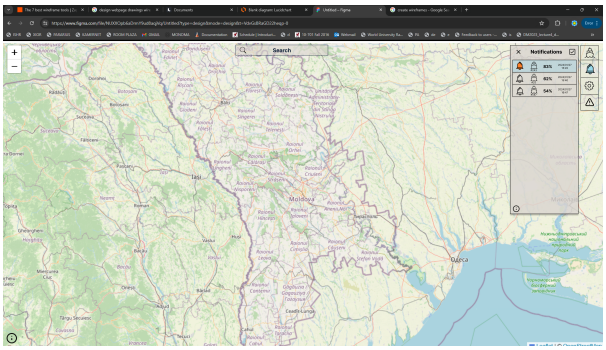


Figure 3.6: *A wireframe denoting how the final version of the notifications list had to look like. Each notification entry includes the time the notification was sent, the ship's ID that generated the notification and whether or not this notification was read. Clicking on the notification entry leads to the notification details page.*

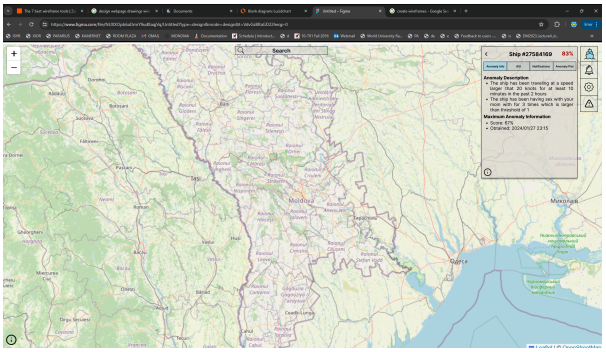


Figure 3.7: *A wireframe denoting how the ship details page had to look like. It denotes the current information for the ship, including the calculated anomaly details (in this screenshot in particular), but also with the option to choose and view the AIS information of the ship, the notifications assigned to it and an anomaly plot, which denotes how the ship's anomaly information changed over time.*

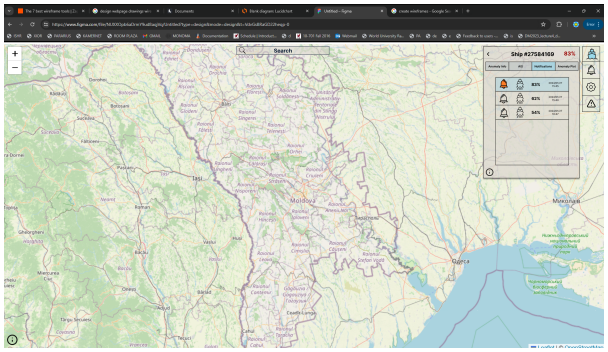


Figure 3.8: *A wireframe denoting how the notification list for a particular was supposed to look like. This sub-page is supposed to be opened when a ship is selected and the "notifications" tab is clicked. It provides information on what notifications were generated for this particular ship.*

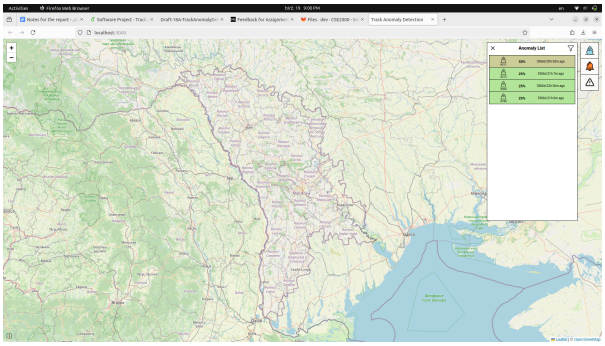


Figure 3.9: *A screenshot of the final application denoting how the anomaly list page looks like. It closely matches the designed implementation from the Figure 3.5*

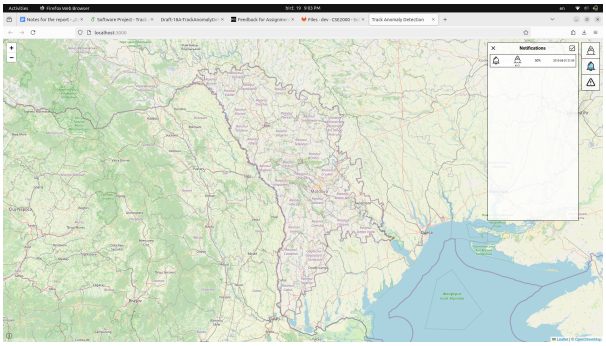


Figure 3.10: *A screenshot of the final application denoting how the notification list page looks like. It closely matches the designed implementation from Figure 3.6.*

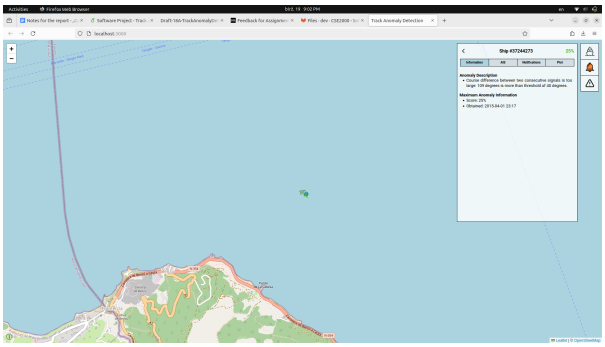


Figure 3.11: *A screenshot of the final application denoting how the ship details page looks like.*

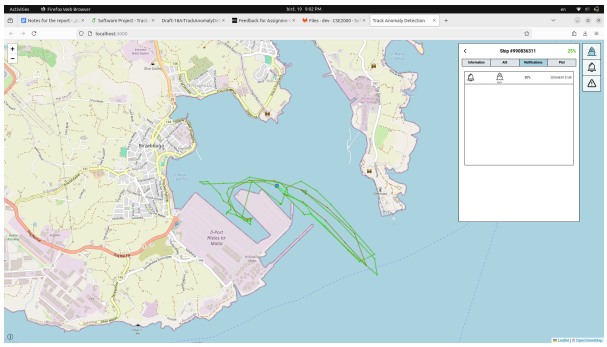


Figure 3.12: *A screenshot of the final application denoting how the ship details page looks like when a list of notifications for a particular ship is displayed.*

Chapter 4

Quality Assurance Analysis

To achieve high results, our team continuously applied quality assurance measures. This chapter aims to present these and show how such measures helped improve our application. In Section 4.1 our testing methodology is explained in detail. Other additional technical tools are described in Section 4.2. The performed anomaly detection performance evaluation is discussed in the next section (4.3). The following section (4.4) presents the feedback from Stakeholders and Supervisors and explains how we used it to improve our solution. Finally, documentation matters are discussed in Section 4.5.

4.1 Testing Methodology

Testing is performed to ensure that the system adheres to both functional and non-functional requirements. Our team applied different strategies for each requirement, such as reaching enough branch coverage and ensuring system scalability and distributability. The four strategies applied by our team are presented:

- Unit and Mutation Testing (4.1.1)
- Integration Testing (4.1.2)
- Scalability Testing (4.1.3)
- Distributability Testing (4.1.4)

4.1.1 Unit and Mutation Testing

One of the testing techniques applied was Unit testing. Our team had a strict policy to write unit tests for any feature immediately after implementation. This helped our team prevent any software errors between the refactorings and maintain the required 85% branch coverage. In addition to the branch coverage, the statement coverage was monitored constantly. Even though it was not part of the requirements, it was important for our team since only the branch coverage is not an accurate metric, as it does not take into account the methods that do not have any branches (*if* or *switch* statements). To perform unit testing and obtain the coverage statistics we have used multiple tools: *JUnit*[1], *AssertJ*[2], *JaCoCo*[3], and *Jest*[4].

In addition, mutation testing was performed to write effective tests. During this type of testing, *mutants* (faulty code changes) are created to see whether our tests can detect erroneous code. Having a high mutation coverage ensured that the tests were reliable. As with the unit testing, our team applied mutation testing

[1] `https://junit.org/junit5/`
[2] `https://assertj.github.io/doc/`
[3] `https://www.eclemma.org/jacoco/`
[4] `https://jestjs.io/`

continuously, especially because having 85% mutation coverage is one of the requirements. To perform this type of testing and collect statistics, *PiTest*[5] tool was used.

In the end, our code coverage metrics are as follows:

- **Backend**: 98% statement coverage, 96% branch coverage, 96% mutation coverage.
- **Simulator**: 100% statement coverage, 100% branch coverage, 100% mutation coverage.
- **Frontend**: 59% statement coverage, 54% branch coverage, out of which:
  (a) UI parts: 38% statement coverage, 32% branch coverage.
  (b) Non-UI parts: 100% statement coverage, 100% branch coverage.

4.1.2 Integration Testing

Integration tests are important for our application. As the system consists of many interconnected components, such as Kafka Queue, Anomaly Detection Pipeline, and the web server (see Section 3.1.5), the way these components interact must also be tested to ensure that they work as expected. This cannot be done by simple unit testing, and a different strategy must be implemented.

The construction of the integration tests is more complicated than the unit tests. The implementation of an integration test includes spawning a local Kafka and a local Flink clusters (using *Embedded Kafka ZKBroker* and *Flink Testing Utils*), and then fake ship AIS signals are sent to Kafka. These are then processed by the Anomaly Detection Pipeline, and the web server endpoint is called to assert the expected result. In this way, the whole system integration is being tested, from the processing events at Kafka to calculating scores inside the Flink pipeline to web server endpoint functionality.

There are only two integration tests, each taking about 40 to 50 seconds, most of which is taken by instantiating the local Kafka and Flink clusters. Because of the time tests take to run, multiple related scenarios and assertions are combined into a pair of tests. This way, running the tests takes less time, but integrating the system components is still being asserted thoroughly.

In these integration tests, there are various scenarios that are tested through integration testing. One part of the first test checks if the system correctly calculates the score for a ship, and another part checks if the system can handle incorrect (non-JSON) strings being sent to the Kafka topic. The other test checks if the scores are calculated correctly for several ships that are anomalous based on different heuristics.

4.1.3 Scalability Testing

Scalability testing was performed to ensure the system was scalable, which was one of the requirements. For that, our team wrote a stress testing application that produced the chosen amount of random signals generated per second. Then, certain places in the pipeline or the frontend could be inspected and checked for throughput bottlenecks.

The system had to be improved to handle the required signal amount. The *should* requirements required it to handle at least 5 000 signals per second, while the system itself could handle at most approximately 300. The reason for that was found to be the use of framework *KafkaStreams* and poor rendering of the map in the frontend. These bottlenecks were fixed by exchanging *KafkaStreams* with immediate *Flink* computation and rendering ships in clusters in the frontend. After these changes, the final system can finally handle at least the required number of signals.

4.1.4 Distributability Testing

To prove the system could be scaled to handle almost any amount of incoming signals, we had to ensure the critical system's components could be distributed. Due to how the application was designed, distributing the messaging queue and the database was trivial since both Kafka and Druid were run as completely

[5] `https://pitest.org/`

independent processes. Furthermore, neither the messaging system nor the database were the performance-limiting components at any time in the development process. Therefore there was no focus on distributing the two mentioned components.

On the other hand, the main bottleneck of the application was the anomaly score calculation part, entirely performed in Flink. This part required distribution to make the bottleneck larger and, as a result, improve the system's throughput. Therefore, a significant effort was made to allow Flink to run on machines at the same time. Multiple configuration decisions and some refactoring had to be made, described in Appendix D.7.

Figure 4.1: *The photograph for the setup of the distributability testing. Two computers were connected into a realistic mini-cluster using a router.*

The final testing results were achieved by connecting two computers into a realistic mini-cluster (the photograph can be seen in Figure 4.1). In particular, we acquired a router with two computers (an HP computer and a MacBook) which we used to create a simple cluster. Then, Flink was configured to run on two machines and performance was compared. The final result was that the throughput rose from approximately 5 000 signals per second to around 8 000 signals per second by simply adding an additional machine into the cluster. Therefore, we successfully tested and proved that our application could be distributed and that its performance improved.

4.2 Additional Technical Tools

Apart from the tools used to test the functionality (mentioned in Section 4.1), other tools were used to assess the quality of our application. These tools include static analysis checkers, linters, and formatters. All of them are included in the GitLab pipeline to run the checks automatically in the merge requests.

The exact tools used are as follows. For the Java code, we have used *CheckStyle*[6] as the code style checker and *PMD*[7] as the static analysis tool. To review the frontend code, *ESLint*[8] linter is used to do static

[6] `https://checkstyle.sourceforge.io/`
[7] `https://pmd.github.io/`
[8] `https://eslint.org/`

analysis, and the *Prettier*[9] formatter to format and structure the code to increase the readability and be consistent.

4.3 Anomaly Detection Performance Evaluation

The lack of anomaly labels for the data that was used for developing the application motivated the decision to derive a rule-based system as a solution for the task of detecting anomalous behaviour. As described in Section 3.2.3, the custom detection algorithm was designed by leveraging both domain-specific information (see Section 3.2.1) and the results of the prior conducted data analysis (see Appendix B.1). Therefore, testing the performance of the heuristics-based algorithm required a set of reference labels which could be interpreted as an external evaluation benchmark.

4.3.1 Establishing the Evaluation Benchmark

For deriving the necessary labels, we leveraged the Isolation Forest algorithm [12]. As an anomaly detection algorithm, the Isolation Forest estimator learns a decision function based on the latent patterns of the provided data and classifies each data point as either an outlier (anomalous) or an inlier (not anomalous). Training the aforementioned model required pre-processing the subsampled AIS dataset (see Appendix B.3), such that during training, the model would be presented with the same informational context as the implemented custom detection algorithm.

Two strategies (see Appendix B.3) were leveraged for estimating the labels of the training examples. First, a global Isolation Forest estimator was trained on the entire training set so that the model could learn the existing anomaly patterns in the context of all the signals. Secondly, a local Isolation Forest estimator was trained for each individual ship from the training sample, such that the respective estimator learned the existing patterns within the individual context of a vessel. The plot describing the distribution of the estimated global anomaly scores for the training set is presented in Figure 4.2.

4.3.2 Evaluation Results

Given the anomaly score distributions retrieved from the system and the estimated global and local score distributions of the Isolation Forest models, we can compare the distributions corresponding to each individual ship by computing the average absolute difference between the scores corresponding to the same AIS signal (see Appendix B.3).

Figure 4.3 and Figure 4.4 display the three anomaly score distributions corresponding to the AIS data of 2 unique ships selected from the training set.

The plots displayed in Figure 4.3 and Figure 4.4 lead to the following conclusions:

- We observe a high correlation between the global score distribution and the system's distribution.
- The system's score distribution follows a similar trend to the Isolation Forest score distributions, correctly identifying the spikes in anomalous behaviour for the analyzed ships.

In addition to the aforementioned conclusions, we leveraged the average values of the deviation between the system's score distribution and the respective local and global score distributions for each ship in the training set in order to obtain a rough estimate of the accuracy metric. As a result, the value of **80.9%** was derived for the defined accuracy metric of the implemented algorithm(see Appendix B.3).

The arguments stated above support the validity of the chosen heuristics for the task of identifying anomalous behaviour in maritime traffic. The positive result of the evaluation also demonstrates that the threshold values utilized within the derived heuristics seem to correspond to the intrinsic general patterns of the data, thus ensuring the correct detection of anomalous AIS signals.

[9] `https://prettier.io/`

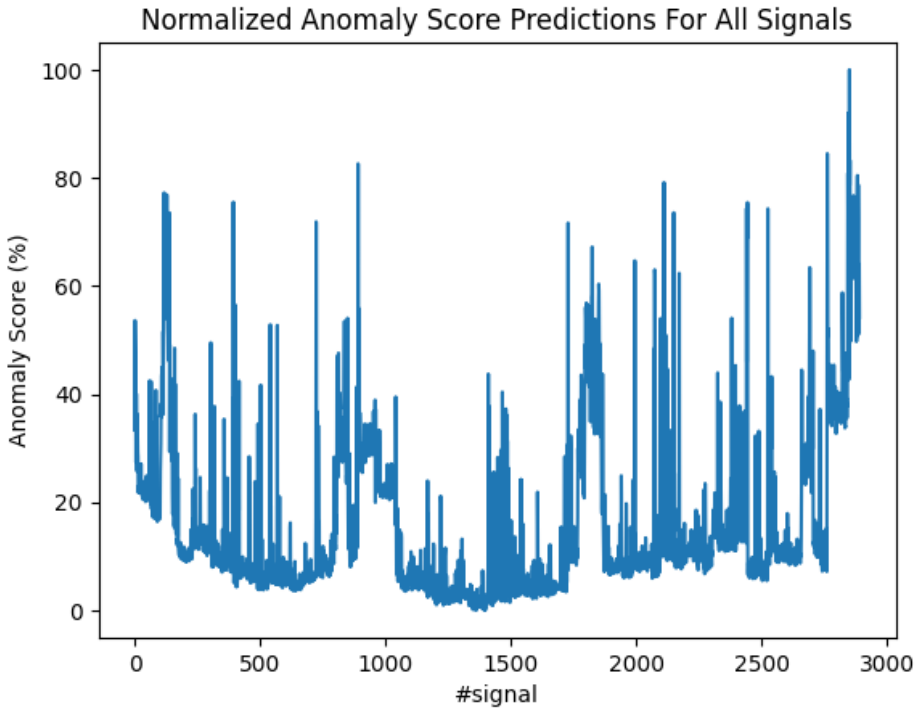


Figure 4.2: Plot of estimated anomaly scores for each AIS signal in the training set predicted by the global Isolation Forest model

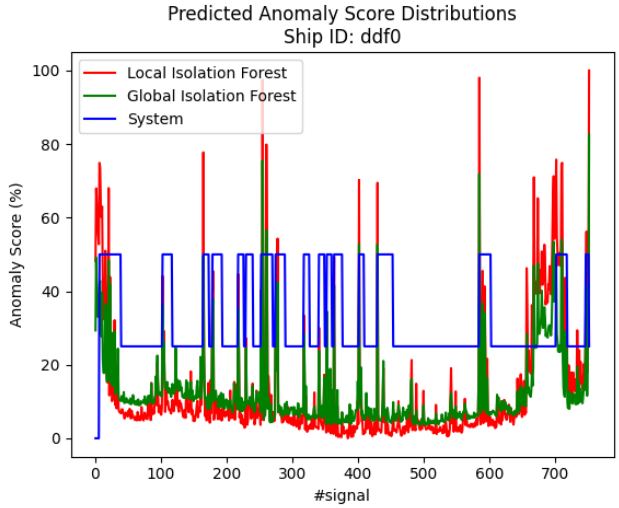


Figure 4.3: Plot of the three anomaly score distributions computed by the algorithm of the system and the trained global and local Isolation Forest models. It can be observed that the anomalous peaks identified by our heuristics closely match the peaks identified by an isolation forest. That could imply that the anomaly forest learns similar heuristics that we have identified.

4.4 Feedback from Stakeholders and Supervisors

Throughout the project, the team maintained a continuous interaction with the main stakeholders of the application (i.e. the representative of the client and the course coordinators) and the supervisors of the team (i.e the coach and the teaching assistant), receiving valuable feedback at each stage of the development

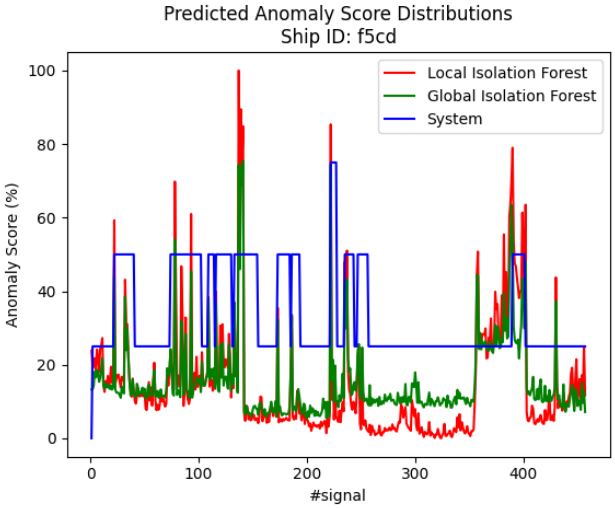


Figure 4.4: Plot of the three anomaly score distributions computed by the algorithm of the system and the trained global and local Isolation Forest models.

process. The main channel through which feedback was received was the regularly conducted meetings, which were independently organized with each person that was enumerated above. In order to receive the desired opinions on our application at each phase of its lifecycle, we requested feedback either by demonstrating the functionality of the application or by preparing sets of narrow questions. Additionally, the team received important pieces of feedback after the midterm presentation, conducted in the fifth week of the project.

The client representative provided the majority of the technical feedback, focusing both on the implementation of application features and system architecture choices. The indications received by the team had the goal of ensuring the satisfaction of the end user of the application (i.e. a maritime traffic operator). For example, the obtained feedback emphasized the need for a correct design of the graphical user interface of the application, such that the attention of the user would be centred around the displayed ships rather than on other graphical components. An important remark made by the client concerned the aspect of testing the accuracy of the developed anomaly detection algorithms, highlighting the importance of demonstrating their validity and performance.

Substantial feedback on the frontend component of the application was provided by the teaching assistant during the application demonstration sessions. In this context, remarks were made about the implementation of specific features of the user interface (e.g. the colour scheme of the graphical user interface) and the security aspect of the application.

4.5 Documentation and Manuals

The application that we have written was documented in several ways. JavaDoc and JSDoc comments were added to the methods, and some parts of the code, especially more tricky ones, were also accompanied by comments. In addition, the steps to set up and run the project were thoroughly documented in the *README* files in our repository. Moreover, our team has created three separate documents: a Developer Manual (Appendix D), an OpenAPI specification (Appendix E) and a User Manual (Appendix F). The Developer Manual provides the full technical specification of the system, providing relevant assistance for a future developer. The OpenAPI specification describes all the endpoints supported by the backend. The User Manual provides detailed instructions for using the application.

Chapter 5

Ethical Considerations

An application that was created is intended to be used by the military, which is a highly trusted, official and government-funded party. Moreover, the application deals with both processing and generating potentially sensitive and even classified data. Given these factors, the application's development and deployment require careful consideration of various ethical implications. Therefore, in this chapter, ethical considerations regarding the project are discussed. In particular, Section 5.1 describes concerns about algorithms' bias. Section 5.2 discusses the considerations that should be taken when storing personal data, while a list of less important ethical considerations is presented in Section 5.3.

5.1 Algorithm Bias

Throughout the development process, several human-defined heuristic models were implemented to effectively detect anomalies in ship tracks. However, this algorithm choice poses a number of ethical considerations.

First, there is a possibility that the trained algorithm exhibits bias towards specific groups of ships, considering the **sensitive nature of the AIS data**. This could occur if certain features of the AIS signal have a greater impact on the final anomaly score than others. For instance, information about the country of registration and the ship's origin port could potentially result in discrimination based on ethnicity or nationality.

To address this problem, we made two key decisions:

- We did not introduce heuristics that relate to sensitive attributes, such as destination port.
- We assigned equal weight to all the heuristics contributing to the anomaly score. This ensures that even if new heuristics introduce bias, they would not unbalance the final anomaly score that is being displayed.

The second ethical consideration for the algorithm is that the heuristics were created by **human beings**. Since the developer team is made up of young male students from Eastern Europe, our perception of what is anomalous may differ from the objective truth due to our cultural and political environment.

To account for personal bias, a meeting was arranged with a direct stakeholder to obtain precise information on what should be considered anomalous. We then used only that information to compute the anomaly score, minimising our own influence.

While all of the mentioned ethical aspects were addressed, several important points need to be clarified:

- The outcomes of the algorithm are computed from the perspective of a Royal Dutch Navy operator. Thus, its main objective is **not** to depict the general moral view of equality but to find suspicious ships that could cause threats and are worth investigating.

- In certain scenarios, being biased towards certain groups could make the model more accurate and desirable. For instance, in case a country is at war, its ships are way more likely to depict suspicious behaviour, which could be related to military operations.
- No direct actions should be taken only from the conclusions drawn by the model. The application is only a tool used by the operators to improve their work efficiency.

Despite the mentioned circumstances, the algorithm was created with the aim of minimising bias. This is important not only due to the potential consequences for discriminated groups but also because bias could make the algorithm less accurate and adaptable. The solution to our problem should be as precise and impartial as possible. Allowing bias towards certain groups could put them at a significant disadvantage, particularly because external circumstances such as geopolitical tensions and wars change over time while the algorithm remains static.

5.2 Collecting Personal Data

Dealing with real geolocation data of ship vessels is a crucial aspect of the project. We decided to store the AIS data in a database as it is essential for parts of the application's functionality. As mentioned in the previous section (5.1), AIS data contains various sensitive features. If this data were to be exposed to the public, it could lead to serious consequences, including diminished trust in the military and exposure to defence secrets. Therefore, responsibly managing this data is a sensitive and critical issue, and there are two considerations to take into account:

- Storing sensitive personal data makes it vulnerable to **exposure to malicious parties**, especially considering the military context. Therefore, a critical aspect of the project should be to store the data in a way that is inaccessible to external parties other than the military personnel.
- Collecting personal data poses the risk of further increasing the **imbalance of power** between governments, authorities, and even social classes, as it could be utilised to a significant extent [13].

Although efforts were made to minimise ethical risks in this project, data safety was not given enough consideration due to time constraints (for a more detailed explanation, see Section 7.4). However, this should not be a high concern, as the current version of the application is not intended for direct military use; it would likely only be used as a source of ideas for improving existing technologies, which already meet data safety requirements.

5.3 Additional Considerations

In addition to the two considerations mentioned above, we took into account three other relevant ethical factors that are worth noting:

- The application is intended for use by a specific government. Before proceeding with the development, it is important to question the morality of creating tools for a particular authority, as the application could potentially be misused for malicious and immoral purposes. It is crucial to consider why and, most importantly, whether the government should be trusted.
- The outcomes of the application (i.e. anomaly information retrieved) may have a significant impact on people's lives. If the system is used extensively and its outcomes are deemed completely reliable, there may be situations that would cause:
    - innocent people being wrongly accused or face consequences for actions they did not commit.
    - actual criminals going unnoticed, which could have been detected if the system were more accurate or if a human operator was monitoring the vessel.

  For that reason, the system should never be trusted to independently make decisions: in the end, it should be a human being making the final decision on whether certain actions should be taken.

- Since the application would be used within the military context, it is crucial to prioritise the security of both personal data (as described in Section 5.2) and the system itself. Ideally, the application should be designed to be unbreachable by malicious parties. However, the team did not prioritise this during development as it was not in the requirements, and the system would likely operate within an internal server on a ship. In such a scenario, the only feasible way to hack it would be physically accessing the server, which is highly improbable. However, this issue could be left as future work for other teams (refer to Chapter 7).

Chapter 6

Collaborative Development Methodology and Process

Throughout the project, a variety of agreements and methodologies were utilized for organizational purposes to help ensure a successful project outcome. Therefore, this chapter presents how the work was structured throughout the ten-week period. In particular, Section 6.1 provides an overview of the general methodology strategy chosen. Section 6.2 discusses the decisions taken for optimizing the process.

6.1 General Workflow Structure: Agile Framework

The Agile framework was chosen as the general development process model. According to [14], using the Agile framework has the following advantages:

- The Agile model allows for **high flexibility** in adjusting the application requirements at each development cycle.
- The Agile model enables incremental and **fast delivery** of application features, which was essential for achieving the MVP (Minimal Viable Product) in the first weeks of the project.
- The Agile model forces continuous testing and integration, ensuring high **application quality** throughout the entire development process.

The length of the cycle was chosen to be one week, as it coincided with weekly meetings with the TA. Moreover, since GitLab was used as a platform to store repositories, its features, such as Issues and Milestones, were used for organizing weekly tasks.

Lastly, as a key part of the Agile methodology, the changes proposed by the client were constantly considered to ensure their satisfaction. Additional functionalities were accepted if they satisfied the following criteria:

- The change did not significantly deviate from the initially defined requirements.
- The required time for implementing the change fits within the existing time constraints.
- The change did not introduce risks like defying data privacy or posing social concerns.
- The change corresponded to the team's current level of expertise, or the knowledge required to implement the change was learnable within the existing time constraints.
- Unanimous support was required to accept the change.

6.2 Optimizing Teamwork Efficiency

Before the start of the development process, a strict **systematic** workflow structure was established to enhance productivity and ensure that all set-out requirements were fulfilled. The decided-upon measures were as follows:

- **Sprint meetings**. At the beginning of **each** sprint, several meetings are held:
  - **Internal meeting**. It is a team-only meeting that consists of:
    - Concluding previous sprint's work. The development branch is merged to the main branch, and a corresponding retrospective document is filled.
    - Planning of the upcoming sprint. A corresponding sprint document is filled. Required GitLab entities (issues, milestones, etc.) are created in accordance with the sprint. All retrospective and sprint documents can be accessed in GitLab[1].
  - **Client meeting.** A meeting with the client is held, where the team's weekly progress and plans are shared, and any necessary questions are asked to ensure alignment between both parties. Detailed notes and agendas for all meetings with the client can be found in GitLab[2].
  - **Teaching Assistant meeting**. During the meeting, necessary information is exchanged, progress is presented, and relevant questions are addressed. All meeting notes and agendas are documented in GitLab[3].
- **Long-term planning**. In the initial weeks of the project, a high-level plan for all sprints was established. By adhering to the initial timeline, the team aimed to ensure the timely completion of all requirements. This plan was occasionally adjusted throughout the project. However, the general points have been strictly followed.
- **Meetings with Project Coach**. Occasional meetings with the project coach have been held once serious questions regarding the technical and research parts of the project have arisen. The agendas and minutes for these meetings can be found in GitLab[4].
- **Division of responsibilities**. Each member was assigned specific minor responsibilities to maintain throughout the project. The team utilized the role-based task assignment model [15] to minimize ambiguities within task ownership. These responsibilities included official communications with other parties (TA, coach, client, etc.), monitoring the rubric, monitoring GitLab activity, document formatting, etc. In addition to this, the team strived to divide the tasks so that each member gets to do the tasks from each project component at some point (frontend and backend). More detailed division of work can be found in the Appendix G.
- **Internal rules**. Before the development process began, a list of internal agreements for general work quality was decided upon. It consisted of technical guidelines that each team member was supposed to adhere to throughout the project. One member was held responsible for constantly monitoring the internal rubric and notifying others if it was not followed. The exact rubric can be found in the Appendix H.

  The workflow agreements proved to be incredibly valuable for the project. They assisted in establishing clear guidelines and expectations for each team member, ensuring that everyone was aware of their roles and responsibilities.

[1] `https://gitlab.ewi.tudelft.nl/cse2000-software-project/2023-2024/cluster-w/18a/documents/-/tree/main/sprints`

[2] `https://gitlab.ewi.tudelft.nl/cse2000-software-project/2023-2024/cluster-w/18a/documents/-/tree/main/meetings/client`

[3] `https://gitlab.ewi.tudelft.nl/cse2000-software-project/2023-2024/cluster-w/18a/documents/-/tree/main/meetings/ta`

[4] `https://gitlab.ewi.tudelft.nl/cse2000-software-project/2023-2024/cluster-w/18a/documents/-/tree/main/meetings/coach`

6.3 Use of Generative AI

Throughout the project, several Generative Artificial Intelligence tools were utilized for minor purposes.

- ChatGPT[5] was used occasionally by several teammates for debugging reasons, in order to understand why certain errors were thrown.
- Copilot[6] was used for programming by one team member which has a built-in AI assistant.
- Grammarly[7] was used to check the grammar when writing the report.

Apart from the mentioned tools, the project was completed without any external assistance from Generative AI.

[5] https://chatgpt.com/
[6] https://copilot.microsoft.com/
[7] https://app.grammarly.com/

Chapter 7

Future Work

The current version of the implemented application is already a valuable tool. However, there exists plenty of room for improvement. This chapter outlines several key areas that could be considered for future work in case the application is extended. Section 7.1 describes potential improvements to the algorithm, Section 7.2 describes the idea of extendability for different users, while the possibility of extending the application to support plane data is presented in Section 7.3. The final section (7.4) discusses data safety enhancements and additional possible improvements are briefly described in Section 7.5.

7.1 Enhanced Machine Learning Techniques

The anomaly detection in our application currently relies on human-defined heuristics. While heuristic-based algorithms are reliable, the efficiency of the anomaly detection process could be improved by implementing more extensive and automated machine learning (ML) techniques. However, a key challenge with ML algorithms is the lack of explainability, which is still an emerging topic and has not yet been extensively researched[16]. Therefore, adding this feature may require thorough research and substantial effort. Some possible solutions that we propose include:

- Using ML techniques for separate heuristics. It would allow the identification of more complex patterns for individual fields, and some form of explainability would still exist, as the operators would be aware of the heuristic that recognized the anomaly.
- Training a supervised ML model with manually labelled data based on human-defined heuristics. However, while this approach is feasible, it is highly constrained by a chosen labelling approach.
- Leveraging existing ML libraries for explainability. There are several open-source libraries, such as *SHAP*[1], designed to explain the outputs of a model based on its structure.

7.2 Introducing Different Users

The current application version does not support a user account system. However, introducing support for different users might be a highly beneficial feature for several reasons:

- In particular scenarios, certain data should only be accessible to specific officers and not all users. For example, if access to the AIS information of certain classified vehicles should be restricted from regular navy personnel, measures should be in place to limit their authorization. This would require support for a user account system.

[1] https://shap.readthedocs.io

- In specific scenarios, certain notifications should only be sent to specific operators. For example, if certain operators are responsible for different groups of ships, their personal notification list should not be cluttered with unrelated ship notifications.
- It is possible that different operators may require personalized and configurable versions of the same or even different algorithms. For instance, if the system were to be used on an anti-piracy mission in the Red Sea, the algorithm configurations may be significantly different compared to, for example, regular anomaly detection in the Baltic Sea. Implementing this, however, would require the system to support dynamically modifiable algorithms, which may be a challenging task considering the distributability of the system.

Although adding user functionality may not be overly complex, enabling the dynamic modification of algorithms could pose a significant challenge to the current system, as it may require redefining how algorithms are computed in general.

7.3 Support for Aircraft Data

The current version of the application can only process and display ship data. However, in the future, it could be expanded to include the functionality to incorporate a completely new type of stream data that represents aircraft. This task would not be extremely difficult, as the current implementation of the application allows for the easy addition of new streams, extending the anomaly score calculators, and extending the user interface. Therefore, adding the planes feature would require making very few modifications to the existing code. However, it is very likely that completely new heuristics or even a new algorithm would need to be designed, as planes represent a completely different form of transportation.

7.4 Data Security

The application does not consider the security of the data it receives and uses. However, given the military context in which the application would be used, the system's security and resilience to malicious parties may be of significant importance. Numerous vulnerabilities in the system exist:

- The database contents, which potentially contain classified information, could be easily accessed by malicious parties, as it is not secured in any way.
- The data flow security between the backend and frontend is not considered at all.
- Misleading AIS data could be sent by malicious parties, which could disturb the application's operations.

Our project's requirements did not include solving any of the above-mentioned issues, which are themselves completely independent problems. Therefore, extensive research should be performed to eliminate the mentioned vulnerabilities.

7.5 Other work

Apart from the improvements mentioned in the previous sections, there are some other enhancements that could be implemented:

- The implemented application currently contains a single static algorithm that is used for anomaly detection. However, in specific cases, such as special navy missions, it would be highly beneficial if the algorithm could be adjusted by the navy operator to meet their specific needs for a particular scenario. However, this addition, similar to introducing different users, potentially requires strict modifications to the current implementation.
- Ship grouping into clusters could be optimized in the frontend. The current grouping technique and its trade-offs are explained in the Map section of the Appendix D.

- The scalability of the backend could be improved. Currently, the service classes in the backend receive data from the Kafka topics through implemented extractor objects. However, it might be beneficial to remove the extractor step and query the database directly. This change should allow the application to process even larger amounts of data.
- The application starting process needs improvement to make it more convenient. Currently, running the application requires cleaning and starting many processes, leading to a time-consuming and inefficient procedure. However, this can be highly optimised by using tools like Docker[2], which makes the starting of many processes a far more timely and manageable task.

[2] `https://www.docker.com/`

Chapter 8

Conclusions and Recommendations

The purpose of this report was to propose a solution for automating the detection of suspicious ships at sea. To achieve this, a software system had to be designed which could manage large loads of ship signals entering the system, calculating anomaly information and displaying relevant information to the end user. Due to the large number of signals sent by the watercraft every second, the system was supposed to be highly scalable and easily distributable across multiple computers. Furthermore, the anomaly scores assigned to ships were supposed to be accompanied by human-readable explanations to ensure that the Navy operators could trust and understand the system.

The final implemented system addressed all of the mentioned technical problems. Firstly, to ensure that it was scalable and distributable, specialized stream processing frameworks were chosen: Kafka as a message queue for handling many large incoming streams of signals, Flink as a score calculation engine and Druid as a stream database. All three of these frameworks are specifically designed to be distributable with little effort. Additionally, to solve the problem of explainable anomaly detection algorithms, a rule-based approach was taken. To be more precise, a list of heuristics was derived based on the provided requirements, data analysis and in consultation with a Navy expert. Inside the anomaly detection system, these rules are applied to all incoming signals for calculating anomaly scores and respective explanations. Finally, the user interface had to be implemented to handle displaying tens of thousands of ships at a single time, which required the use of technical optimizations, ship clustering techniques and making a tradeoff between the amount of displayed information and the performance of the interface.

In conclusion, the system's main goals were reached - all *must* and *should* requirements were implemented, with a fraction of the could requirements done as well. In particular, the final system is able to handle at least the required load of incoming Big Data (5 000 signals per second) while also including an intuitive and fast user interface. Additionally, the anomaly detection techniques are easily extendable and modular. Furthermore, the implemented software was rigorously tested, including proper unit, mutation, integration, stress, distributability and user interface tests covering the vast majority of code and functionalities. Also, an evaluation of the accuracy of our algorithms was performed to test how precise the derived rules were, resulting in an observation that our heuristic-based algorithm closely matches the behaviour of isolation forests, with an above 80% similarity rate. Finally, while developing the application, ethical considerations were taken into account, including but not limited to bias and personal data protection concerns.

For the Ministry of Defence, which is our client, we have two main recommendations. First, since we were not given access to a realistic classified dataset used in the military, we could not fully tailor nor test the system for realistic Navy use. Therefore, we recommend that the Navy developers fine-tune relevant parts of the system in order to fit their needs even better. For instance, they could update the data schema to the one that is used inside of the military. Secondly, we recommend that Navy operators use the system with care. The anomaly detection algorithms that were produced are not perfect and should still be used under human supervision, where the operator makes the final decision. In particular, the goal of the current system is to aid the navy operators and not to replace them.

Bibliography

[1] International Maritime Bureau, Tech. Rep., "2023 Annual IMB Piracy and Armed Robbery Report" icc-ccs.org. https://www.icc-ccs.org/piracy-reporting-centre/request-piracy-report (accessed May 14, 2024).

[2] United Nations Office on Drugs and Crime, Tech. Rep., "World Drug Report 2022" unodc.org. https://www.unodc.org/unodc/en/data-and-analysis/world-drug-report-2022.html (accessed May 14 2024).

[3] European Parliament, Tech. Rep., "Impacts of the Recession on Transport: A Review of Possible Strategies and Measures for Reducing the Effects of the Economic Crisis on Transport" europarl.europa.eu. https://www.europarl.europa.eu/RegData/etudes/etudes/join/2009/419096/IPOL-TRAN_ET(2009)419096_EN.pdf (accessed May 14, 2024).

[4] G. Vilone and L. Longo, "Notions of explainability and evaluation approaches for explainable artificial intelligence," *Information Fusion*, vol. 76, pp. 89–106, 2021. DOI: 10.1016/j.inffus.2021.05.009.

[5] Global Investigative Journalism Network, "Tracking Ships at Sea" gijn.org. https://gijn.org/2018/05/23/tracking-ships-at-sea/ (accessed May 14, 2024).

[6] G. D'Amicantonio, E. Bondarau, and P. H. N. de With, "Utrand: Unsupervised anomaly detection in traffic trajectories," 2024. DOI: 10.48550/arXiv.2404.12712. eprint: 2404.12712.

[7] S. Sabour, S. Rao, and M. Ghaderi, "Deepflow: Abnormal traffic flow detection using siamese networks," 2021. DOI: 10.48550/arXiv.2108.12016. eprint: 2108.12016.

[8] M. Xu, J. Wu, H. Wang, and M. Cao, "Anomaly detection in road networks using sliding-window tensor factorization," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 12, pp. 4704–4713, 2019. DOI: 10.1109/TITS.2019.2941649.

[9] E. Miranda, "Moscow rules: A quantitative exposé," *Agile Processes in Software Engineering and Extreme Programming. XP 2022. Lecture Notes in Business Information Processing*, vol. 445, pp. 15–29, 2022. DOI: 10.1007/978-3-031-08169-9_2.

[10] MarineTraffic Research, "MarineTraffic Automatic Identification System (AIS) Dataset." marinetraffic.com. https://www.marinetraffic.com/research/dataset/marinetraffic-automatic-identification-system-ais/ (accessed June 11, 2024).

[11] M. Levlin, "DOM Benchmark Comparison of the Front-End JavaScript Frameworks React, Angular, Vue, and Svelte" doria.fi. https://www.doria.fi/handle/10024/177433 (accessed May 21, 2024).

[12] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, 2008, pp. 413–422. DOI: 10.1109/ICDM.2008.17.

[13] T.S.E. of Philosophy, "Scientific Research and Big Data" plato.stanford.edu. https://plato.stanford.edu/archives/sum2020/entries/science-big-data/ (accessed May 21, 2024).

[14] Z. Tu, "Research on the Application of Layered Architecture in Computer Software Development" drpress.org. https://drpress.org/ojs/index.php/jceim/article/view/14398 (accessed May 21, 2024).

[15] H. Zhu and Y. Zhu, "Group role assignment with agents' preferences," pp. 605–610, 2017. DOI: 10.1109/ICNSC.2017.8000160.

[16] R. Marcinkevičs and J. E. Vogt, "Interpretability and explainability: A machine learning zoo mini-tour," 2023. DOI: 10.48550/arXiv.2012.01805. eprint: 2012.01805.

[17] DoubleCloud, "Apache Kafka vs. Flink - Choosing the right streaming data platform" double.cloud. https://double.cloud/blog/posts/2023/06/kafka-vs-flink/ (accessed May 21, 2024).

[18] AnalytixLabs, "Flink vs. Kafka: A Quick Guide to Stream Processing Engines" medium.com. https://medium.com/@byanalytixlabs/flink-vs-kafka-a-quick-guide-to-stream-processing-engines-b09dd0e6b8af (accessed May 21, 2024).
[19] Apache Software Foundation, "Apache Flink Machine Learning Library" nightlies.apache.org. https://nightlies.apache.org/flink/flink-ml-docs-master/ (accessed May 21, 2024).
[20] The Apache Software Foundation, "Multi-Language Support in Apache Storm" storm.apache.org. https://storm.apache.org/about/multi-language.html (accessed May 21, 2024).
[21] Macrometa, "Apache Storm: Architecture, API, Limitations" macrometa.com. https://www.macrometa.com/event-stream-processing/apache-storm (accessed May 21, 2024).
[22] StackShare, "Flink vs Samza" stackshare.io. https://stackshare.io/stackups/flink-vs-samza (accessed May 21, 2024).

Appendix A

Application Requirements

A.1 Overview

The formulated requirements are split according to the MoSCoW method. The respective list comprises the initially formulated functional and non-functional requirements of the developed application together with the additions that were performed throughout the development process. The requirements that were added or changed throughut the project are marked with the "*" symbol. The finished requirements are marked with the green background.

A.2 Terminology

AIS Signal - Automatic Indentification System signal which encapsulates important information about a vessel, such as the unique identification, position, course, and speed.

Anonaly score - score that depicts how suspicious a certain vessel is behaving based on the received stream of corresponding AIS signals. The more suspicious the vessel behavior, the higher the anomaly score should be.

A.3 Must Requirements

A.3.1 Functional Requirements

1. The User must be presented with an interactive map on the screen.
2. The User must be able to zoom in/out and move on the virtual map, such that one can freely select the desired region for visual inspection.
3. The map must contain visual indicators representing ships.
4. When the User clicks on a particular ship (visual indicator), its information must be clearly presented on the side of the map:
   (a) If already computed, the anomaly score is presented.
   (b) The User is presented with short explanation of why the track is deemed as anomalous, according to the algorithm. The allowed explanations (for the must requirements) include:
      i. suspicious speed of the ship, for example: when the velocity of the ship exceeds a certain threshold, the ship will be considered anomalous
      ii. suspicious turning (change of orientation) of the ship

iii. suspicious intervals between AIS signals (too large intervals for a ship which is at sea)

(c) All task-relevant pieces of AIS information provided by that ship must be listed: hash of the ship, speed, latitude, longitude, departure port and heading direction.

(d) ***** The associated timestamp of the AIS signal must be displayed in the ISO-8601 international standard with UTC timezone.

5. The User must be able to close the container containing detailed information about a particular ship.
6. The visual indicators of ships on the map must have a certain color based on the corresponding anomaly score: the higher the anomaly score, the more red the assigned color should be.
7. A separate list of ships which currently have a calculated anomaly score, must be visible for the User by the side of the map. Each entry in the list should include:
   (a) The anomaly score of the transport vehicle.
   (b) ***** Ship identifier: the ship ID, the departure port.
8. ***** The User must be able to collapse and reopen the information side-tab located at the side of the map.
9. ***** The information side-tab must be information-dense such that the User has a better overview of the map.
10. When the User clicks on an entry in the list, the action must have the same effect as if the ship (visual indicator) was clicked on the map (i.e. the 4th requirement above).
11. When the User clicks on an entry in the list, the map must zoom in and center around the visual indicator of that ship.
12. The entries in the list must be sorted descendingly according to the anomaly score.
13. AIS information must be updated in real-time upon receiving a new AIS signal:
    (a) The position of the ship is automatically updated on the map.
    (b) The anomaly score is automatically updated in the list of ships.
    (c) The anomaly score is automatically updated in the information of the ship, in case it is currently viewed by the user.
    (d) The list will be sorted after each update to the score.
14. A ship must no longer be displayed on the map if it has not transmitted any AIS signal for 30 minutes. 30 minutes were chosen, since typically ships transmit signals every 2 seconds to 3 minutes[1].

A.3.2 Non-Functional Requirements

1. For the implementation of the must requirements, the dataset is taken from *marinetrajfic.com*: the name of the dataset is "Open Research Maritime Vessel Tracking Data"[2].
2. For the implementation of the must requirements, a data simulator based on the mentioned dataset must be implemented such that it simulates the stream of incoming live AIS signals.
3. Above 85% branch test coverage must be achieved for the backend of the application.
4. Mutation testing must be performed for the backend testing of the application. Above 85% of nonequivalent mutants must be killed.
5. Non-UI parts of the front-end must be branch tested with a coverage of higher than 85%.

[1] `https://arundaleais.github.io/docs/ais/ais_reporting_rates.html`
[2] `https://www.marinetraffic.com/research/dataset/marinetraffic-automatic-identification-system-ais/`

6. The application's front-end must work on the latest stable version of Google Chrome (Chrome Stable 124).
7. The code must be well-documented. Specifically, (at least) the following is needed: documentation for building and running the project, explanation of high-level architecture, comments in the code, and OpenAPI specification.
8. The application must be capable of handling at least 2 000 AIS and anomaly score updates per second, assuming simple rule-based anomaly detection algorithms.
9. The GUI of the application must support fast transitions (less than 1 second) between different zooming levels.
10. The backend must work on Linux (Ubuntu 22.04).
11. The Anomaly Detection algorithm/pipeline must have a modular nature.
12. The stream sources of AIS updates must have a modular nature.

A.4 Should Requirements

A.4.1 Functional Requirements

1. The application should send a notification in case a high-certainty (above 90%) anomaly is identified. A notification should include:
   (a) The ID of the ship that became anomalous
   (b) The time and date when the ship became anomalous
   (c) Anomaly score
2. The User should have access to a history list of all notifications. New notifications should be appended to that list.
3. Once the User hovers upon a visual indicator of a ship, (less detailed) information about it should appear (including its ID, speed and current anomaly score).
4. When the User zooms out the map, in order to avoid the cluttering of the vehicles in dense areas, they should be aggregated and displayed as a single summarizing visual element on the map. The assigned summary should include the maximum anomaly score of a ship within the cluster. (This requirement was updated during the process)
5. The User should be able to customize the filtering of the ships: display only the ones that have an anomaly score above a specified threshold.
6. Once the User clicks on a visual indicator of a ship, its path trajectory should appear on the map.
7. The trajectory of the transport vehicle should be gradient-colored depending on the anomaly score at a given location (for the locations corresponding to higher anomaly scores, the track should be more red).
8. The path trajectory should be automatically updated, in case the User is currently inspecting it.
9. The explanations for why a ship is regarded as an anomaly should be extended and improved:
   (a) The previously added explanations should be enhanced: instead of simply identifying which parts of a trajectory are suspicious (suspicious speed, suspicious turning, etc.), now explanations for why they are suspicious are added. For instance: suspicious speed: the ship is going over 18 knots.
   (b) New categories for why a ship is anomalous should be added. To be done after communicating with navy operators.

10. The visual indicators should distinguish between moving and stationary ships (i.e. a moving ship should be displayed as an arrow, while a stationary ship is displayed as a circle).
11. The User should be able to click on the notification to get more detailed information about it. When the notification is clicked on, the detailed information about the vessel at the moment in time that the anomaly happened should be given (AIS information + anomaly score explanation) and that point is marked in the map. The map should be centered around the corresponding ship and its trajectory should be shown.
12. The detailed information of a transport vehicle should contain a list of all notifications related to it.
13. The detailed information of a ship should contain the maximum anomaly score ever obtained.
14. The User should be able to view a plot of how the anomaly score of a single ship changed throughout time within the detailed information of a transport vehicle.
15. * The User should be informed of the errors that occur in the frontend of the application (software bugs and server connection issues) by the means of a separate field, which does not take space on the screen, unless opened from the side-tab.
16. *There should be an information button in the UI, which, once clicked, would show the explanation of the displayed window. It should include the description of what can be seen in the current page and explanation of what functionality it has.
17. * Once a ship's indicator is clicked and centered, it should remain centered in the map while the ship is moving, unless the user zooms in/out or moves using the mouse.

A.4.2 Non-Functional Requirements

1. Integration tests should be written for the back-end of the application.
2. The application should be capable of handling at least 5000 AIS and anomaly score updates per second, assuming a simple rule-based anomaly detection algorithm.
3. A stress test should be written to ensure the scalability of the backend that it can handle 5000 AIS per second.

A.5 Could Requirements

A.5.1 Functional Requirements

1. The User should be able to search for a particular transport vehicle.
2. There should be a search bar for notifications.
3. The User should be able to select an anomaly threshold for receiving the notifications.
4. The anomaly score calculation algorithm should be implemented so it does not fluctuate.
5. The User should be able to hide/unhide some transport vehicles on the map that they are not interested in.
6. The User should be able to silence notifications for irrelevant transport vehicles.
7. Different Users should have different accounts.
   (a) Users should be able to connect with their accounts.
   (b) Pinned items should be different per each User (each User only sees the pinned transport vehicles that they pinned).

(c) Hidden tracks should be different per each User (for each User, only the tracks that they have hidden are not shown, and all other tracks are shown, regardless of whether other Users have hidden them).

8. The User should be able to request a specific past state of the map.
9. The User should be able to choose to view the real-time version of the map.
10. The User should be able to choose some transport vehicle to pin. For the pin, the information for the pinned vehicle will be displayed both at the top of the list of anomalies and at the list of notifications.
11. The application should be able to display and process the anomaly scores and trajectories for air traffic data: planes, and helicopters.
12. ***** The list should show a representative icon of the type of transport vehicle next to it to indicate whether the track is of a ship, plane or other transport vehicle.

A.5.2 Non-Functional Requirements

1. Distributability should be tested by connecting the team's laptops to a cluster and testing distributability.
2. The history of the anomaly scores should be saved in the database.
3. All components in the back-end are additionally integration-tested.

A.6 Will Not Have Requirements

A.6.1 Functional Requirements

1. The developed software system will not support the processing and visualization of terrestrial traffic data.

A.6.2 Non-Functional Requirements

1. The software system will not implement any measures to ensure the safety of data that is processed and persisted into the data base.
2. The application will not include any security features to protect against unauthorized access or other security threats.

Appendix B

Data Analysis, Anomaly Algorithm Design & Evaluation

This appendix focuses on presenting the results of the data analysis conducted on the available ship data set, motivating our chosen anomaly detection heuristics and evaluating the accuracy of the final implemented algorithm for the Track Anomaly Detection system. Section B.1 presents general information about the data set, Section B.2.1 motivates the threshold choices for the implemented heuristics, and Section B.3 provides an accuracy evaluation of the implemented algorithm reported to a baseline isolation forest-based anomaly detection algorithm. Every claim is supported by visual plots and graphs. The analyzed dataset was extracted from MarineTraffic Research website [10]. For the reference code, more plots and explanations please refer to the team's Research repository[1].

B.1 General Data Information

The available data set represents a selection of AIS data collected from the MarineTraffic coastal network covering areas of the Mediterranean Sea. An AIS(Automatic Identification System) signal is a form of communication maritime vessels use to broadcast information about themselves to other ships and coastal authorities.

In B.1, the corresponding features of the available dataset are represented.

Feature	Explanation
SHIP_ID	The anonymized ID of the ship
SPEED	Ship's speed measured in knots
LON	The longitude of the current ship position
LAT	The latitude of the current ship position
HEADING	The direction in which a ship is pointed often referred to as the direction the "nose" of the ship is facing
COURSE	The actual direction in which the ship is moving. It is different from the heading since some external factors, such as wind, rain, or sea currents, will alter the actual direction of the ship
TIMESTAMP	The time the message was sent (UTC)
DEPARTURE PORT NAME	The name of the last port visited by the vessel

Figure B.1: The features and their explanations of the data from the public MarineTraffic Research dataset of AIS signals.

[1] https://gitlab.ewi.tudelft.nl/cse2000-software-project/2023-2024/cluster-w/18a/research

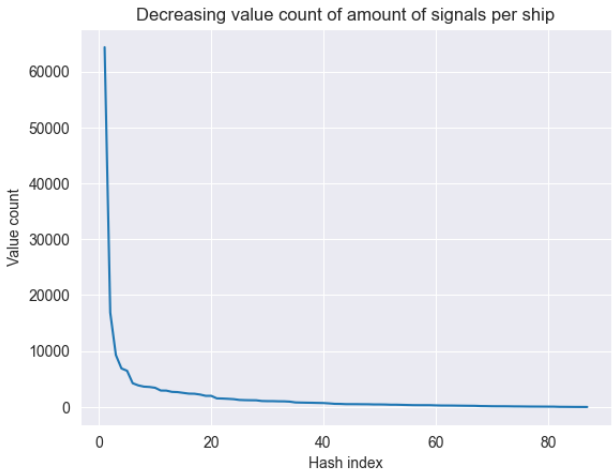


Figure B.2: Number of AIS signals emitted by each ship in decreasing order.

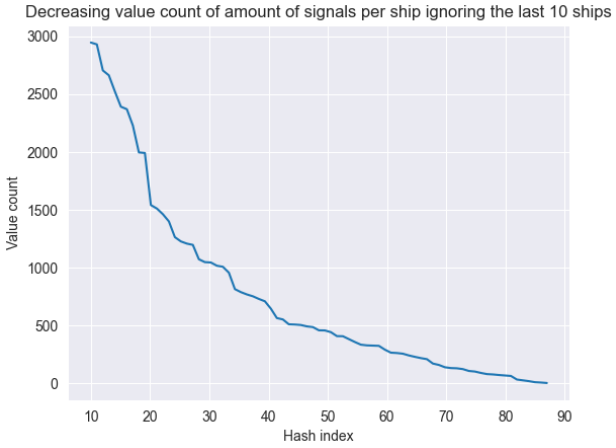


Figure B.3: Decreasing amount of AIS signals emitted by each ship, not considering the top-10 ships that emitted the most signals.

In total, the dataset consists of 180934 signals. The signals belonging to the dataset were generated by 87 ships. The first signal was emitted on the 1^{st} April 2015, and the last was on 28^{th} April 2015.

A 511 heading value means that there is no heading value, according to the MarineTraffic Research website. Out of all the signals, 56721 headings are equal to 511, which represents approximately 32% of all the signals.

The plot B.2 suggests that some ships emitted a much larger amount of signals than the other ships.

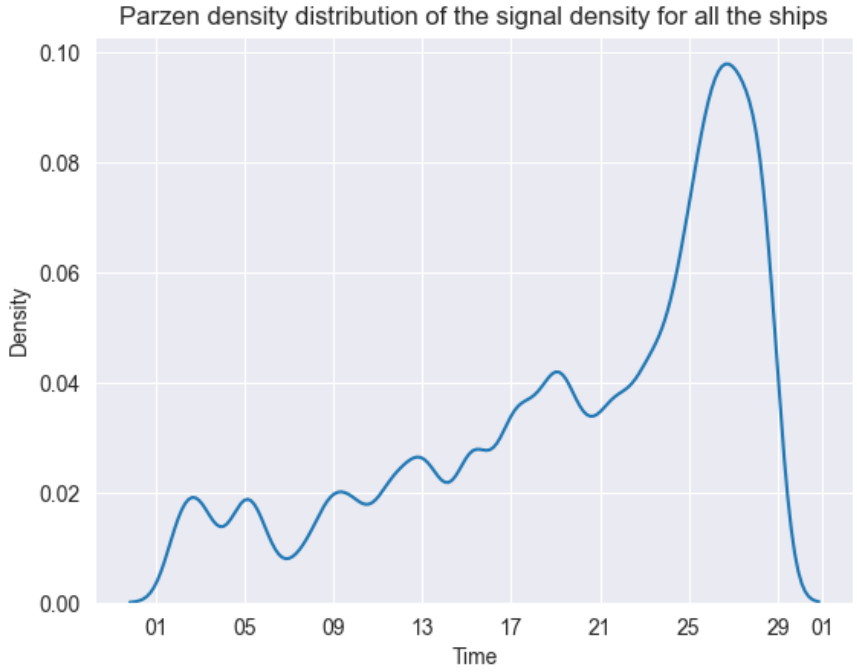


Figure B.4: The Parzen density distribution of the emitted signals over time for all the ships in the dataset.

The Figure B.4 shows that there were many more signals were sent in the period 24-28th of April compared

to the other dates.

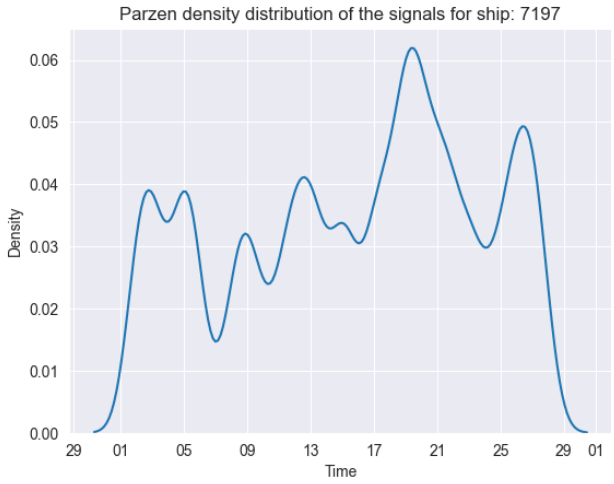


Figure B.5: Density of AIS signals over time for the ship 7197.

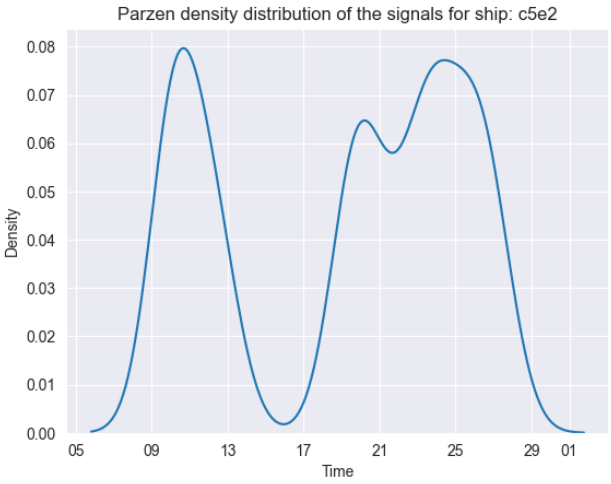


Figure B.6: Density of AIS signals over time for the ship c5e2.

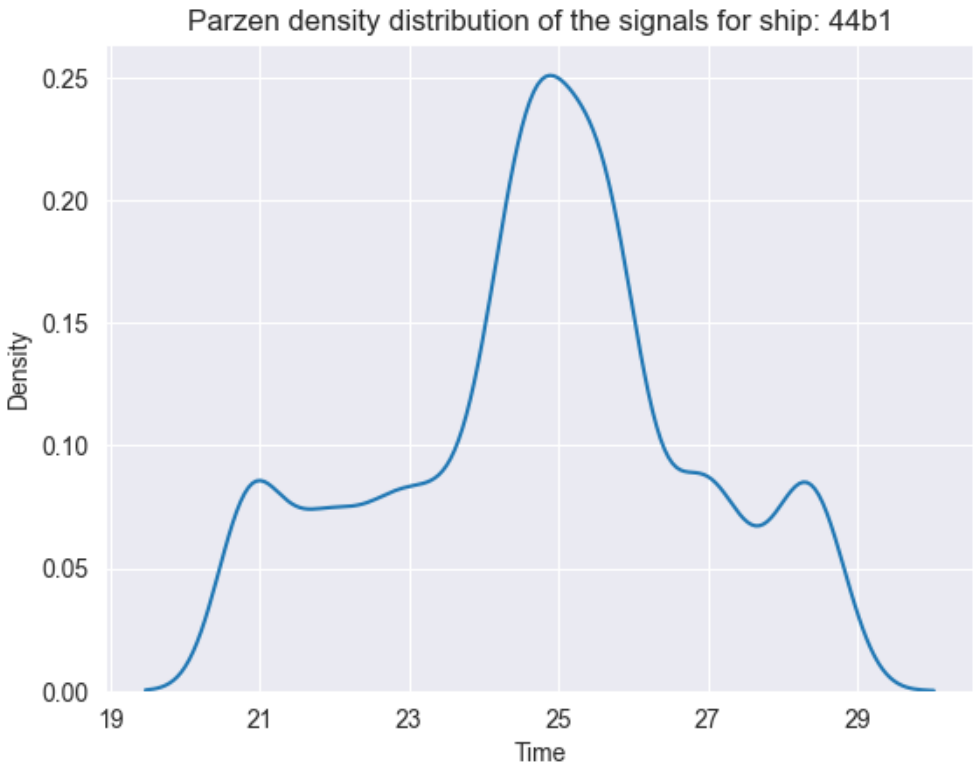


Figure B.7: Density of AIS signals over time for the ship 44b1.

As shown in B.5, B.6 and B.7 this concentration of signals during the end of April is mostly due to the following factors:

- Some ships like 7197 constantly send signals for the whole duration of the timeframe.

- Some ships like c5e2 send more signals towards the end of the timeframe.
- Some ships like 44b1 start sending signals at the end of the timeframe.

Port name	Nº Signals
VALLETTA	104734
CEUTA	15885
MARSAXLOKK	14182
GIBRALTAR	11946
NEMRUT	8298
PALMA DE MALLORCA	5670
BARCELONA	4574
YALOVA	3458
ISKENDERUN	2300

Port name	Nº Signals
VALENCIA	2027
AUGUSTA	1995
TUZLA	1663
ALEXANDRIA	1348
GENOVA	975
PORT SAID	793
DILISKELESI	729
CARTAGENA	159
DAMIETTA	107
TARRAGONA	72
PIRAEUS	19

Figure B.8: Table of reported number of signals for every port.

The corresponding ships have data incoming from 20 ports. In Figure B.8, the reported number of signals per port is presented. Unfortunately, every ship has a constant port for the whole data set, which makes it harder to differentiate between different routes and add some components in the anomaly detection pipeline focused on heuristics.

B.2 Anomaly Algorithm Design

In this section, an explanation of the derived threshold values for the heuristic components is provided.

B.2.1 Speed Component

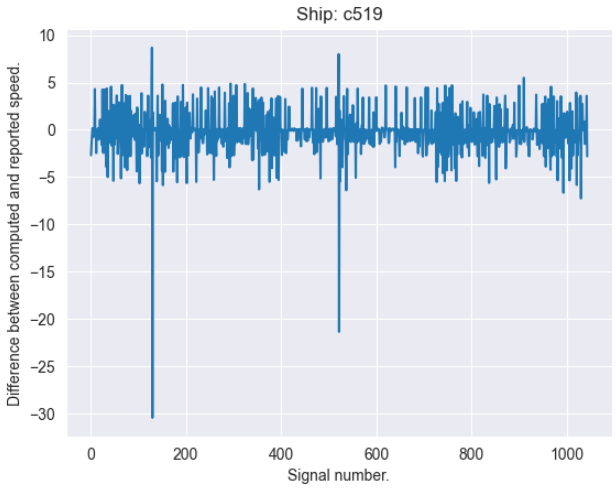


Figure B.9: Difference between computed and reported speed for ship c519 in km/h.

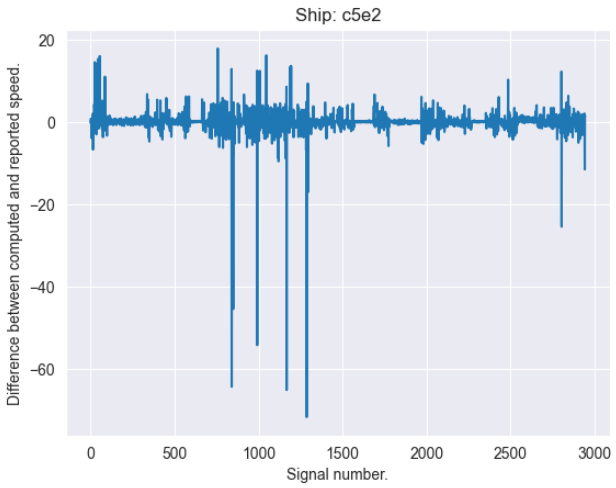


Figure B.10: Difference between computed and reported speed for ship c5e2 in km/h.

As we can see in Figures B.9, B.10 and B.11, there are some spikes which look anomalous for an absolute difference greater than 10 km/h, which is the threshold chosen for this rule.

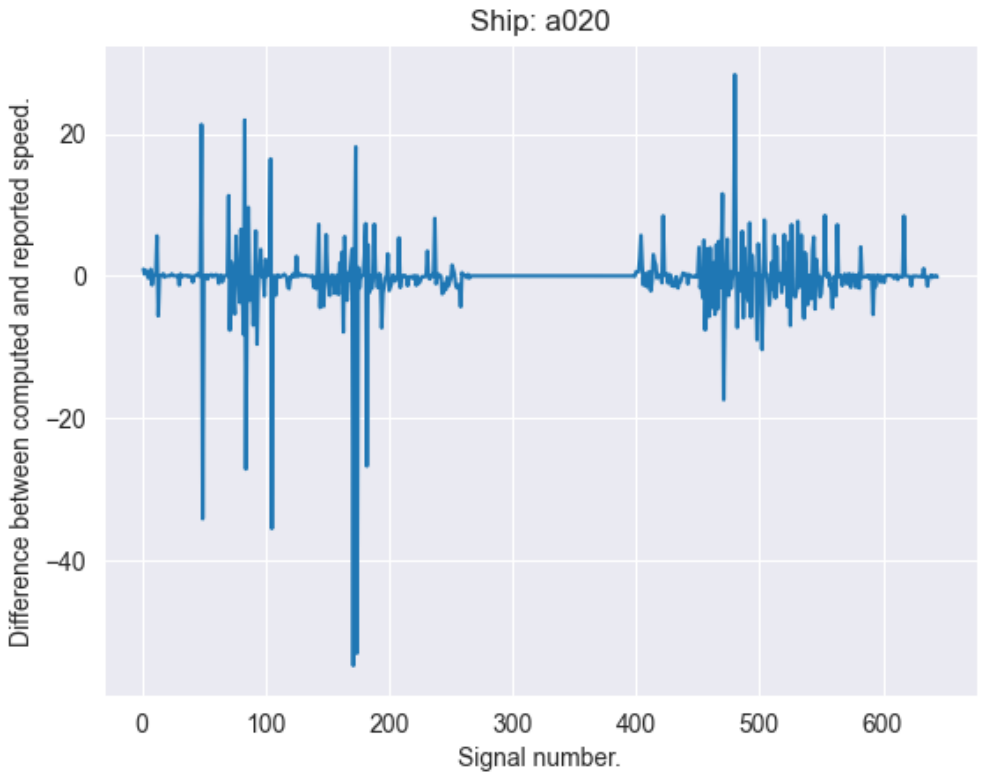


Figure B.11: Difference between computed and reported speed for ship a020 in km/h.

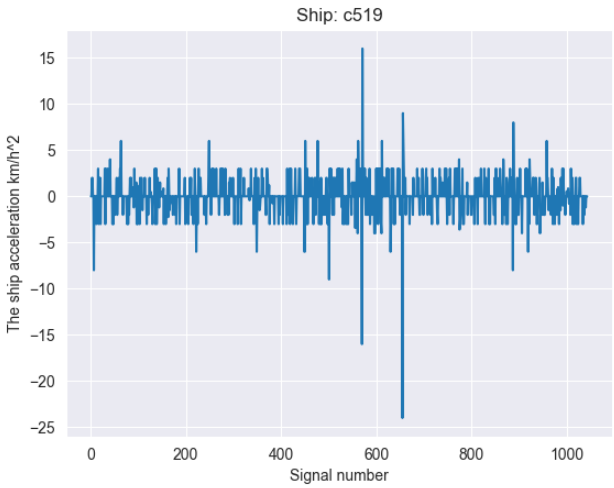


Figure B.12: Difference between computed and reported speed for ship c519 in km/h.

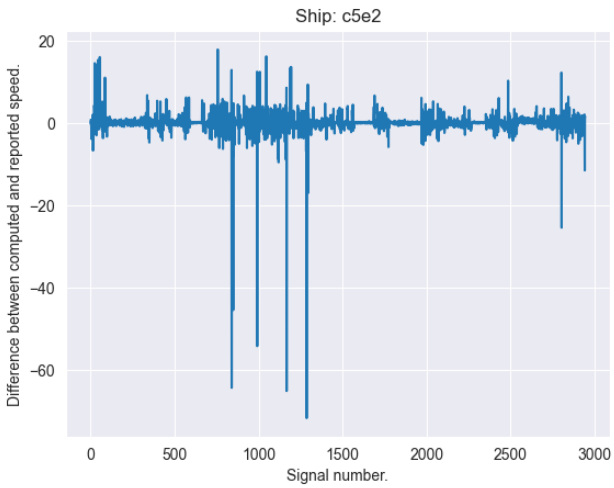


Figure B.13: Difference between computed and reported speed for ship c5e2 in km/h.

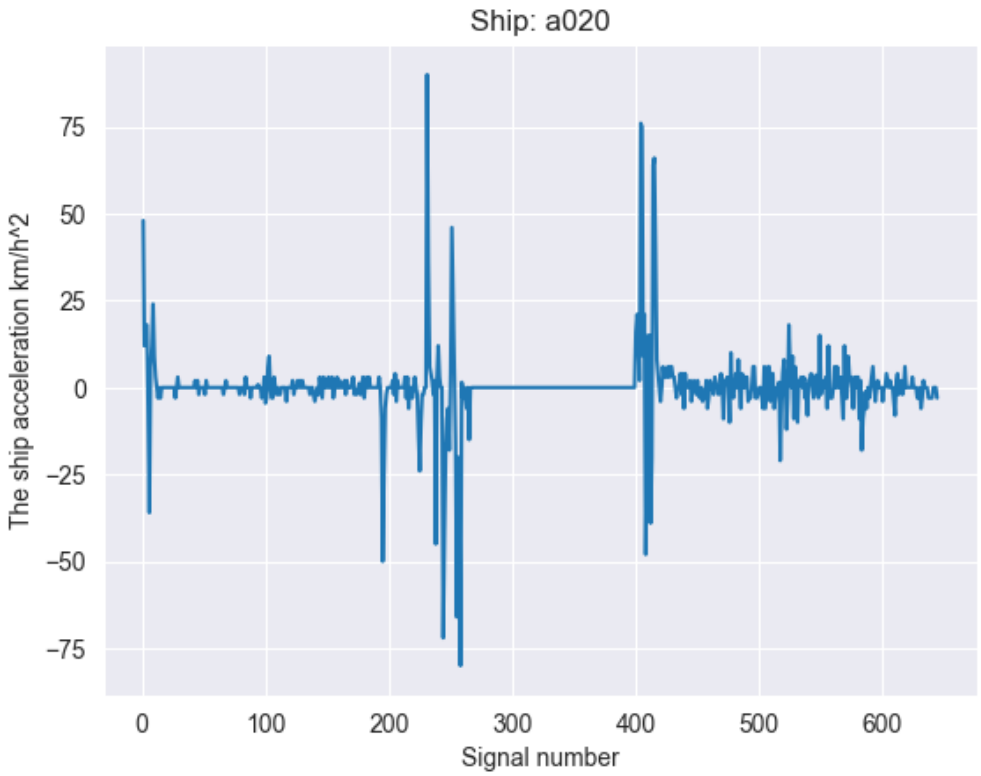


Figure B.14: Difference between computed and reported speed for ship a020 in km/h.

As we can see in Figures B.13, B.14 and B.12, there are some spikes which look anomalous for an acceleration greater than 50 km/h^2, which is the threshold chosen for this rule.

The other components are presented in detail in the team's Research repository.

B.3 Anomaly Algorithm Evaluation

This section of the appendix focuses on presenting the methodology and results of the accuracy evaluation that was conducted in order to assess the performance and reliability of the implemented anomaly detection algorithm within the application.

B.3.1 Data Loading and Data Preprocessing

For the purpose of training the Isolation Forest model, we subsampled multiple ship instances with varying values of the computed anomaly score: The IDs of the selected ships are listed below:

- $'0x336a704d597478a6d093225a40f5fd3ab79088df'$
- $'0x5172715d4ef2b15d977c4e056ddb94c741c30fdd'$
- $'0x641b4c65e679b09d89b44cf54a787751128acba9'$
- $'0xa24af55bef77a07e396ffb0f26516cf7784089a4'$
- $'0xaf60323d035145fa53b4768dbef213c04ca6075c'$
- $'0xc5890253c349d9aeeb6cee8eaa09a54aeea0dc5f'$
- $'0xf85dcd36be27e20d5b986e3e62529e04d9b11714'$

The training data was composed of multiple sequences of AIS signals, each sequence corresponding to a particular ship. Within one sequence, the AIS signals were ordered consecutively, in the ascending order of their timestamps. For the purposes of this analysis, the sequences were selected from the following time window: **WINDOW START** - 2015-04-01T00:00:00 **WINDOW END** - 2015-04-28T23:59:59. The total number of signals falling into the defined time window was 2889.

Feature Engineering

As the Isolation Forest model had to be trained on a dataset consisting of singular data points, rather than time series data, we derived new features for our data, such that their (values) encodings reflected the time dependency between the consecutive signals of each ship.

In order to introduce the temporal aspect to our data, we introduced an additional feature for each of the existing features. Each of this new feature reflected the difference between the values recorded in the "current" signal and the "previous" signal.Therefore, for the data at-hand we derived the following additional features with the corresponding values:

- **SPEED_DIFF** - Numerical difference between the speed recorded in the AIS signal and the speed value recorded in the previous AIS signal.
- **LAT_DIFF** - Numerical difference between the registered latitude coordinate and the latitude coordinate of the previous signal.
- **LONG_DIFF** - Analogous to the LAT_DIFF feature.
- **COURSE_DIFF** - Analogous to the SPEED_DIFF feature.
- **HEADING_DIFF** - Analogous to the COURSE_DIFF feature.
- **TIME_DIFF** - The difference computed in seconds between the analyzed timestamp and the timestamp of the previous AIS signal.

Feature Encoding

To adapt the data to the format necessary for training the anomaly detection model, we encoded the values of the VESSEL_HASH AND departurePortName.

B.3.2 Model Training

Model Selection

As our task is to derive an artificial benchmark for the anomaly detection model used by the application, we considered the Isolation Forest algorithm [12].

Anomaly Score Estimation Strategy

The heuristics-based anomaly detection algorithm leveraged by the developed system analyzes each incoming data point (new AIS signal) within the individual context of the respective ship. In more detail, the different feature values encapsulated by the new AIS signal are passed through a chain of heuristic-based score calculator components. At the lower level, a stateful map operation is performed with each new value of a specific feature, taking into consideration the previous value of the same feature. Based on the defined heuristics of the respective feature, the previously mentioned operation returns an increment to the general anomaly score assigned to the AIS signal that undergoes processing.

As the Isolation Forest algorithm leverages the existing patterns within the data, we devised two strategies for estimating the analogous "anomaly score" for each AIS signal:

- Train an Isolation Forest estimator on the whole preprocessed dataset, corresponding to the time-filtered data of the selected ships, such that the estimator learns the existing anomaly patterns in the context of all the signals. Such general anomaly patterns could be interpreted as general heuristics

that are applied for each new signal, irrespective of the type and individual context of a ship. The anomaly score predictions of the respective estimator will be referred to as global estimates/scores.

- Train an Isolation Forest estimator for each individual ship such that the estimator learns the existing patterns within the individual context of a vessel. The learned individual patterns could be interpreted as custom heuristics of the respective ship. The anomaly score estimates per each ship computed using this strategy will be referred to as local estimates/scores.

Training the Isolation Forest Estimator

The sklearn implementation[2] of the Isolation Forest anomaly detection model will be used. The model is trained on the preprocessed data, with the following hyperparameter values:

- n_estimators: 100
- max_samples: 'auto'
- contamination: 'auto'
- max_features: 1.0

Plot of the Detected Outliers

In order to visualize the learned decision boundary (see Figure B.15), we reduce the dimensionality of the dataset from 13 to 2 dimensions. The t-SNE[3] dimensionality reduction algorithm was leveraged for reducing the dimensionality of the training dataset.

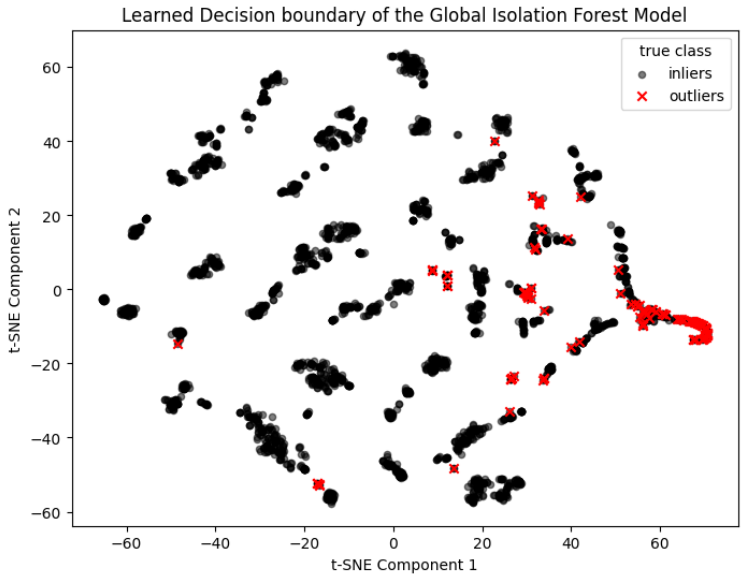


Figure B.15: Learned Decision boundary of the Global Isolation Forest Model, where the t-SNE Components 1 and 2 correspond to the features of the dimensionally-reduced dataset.

Anomaly Scores Comparison

Given the anomaly score distributions retrieved from the system and the estimated score distributions of the Isolation Forest models, we can compare the distributions corresponding to each individual ship by computing the average absolute difference between the scores corresponding to the same AIS signal.

[2] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html
[3] https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE

Therefore, by performing the above computations, we compare the anomaly-detecting capabilities of the system to those of the trained Isolation Forest estimators.

Given that:

- S_{system} represents the array of the time sorted computed anomaly scores
- S_{local} represents the array of the time sorted anomaly scores computed within the individual context of the ship
- S_{global} represents the array of the time sorted anomaly scores computed within the global context of the selected ships

Compute the average score deviation values for both the local and global estimations: $\delta_{local} = (\frac{1}{len(S_{local})}) \cdot \sum_i |S^{(i)}_{local} - S^{(i)}_{system}|$ $\delta_{global} = (\frac{1}{len(S_{global})}) * \sum_i |S^{(i)}_{global} - S^{(i)}_{system}|$

With the computed deviation values, we can roughly estimate the accuracy of the anomaly detection algorithm employed by the system as the average deviation of the scores assigned by the system and the scores estimated by the Isolation Forest models:

$\alpha_{system} = \frac{(100\% - \delta_{local}) + (100\% - \delta_{global})}{2}$

The roughly estimated accuracy of the system with respect to the derived benchmark is 80.9% which demonstrates a high correlation between the performance of the system's algorithm and the leveraged Isolation Forest algorithm in the context of detecting anomalies within the provided subsample of AIS data.

Appendix C

Comparing Streaming Frameworks

The following appendix compares different streaming frameworks that could tackle the following two problems:

- **Problem no. 1**. In a real-world scenario, multiple large streams of AIS (Automatic Identification System) signals will be sent into the application. The application must handle receiving and manipulating all of these streams in a persistent and scalable way.
- **Problem no. 2**. A stream of incoming AIS signals must be processed fast so that an anomaly score for the ship corresponding to the AIS signal can be updated in real-time.

Apache Kafka

Arguably the primary streaming framework, used by Uber, Spotify, Shopify, and thousands of other companies. This piece of software can act as a messaging queue, as a persistent log, and with the help of Kafka Streams - as a stream processing library as well. In practice, Kafka is often used to handle massive amounts of large streams incoming and outgoing from an application - i.e., it is used as a messaging queue. As mentioned, it also has support for stream processing and analytics, although the Kafka Streams library has limited capabilities.

In our case, Kafka would be the perfect solution for the first problem, since it is built for precisely that: handling a large number of streams coming into an application. Additionally, Kafka keeps a log of incoming events, therefore it could also be used as a logging and persistence component to not lose any important data, such as AIS signals received.

Furthermore, Kafka could indirectly tackle the second problem, since the Kafka Streams library is made for processing streaming data. However, there are two problems with using Kafka Streams for the processing itself: 1) the library has limited functionality (at least compared to Flink) 2) for stream processing it is considered to have higher latency than Flink [17], [18]. Therefore, we could use Kafka alone to solve the first two problems, but Kafka is optimal only for the first one.

Apache Flink

Unlike Kafka, Flink is not used as a messaging queue. Instead, the main purpose of Flink is stream processing and analytics. It allows for fast, scalable, and reliable stream processing. Additionally, Flink can consume from Kafka. This is often done in practice: large incoming data streams are handled by Kafka, and most of the processing itself is done using Flink. Additional important features of Flink include: 1) can handle out-of-order incoming events (which Kafka Streams cannot do) 2) can be used from a few languages (Java, Scala, Python) 3) Flink has Flink ML library [19] which can aid in building ML pipelines if we end up using more complex algorithms for anomaly detection 4) the wide use of Flink has resulted in it being a largely documented framework, which is helpful while developing the pipeline.

Apache Storm

A stream processing framework, similar to Flink. Just like Flink, it provides low-latency streaming and fault tolerance. Additionally, it is built to have support on almost any language [20]. On the other hand, Apache Storm has a few drawbacks: 1) it does not support stateful operations [21] which are crucial for track anomaly score calculation; 2) limited-to-none time window support [21], which is also important for anomaly score calculation.

Apache Samza

Apache Samze is another stream processing framework. In terms of functionality, it is very similar to Flink: it supports low-latency streaming, is scalable, and works well when combined with Kafka. As a drawback, according to [22], Samza "does not have built-in state management capabilities and relies on external systems like Apache Kafka" for managing the state. Finally, Samza is much less popular than Flink and therefore has less documentation and less community support.

Other frameworks

Other popular big-data frameworks such as Apache Apex or Apache Spark are a not-as-good fit for either of the two problems due to the following reasons: Apache Apex is retired; Apache Spark is mostly used for batch processing and not stream processing.

Appendix D

Developer Manual

D.1 Overview

This manual aims to provide all the necessary information for software developers to set up, edit, and maintain the code for the Track Anomaly Detection application.

As a note, all paths to the files or folders mentioned in this manual are relative to the root folder of the application's entire code base.

D.1.1 Terminology

- **Track** - the ship's path from the departure port to the destination.
- **AIS** - Automatic Identification System
- **AIS signal** - the signal that ships must frequently send. Such a signal contains information about the vessel: its position, rotation, destination, etc.
- **Anomaly score** - the score between 0% and 100% given by the application to a certain track. It tries to represent how suspiciously the ship is behaving in its track. However, note that it should not be taken as some "ground truth", and further evaluation should be done.

D.1.2 The structure of the code base

The repository consists of the following folders and files:

- `.gitlab` - Template files for creating Issues and Merge Requests in GitLab.
- `.idea` - IntelliJ IDEA project files.
- `backend` - Code for the backend part of the application.
- `config` - CheckStyle and PMD rulesets.
- `frontend` - Code for the frontend part of the application.
- `simulator` - Code for the simulator.
- `stress-tester` - Code for the stress tester.
- `.gitignore` - Files to be ignored in git repository.
- `.gitlab-ci.yml` - Gitlab CI pipeline setup.

- `README.md` - Main README file. Instructions on how to set up the project can be found there.
- `README.md` - Main README file. Instructions on how to set up the project can be found there.

D.1.3 Setting up and running the application

Instructions are in the README files of each part of the application (backend, frontend, simulator, and even stress tester). The main README file is at the top level of the repository: `README.md`.

D.2 Technology Stack and Architecture

D.2.1 Technology Stack

- **Apache Kafka** for distributed event handling with its message queue. The events coming to the queue are ships' AIS signals.
- **Apache Flink** for distributed stream processing. It is used to calculate the anomaly scores for the tracks.
- **Apache Druid** as the database for storing a large amount of data coming in streams.
- **Java Spring Boot** for the web server.
- **React.js** for the frontend.

D.2.2 System Design

This section describes the architecture of the system. The more detailed explanation of each part of the system can be found in the following sections of this Developer Manual.

The mentioned technologies were connected to a coherent functional system. The design of it is presented in Figure D.1. It can be split into the following parts:

- Simulator (Java)
- Event Messaging Queue (Kafka)
- Stream Processing and Anomaly Score Calculation (Flink)
- Web Server (Spring Boot)
- Database (Druid)
- Frontend (React.js)

The Simulator is the first part of the system. It is an application that simulates events that mimic historic AIS signals. Those signals are sent by ships in periodic intervals to the Kafka Messaging Queue. As the deployed system will use real-time world AIS data, the Simulator will not be run in the deployed system. Thus, it is a separate process not part of the web server.

Then, the Kafka Messaging Queue takes incoming produced data and stores it in topics (queues). Even though the only producer is the Simulator, Kafka Queue could handle more if needed. For instance, multiple military radars sending information from the ships could be added as separate producers to the Kafka Queue. Since Kafka needs to be started as a separate process, it is detached from the web server.

The next step for the data flow is the Flink stream processing. This part of the code is a consumer of the Kafka Queue, meaning it takes a stream of events from it. The taken stream is split according to the vessel ID, and then each ship's anomaly score is calculated. A past state is used to calculate the anomaly score for a new data entry, which is provided by the Flink stateful map functions. In addition, during the processing of AIS signals, notification data is calculated, and the current ship details objects are aggregated. The new details are written to the Kafka topic, and also saved to the Druid database.

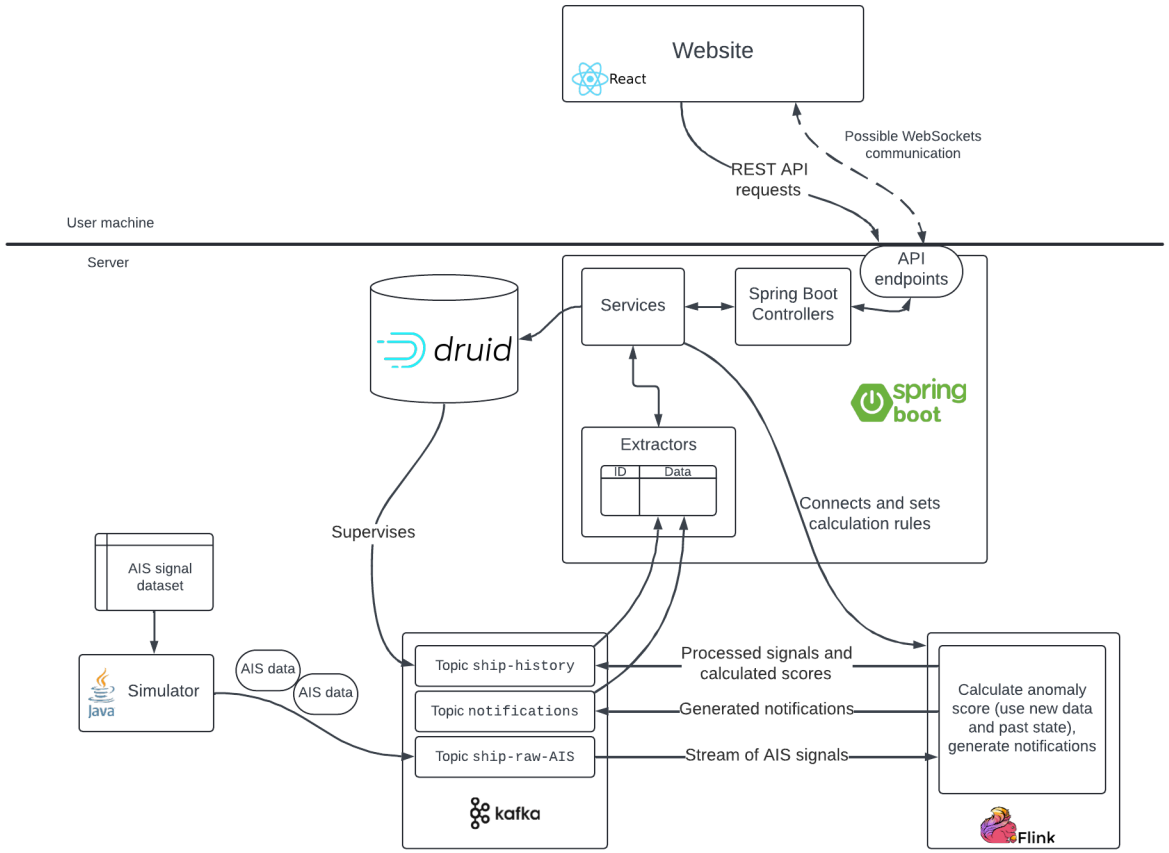


Figure D.1: *System Design Layout for the whole Track Anomaly Detection application. Boxes represent separate steps and components in the system, while the arrows represent the data flow.*

The communication between the client and the server is done through a REST API and WebSockets requests. To handle these, a web server was implemented using the Spring Boot framework and structured according to the layered architecture: controllers, services, and core domain code. The controllers handle the API endpoints. When a request is received, the controller calls service classes, which call the internal classes to handle the logic of the request.

The Apache Druid database is used for storing ships signals and score histories. The database is set to supervise the Kafka topic to which the calculated anomaly scores along with the corresponding AIS signals are sent. This way, web servers are able to query the database when the ship historical data is needed. For example, such a functionality is useful when services need to retrieve the ship's anomaly scores throughout the track, so that the colourful trajectory can be displayed in the frontend.

The frontend is the final part of the design and is the only visible part to the system user. Its implementation relies on the React.js framework. As a simplification, in Figure D.1, the backend part of the frontend is skipped. However, when deploying, the React server should be started on the backend, so that the client computer can retrieve the resources, such as HTML pages, CSS, and JavaScript code.

D.3 Backend

This section describes the backend in detail. It consists of the following parts: event handling with Kafka (D.3.1), stream processing with Flink (D.3.2), and a Spring Boot web server (D.3.3).

D.3.1 Kafka Topics

There are three Kafka topics (queues) used:

- `ships-raw-AIS`
- `notifications`
- `ships-history`

The events pushed to these topics are Strings, following the JSON format. These JSONs represent the required objects from the folders `backend/src/main/java/sp/model` and `backend/src/main/java/sp/dtos/ExternalAISSignal.java`. The examples of the Strings are below, and the general serialization logic is implemented in `backend/src/main/java/sp/pipeline/utils/json/SoftJsonSerializationSchema.java`.

Topic `ship-raw-AIS`

Ship AIS signals should be sent to this topic. Anomaly score calculation (done using Flink stream processing) consumes the ship data from this topic.

The events that come to this topic should be Strings representing JSON serialized version of the object `ExternalAISSignal` (described in `backend/src/main/java/sp/dtos/ExternalAISSignal.java`). Here is an example of such a String:

```
{
    "producerID": "simulator",
    "shipHash": "0x6d6794f3186f584637721a1e1789fd2e71c28195",
    "speed": 0.2,
    "longitude": -5.354052,
    "latitude": 35.92466,
    "course": 314.0,
    "heading": 52.0,
    "timestamp": "2015-04-01T20:29:00Z",
    "departurePort": "CEUTA"
}
```

Topic `notifications`

Notifications are meant for the user (the Navy operator) to get attention to the ships that became anomalous. After calculating the ship's anomaly score, some notifications are created in the pipeline. After being created, notifications are saved to this Kafka topic, from which the extractor frequently polls the notifications so that the frontend can get them (through the API endpoint via Spring Boot controller and service).

Refer to the representative sections to get more information about the pipeline flow (Section D.3.2), extractors (Section D.3.2), and web server structure (Section D.3.3).

The events that come to this topic should be Strings representing JSON serialized version of the object `Notification` (described in `backend/src/main/java/sp/model/Notification.java`). Here is an example of such a String:

```
{
    "id": 210698893,
    "shipID": 990836311,
    "currentShipDetails": {
        "currentAnomalyInformation": {
            "score": 50.0,
            "explanation": "Heading difference between two consecutive signals is too large:
     63 degrees is more than threshold of 40 degrees.\nManeuvering is too frequent: 11
    strong turns (turns of more than 40 degrees) during the last 60 minutes is more than
    threshold of 10 turns.\n",
```

```
            "correspondingTimestamp": "2015-04-01T21:59:00Z",
            "id": 990836311
        },
        "currentAISSignal": {
            "id": 990836311,
            "speed": 0.5,
            "longitude": 14.54607,
            "latitude": 35.82201,
            "course": 153.0,
            "heading": 192.0,
            "timestamp": "2015-04-01T21:59:00Z",
            "departurePort": "VALLETTA",
            "receivedTime": "2024-06-14T18:45:29.028162548+02:00"
        },
        "maxAnomalyScoreInfo": {
            "maxAnomalyScore": 50.0,
            "correspondingTimestamp": "2015-04-01T21:59:00Z"
        }
    },
    "read": false
}
```


Topic `ship-history`

This topic saves the ship details with their calculated anomaly scores. Similarly, as for `notifications`, the extractor frequently polls this topic so that the frontend can get the current ship details.

Note that usually firstly only the ship details without the anomaly score will be saved to this topic, and only after the score is calculated updated details are pushed to this topic.

The database (configured using Apache Druid) is linked to this topic and will permanently save the events from here. This also allows for implementing features, such as showing the history of anomaly scores for the ships.

Refer to the representative sections to get more information about the pipeline flow (Section D.3.2), extractors (Section D.3.2), and web server structure (Section D.3.3).

The events that come to this topic should be Strings representing JSON serialized version of the object `Notification` (described in `backend/src/main/java/sp/model/Notification.java`). Here is an example of such a String:


```
{
    "currentAnomalyInformation": {
        "score": 0.0,
        "explanation": "some explanation",
        "correspondingTimestamp": "2015-04-01T20:27:00Z",
        "id": 990836311
    },
    "currentAISSignal": {
        "id": 990836311,
        "speed": 5.9,
        "longitude": 14.54465,
        "latitude": 35.82241,
        "course": 109.0,
        "heading": 113.0,
        "timestamp": "2015-04-01T20:27:00Z",
        "departurePort": "VALLETTA",
        "receivedTime": "2024-06-14T18:41:43.028537624+02:00"
    },
    "maxAnomalyScoreInfo": {
        "maxAnomalyScore": 0.0,
        "correspondingTimestamp": "2015-04-01T20:27:00Z"
    }
}
```

D.3.2 AIS Signal Processing With Flink

The AIS signals that come to the Kafka topic `ship-raw-AIS` (all Kafka topics are described in Section D.3.1) are processed, the anomaly scores are calculated, and, whenever needed, the notifications are created. The updated ship details are passed back to the Kafka topic `ships-history`, and the notifications are passed back to the Kafka topic `notifications`. All these steps are done using an "Anomaly detection pipeline" in the backend. Its implementation can be found inside `backend/src/main/java/sp/pipeline/AnomalyDetectionPipeline.java`.

The steps for the pipeline's flow are described below. These steps are constructed when the method `buildPipeline()` (in the class `backend/src/main/java/sp/pipeline/AnomalyDetectionPipeline.java`) is called.

For the API endpoints to work, there are "extractors" implemented that frequently poll the Kafka topics `ship-raw-AIS` and `ships-history`. More information about them can be found in Section D.3.2.

Pipeline Part: ID Assignment

The first part of the pipeline is calculating and assigning internal IDs to the ships. This part of the pipeline is built in `backend/src/main/java/sp/pipeline/parts/identification/IdAssignmentBuilder.java`.

To be more precise, the ID Assignment step first takes the stream of data from the Kafka topic `ship-raw-AIS`. This stream contains AIS signal data. An internal ID is calculated for each of the stream signals. The ID calculation details can be found inside `buildIdAssignmentPart` method in the mentioned class.

Pipeline Part: Anomaly Score Calculation

The stream of the AIS signals with IDs calculated goes through the anomaly score calculation part. For this, an interface is created: `backend/src/main/java/sp/pipeline/parts/scoring/scorecalculators/ScoreCalculationStrategy.java`. The interface allows for easy changes in the score calculation strategy so that developers can create or edit new ones easily.

The score calculator strategy that is currently used is implemented in `backend/src/main/java/sp/pipeline/parts/scoring/scorecalculators/SimpleScoreCalculator.java`. Its idea is to split the stream of all ships into a keyed stream (keyed by the ship's ID) and then for each ship to calculate an anomaly score by combining four heuristic checks.

The four heuristic checks used are described in the folder `backend/src/main/java/sp/pipeline/parts/scoring/scorecalculators/components/heuristic`. They implement the abstract class `HeuristicStatefulMapFunction` (in the mentioned folder). This class is a child class of Flink's `RichMapFunction`. This allows to use Flink's states (such as, `ValueState` and `ListState`[1].). The setting up of the heuristic calculation is done in the method `open`, and the anomaly score calculation logic is implemented inside the `map` function (`@Override public AnomalyInformation map(AISSignal value)`).

To see the example of how one specific heuristic score calculation works, you can check the `SignalStatefulMapFunction` class. It extends the abstract class `HeuristicStatefulMapFunction`, and implements a method for checking whether a new AIS signal is considered an anomaly, compared to the past one. It considers the signal an anomaly if the ship did not send the signal for a long time and, during that time, travelled quite some distance. The exact thresholds can be changed inside that class.

[1]Check the Flink documentation at `https://nightlies.apache.org/flink/flink-docs-release-1.19/docs/dev/datastream/fault-tolerance/state/` and `https://nightlies.apache.org/flink/flink-docs-release-1.19/docs/concepts/stateful-stream-processing/`.

How to change the threshold of any of the heuristic checks?
To change the threshold, you must go to the class that checks if a signal is an anomaly based on your desired heuristic.
For example, if you want to change the speed threshold, you have to go to the class `SpeedStatefulMapFunction` (`backend/src/main/java/sp/pipeline/parts/scoring/scorecalculators/components/heuristic/SpeedStatefulMapFunction.java`). At the top of this class, you can find `private static final` variables used for thresholds. One of them is `SPEED_THRESHOLD`.
Similar steps can be performed to change any other threshold.

How to use the past state in the score calculation?
Flink's Rich Map Functions provide availability to use states, such as Value State and List State. The example usage of them can be found in classes `HeuristicStatefulMapFunction` (has the variable `private transient ValueState<AISSignal> aisSignalValueState`) and `ManeuveringStatefulMapFunction` (has the variable `private transient ListState<AISSignal> previousSignalsListState`). You can check how they are used there.
In essence, the steps to use the State are the following (these can also be found in the mentioned example files):

- You need to have the `private transient` state variable in the class.
- In the `open` method, initialize the state for that variable with the new state descriptor for the current Flink environment. You can reuse `getValueState` and `getListState` methods from `HeuristicStatefulMapFunction` class.
- In the `map`, you can get and update the values of the state. The methods are described in Flink's documentation: `https://nightlies.apache.org/flink/flink-docs-release-1.19/docs/dev/datastream/fault-tolerance/state/#using-keyed-state`.

Note that you should be careful when using List State as it can have a heavy load on the memory.

How to add a new heuristic?
As described at the start of this section, the currently used anomaly score calculator is described in `backend/src/main/java/sp/pipeline/parts/scoring/scorecalculators/SimpleScoreCalculator.java`. In its method `setupFlinkAnomalyScoreCalculationPart`, you can see that the four heuristics are used. You can also check the implementation of those heuristic classes to see how to implement one yourself.
To add a new heuristic, you should add a class that extends the abstract class `HeuristicStatefulMapFunction` and override the abstract methods. Then, add this new heuristic class (that you have created) to `SimpleScoreCalculator`, similarly to how other heuristics are added. Note that you should change the `getAnomalyScore` method in the currently existing heuristics so that the sum of the anomaly scores among the heuristics adds up to 100. Another way would be to modify the score calculator to always normalize this score to 100.

How to add a new anomaly score calculation strategy?
The currently used anomaly score calculation strategy is `backend/src/main/java/sp/pipeline/parts/scoring/scorecalculators/SimpleScoreCalculator.java`. Similarly, you can create a new one. It must implement the interface `backend/src/main/java/sp/pipeline/parts/scoring/scorecalculators/ScoreCalculationStrategy.java`. You must override the method `setupFlinkAnomalyScoreCalculationPart` where you implement the logic of how a stream of AIS signals is turned into a stream with objects with calculated scores.
Once you have multiple score calculators, you should specify which one is used in the constructor of the `ScoreCalculationBuilder` in the `Qualifier` annotation (`backend/src/main/java/sp/pipeline/parts/scoring/ScoreCalculationBuilder.java`).

Pipeline Part: Score Aggregation

The stream of AIS signals with IDs assigned and the stream of the calculated scores (these two streams are computed in the pipeline parts described above) are then aggregated to the instances of class `backend/src/main/java/sp/model/CurrentShipDetails.java`. These objects are serialized and sent to the Kafka topic `ships-history`.

When setting up the Apache Druid database, the supervisor for the Kafka topic `ships-history` should be created[2]. This allows the database to save the events sent to this topic.

The critical detail is that two aggregated signals are generated for each AIS signal. At first, the event representing the AIS signal with calculated ID is sent to the mentioned Kafka topic. Later, once the anomaly score is calculated, it is combined with the corresponding AIS signal and the updated details are sent to the topic again. All of this is set up in `backend/src/main/java/sp/pipeline/parts/aggregation/ScoreAggregationBuilder.java`, and the aggregation logic is implemented in `backend/src/main/java/sp/pipeline/parts/aggregation/aggregators/CurrentStateAggregator.java`. Note that the order of these two events can be flipped in some rare cases (due to distributed and parallel computing): the aggregation will first get the calculated score and only later the AIS signal. However, aggregator logic is implemented so that the order of these events does not matter.

Pipeline Part: Notifications

The last part in the pipeline is the generation of the notifications. The notification detection takes a stream of aggregated ship details (described above), and the stream of notifications is generated using the notifications aggregator. These notifications are sent to the Kafka topic `notifications`. The notifications extractor frequently polls from this topic, so the API endpoints work correctly.

The notifications creation (stream aggregation) logic is implemented in `backend/src/main/java/sp/pipeline/parts/notifications/NotificationsAggregator.java`.

Extractors

The extractors are classes that poll from a specified Kafka topic with a specified polling frequency. An abstract class for an extractor is described in the file `backend/src/main/java/sp/pipeline/parts/aggregation/extractors/GenericKafkaExtractor.java`. The classes that inherit this class must override the method `processNewRecord`, which specifies how a consumer record from a Kafka topic should be processed.

[2]The steps to do that are specified in `backend/README.md`.

These extractors keep the required objects ready to be asked by the service classes. For example, the `ShipsDataService` (described in `backend/src/main/java/sp/services/ShipsDataService.java`) calls the `ShipInformationExtractor` (`backend/src/main/java/sp/pipeline/parts/aggregation/extractors/ShipInformationExtractor.java`) to get the current ship information. Ship Information Extractor can provide the current ship information stored inside the hashmap, which is updated by processing new recors. `NotificationsExtractor` works analogously.

D.3.3 Spring Boot Web Server

The web server handles the logic for the API endpoints. The full description of the endpoints can be found in the OpenAPI specification (see Appendix E).

The web server is implemented using a layered architecture. The controllers are described in `backend/src/main/java/sp/controllers`. They have the Spring Boot's annotation `RestController`, and the Spring Boot framework ensures that the required methods are called in the controller classes. Controllers call service classes to be able to handle the request. Service classes are described in `backend/src/main/java/sp/services`. If services need to get ship or notification data (which are coming to Kafka topics, see D.3.1), then they call the extractor classes (see D.3.2).

D.4 Frontend

The general structure of the React components and other frontend code is described in Section D.4.1. The way the map is rendered and ship clustering works is described in Section D.4.2. Some frontend values can be changed in the configuration files, which is explained in Section D.4.3.

D.4.1 Structure of the frontend code

React Components

The components implemented and their hierarchy are as follows:

```
App
|__ LMap
|__ Side
    |__ InformationContainer
    |   |__ AnomalyList
    |   |__ ObjectDetails
    |   |__ NotificationList
    |   |__ Settings
    |   |__ ErrorList
    |
    |__ Sidebar
```

These React components are implemented in the file `frontend/src/App.tsx` and the folder `frontend/src/components`.

A view of the website showing some of the mentioned components can be seen in the Figure D.2.

Some of the mentioned components (`Side` and `LMap`) were implemented as `forwardRef`. This allows for reference of some parts or methods from the upper components, which would otherwise not be possible. Such a design choice was made to reduce React rerendering as much as possible. When a variable that is a state[3] in some component is updated, the whole component (and its children) is rerendered. Because of that, variables like `pageChanger` were moved to the component `Side`. However, they are still needed for ship

[3]Check `useState` hook at `https://react.dev/reference/react/useState`.

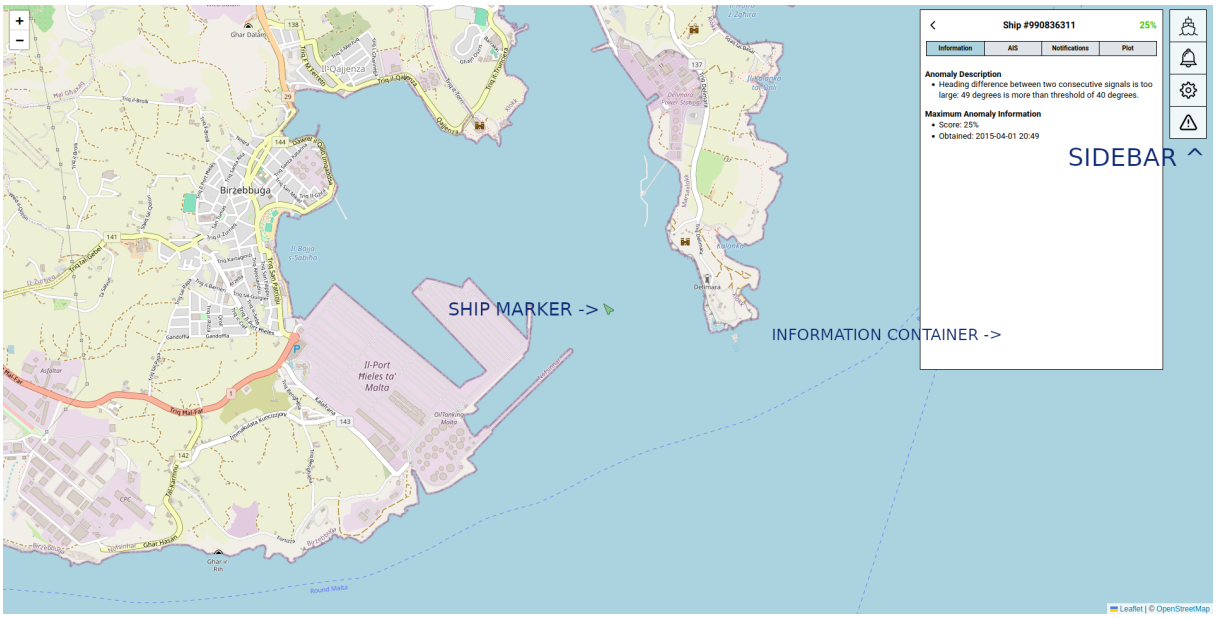


Figure D.2: *View of the website showing the mentioned components: ship marker, information container and sidebar.*

markers (created by the `LMap` component). Using `forwardRef` allows reaching them without rendering the map every time the state changes.

Other files

Apart from the React components, the other files are:

- `assets` - the folder where all the icons are put.
- `configs` - the folder with main configuration values.
- `model` - the folder with the model classes representing the objects retrieved by HTTP requests from the backend.
- `services` - the folder for the service classes.
- `styles` - the folder with CSS styling files.
- `templates` - the folder with some template interfaces used as intermediate objects for the values retrieved by HTTP requests.
- `utils` - the folder with helper methods.

D.4.2 Map

Map component and all of the related code can be found in the folder `frontend/src/components/Map`. The map is implemented using `Leaflet` library[4], and the clustering of the ship markers is done with the help of the `Leaflet.markercluster` library[5].

[4] `https://leafletjs.com/`
[5] `https://github.com/Leaflet/Leaflet.markercluster`

Map Rendering

To initialize the map only once (and not initialize it again on every rerendering), the map is saved as a reference[6], and only initialized once using with `useEffect` hook[7]. The map is initialized by the `initializeMap` method inside `frontend/src/components/Map/LMap.tsx`.

How to change map properties?
Possible options for the Leaflet Map (such as max bounds, zoom level, etc.) are described in its documentation: `https://leafletjs.com/reference.html#map`.
Any of these options can be used during the initialization of the map, which is done in the method `initializeMap` method inside `frontend/src/components/Map/LMap.tsx`.

How to change the map used?
Currently, the tile layer is using OpenStreetMaps. It is created in the method `getTileLayer` inside `frontend/src/components/Map/LMap.tsx`.
However, Leaflet provides more possibilities for the tile layers via plugins. They can be found on the website and used in the mentioned method. Check the official Leaflet website: `https://leafletjs.com/plugins.html#basemap-providers`.

Ship Markers Rendering

Every time the state `ships` is changed (for example, due to frequent polling in the backend), the ship markers on the map are rerendered. It is done by the method `updateMarkersForShips` inside `frontend/src/components/Map/ShipMarkerCluster.tsx`.

To update the markers on the map, the old ones are first cleared, and then the new ones are created and added as a batch to the Marker Cluster Group. After adding them to the marker cluster, the `Leaflet.markercluster` library figures out how to render them and groups some markers into clusters so that the screen view is not cluttered. Clusters show the maximum anomaly score among the ships grouped into it (implemented inside the method `getClusterIcon`).

To speed up the rendering, the `doFilteringBeforeDisplaying` is set to `true` in the configuration files (it is set to `true` by default). This forces to filter the ships which are not on the screen currently so that only the visible markers are added. This also shows only the most anomalous ships. Without such filtering, rendering the map with all the ships becomes too slow once the number of ships is too large (it also depends on the computer the website is running on).

Why filtering was implemented?

As mentioned above, if the value `doFilteringBeforeDisplaying` is set to `true`, the ships are filtered before displaying them on the screen. This is done to tackle the two major bottlenecks of the rendering. These bottlenecks arrise from the fact that the markers and clusters on the screen need to be frequently updated (old ones deleted, and new ones added). If the ship positions would be static for a long period of time, this would not be a problem since when the markers are added and clusters calculated, then the UI works smoothly.

The first mentioned bottleneck is the rendering of the markers. Leaflet becomes slow and freezes the website if it needs to load a lot of markers. For example, that means that it cannot handle adding 10 000

[6]Check `useRef` hook: `https://react.dev/reference/react/useRef`
[7]`https://react.dev/reference/react/useEffect`

ship markers every second. Grouping the ships into the cluster makes this a bit better. However, the way the `Leaflet.markercluster` library is implemented still forces to add a lot of markers, and that also means there is a huge computation overhead. This is because calculating the clusters is the second bottleneck. Recalculating the clusters have to be done frequently, otherwise they do not represent the accurate information.

To solve these problems, the filtering was implemented. When filtered, only the 50 (or any other number selected in the configuration files) most anomalous ships that could be seen on the current screen are displayed (and clustered). This way, the user still sees many ships, and the UI remains responsive. If the users wants to see more ships, they can zoom in to the wanted region.

As a note, if there are not too many ships, and the UI can handle this (all of this vary on different computers), then the filtering can be turned off, so that all the ships are always rendered.

D.4.3 Configuration for the frontend

The configuration files are placed in the folder `frontend/src/configs`.

File `errorListConfig.json`

The default values are:

```
{
  "savedNotificationsLimit": 1000,
  "shownNotificationsLimit": 100
}
```

Explanation of the values:

- `savedNotificationsLimit` - how many error notifications at most are saved in the `ErrorNotificationsService`.
- `shownNotificationsLimit` - how many errors at most are shown in the error notification list that can be accessed on the website through the Sidebar.

File `generalConfig.json`

The default values are:

```
{
  "notificationsRefreshMs": 1000,
  "shipsRefreshMs": 2000,
  "anomalyListMaxEntries": 250,
  "notificationListMaxEntries": 250
}
```

Explanation of the values:

- `notificationsRefreshMs` - how frequently the notifications (about anomalous ships) are polled from the backend. This value is in milliseconds.
- `shipsRefreshMs` - how frequently the ship data is polled from the backend. This value is in milliseconds.
- `anomalyListMaxEntries` - how many entries at most are shown in the anomaly list (that can be accessed through the Sidebar). This is used so that the rendering does not slow down too much.
- `notificationListMaxEntries` - how many entries at most are shown in the notifications (about anomalous ships) list. This is used so that the rendering does not slow down too much.

File `mapConfig.json`

The default values are:

```
{
    "centeringShipZoomLevel": 15,

    "doFilteringBeforeDisplaying": true,
    "maxShipsOnScreen": 50,

    "clusterChunkedLoading": true,
    "clusterChunkInterval": 200,
    "clusterChunkDelay": 100,
    "clusterMaxRadius": 80
}
```

Explanation of the values:

- `centeringShipZoomLevel` - the zoom level used when the ship is centred (triggered when clicking on the ship in the list of anomalies or notifications).
- `doFilteringBeforeDisplaying` - whether to do filtering of the ships before rendering them. Ship rendering is described in Section D.4.2.
- `maxShipsOnScreen` - the number of ships shown on the screen. All ships are shown if it is set to `-1`. It only takes effect if the value `doFilteringBeforeDisplaying` is set to `true`.
- The last four values (`clusterChunkedLoading`, `clusterChunkInterval`, `clusterChunkDelay`, and `clusterMaxRadius`) are used to set up the marker cluster. To see their effect, refer to the official `Leaflet.markercluster` documentation[8].

D.5 Simulator

The simulator is an application that takes the given dataset of AIS signals and starts simulating them from a specified timestamp until some specified end time and with a specified speed. When started, the simulator reads the given dataset, sorts the signals based on their time, and sends them one by one to the Kafka topic `ship-raw-AIS` (see D.3.1).

How to customize the simulator?
Everything is explained in the file `simulator/README.md`. In short, you can change the constants in the method `main` in `simulator/src /main/java/Main.java` before running the simulator.

D.6 Stress Tester (Scalability Testing)

Stress tester is meant to test how well the application can handle higher data loads.

This tester works similarly to the simulator (see Section D.5). The only difference is that it does not take a dataset but instead generates random AIS signals. The randomly generated AIS signals are sent to the Kafka topic `ship-raw-AIS` (see Section D.3.1).

The instructions on how to run it can be found in `stress-tester/README.md`. The main value to customize is `signalsPerSecond` in `stress-tester/src/main/java/Main.java`.

[8] `https://github.com/Leaflet/Leaflet.markercluster?tab=readme-ov-file#chunked-addlayers-options`

D.7 Distributing the Application

The backend of the application contains three large stream-related components that can all be distributed: Kafka, Flink and Druid. With the final version of the application, the main bottleneck concerning the throughput is located in the anomaly score calculation part, which is done entirely in Flink. Therefore, a considerable effort was made during the development process to ensure this bottleneck can be made larger by distributing Flink. Therefore, we describe how to distribute the application with a clear focus on distributing Flink.

By default, when the application starts, an embedded internal Flink cluster is spawned entirely inside of the web server. That is different from what happens with Kafka and Druid, which are fully external services that run independently of our application. The fact that Kafka and Druid are independent from our application allows scaling Kafka and Druid very easily - there are tens of tutorials online on how to distribute these systems, and that can be done separately from our application. The only things that might have to be changed when doing that are the connection, IP addresses, ports, and connection URLs (all in the 'kafka-connection.properties' file).

Fortunately, we have implemented our application, making sure that distributing Flink is also easy. In order to distribute Flink, 3 general steps have to be made:

1. Flink binaries should be downloaded from the Flink repository.
2. An external Flink cluster should be started according to some specifications (we provide a custom specification that we use later).
3. A few configuration properties should be changed in the main application.

We will discuss all of these steps in greater detail next.

Downloading Flink Binaries

Unlike what is happening in the application by default, where Flink binaries are downloaded and managed fully inside of the application using Gradle, for the distributed settings, a separate Flink download has to be performed. In particular, we have used Flink 1.19.0 binaries[9]. Just like Kafka and Druid, it is enough to have Java to be able to work with them. In case of Windows, you should also use WSL.

Additionally, note that you have to download these binaries to every computer/node that will be part of the cluster. Each node will have to run some Flink application, so they all need Flink binaries.

Starting External Flink Cluster

A Flink cluster consists of two main types of components. In our fairly simplified case, that would be a single Job Manager and multiple Task Managers. In essence, a Job Manager is an application that controls Task Managers. A Flink job (in our case, a Flink job is our whole score calculation pipeline) is submitted directly to the Job Manager, which then decides how to distribute this job among Task Managers.

To put it simply, there will be one computer running a Job Manager and multiple interconnected computers that each run a Task Manager, all connected to the same network. The computer that runs the Job Manager can also run a Task Manager. So starting a Flink cluster is as simple as starting a Job manager at a single computer and starting multiple Task Managers in other connected computers.

Before starting the Job and the Task Managers, you first have to write a configuration file for every machine in the cluster. The key settings to change are:

- `jobmanager.bind-host` - we changed it to `0.0.0.0`. It makes sure the Job Manager accepts requests from all over the network.
- `taskmanager.bind-host` - we changed it to `0.0.0.0`.

[9] `https://www.apache.org/dyn/closer.lua/flink/flink-1.19.0/flink-1.19.0-bin-scala_2.12.tgz`

- `rest.bind-address` - we changed it to `0.0.0.0` to ensure the Flink UI panel is accessible from all of the network.
- `jobmanager.rpc.address` - change it to the IP of the computer running the Job Manager. You can find this IP by running `ifconfig` on the terminal (or `ipconfig` if you are using Windows). In case you are using Windows this, (and all other similar) IP should be of the main Windows computer, not of the WSL container.
- `rest.address` - change it to the IP of the computer that is running the Job Manager

These were the key settings required for networking purposes. We have additionally experimented with other configurations a lot in order to achieve better results (and make them stable, since the default Flink settings resulted in a lot of errors being generated due to occasional lack of resources). Therefore, we suggest using the following configurations:

```
cluster.io-pool.size: 4
parallelism.default: 18
pekko.ask.timeout: 5 min
taskmanager.memory.framework.heap.size: 128 mb
taskmanager.memory.framework.off-heap.size: 128 mb
taskmanager.memory.jvm-metaspace.size: 256 mb
taskmanager.memory.jvm-overhead.max: 1 gb
taskmanager.memory.jvm-overhead.min: 1 gb
taskmanager.memory.managed.size: 128 mb
taskmanager.memory.network.max: 300 mb
taskmanager.memory.network.min: 300 mb
taskmanager.memory.task.heap.size: 1011627776 bytes
taskmanager.memory.task.off-heap.size: 1099511627 bytes
taskmanager.network.sort-shuffle.min-buffers: 16
taskmanager.numberOfTaskSlots: 18

blob.server.port: 6124
taskmanager.data.port: 6121

jobmanager:
  bind-host: 0.0.0.0
  rpc:
    address: <IP>
    port: 6123

  memory:
    process:
      size: 1600m

  execution:
    failover-strategy: region

taskmanager:
  bind-host: 0.0.0.0
  host: <IP>
  rpc.port: 6129
  collect-sink.port: 6130

rest:
  address: <IP>
  bind-address: 0.0.0.0
  port: 8084
```

Note that for readability purposes, we have skipped `env.java.opts.all` configuration property that is by default in the Flink configuration file.

In case you are using WSL, you may also need to ensure that all ports that are used by Flink inside of WSL are exposed as the ports of the host Windows machine. To do that, a Powershell terminal with Administrator permissions should be started, and the following command should be run for each port that is mentioned in the configuration file:
`netsh interface portproxy add v4tov4 listenport=<port> listenaddress=0.0.0.0 connectport=<same port> connectaddress=<WSL IP>`
Where `<WSL IP>` is the IP of the WSL container that can be found by running `ifconfig` from inside of WSL.

After the configuration files have been written, a single Job Manager needs to be started on one machine. To start that Job Manager, locate the Flink binaries in the terminal and start the Job Manager script:

```
cd <path/to/flink-1.19.0-bin-scala_2.12/flink-1.19.0>
bin/jobmanager.sh start-foreground
```

Afterwards, the following commands should be run on all machines in the cluster:

```
cd <path/to/flink-1.19.0-bin-scala_2.12/flink-1.19.0>
bin/taskmanager.sh start-foreground
```

If all goes well, you should be able to open the following URL in the browser: `<JOBMANAGER_IP>:8084` and see all Task Managers listed.

Changing Backend Configuration

First, a few ports and IP addresses have to be updated in the `kafka-connection.properties` file. In particular, the following ones should be adjusted:

```
# Kafka server URL (to be changed if external Flink cluster is used)
kafka.server.address=<kafka_IP>:<kafka_port>
bootstrap.servers=<kafka_IP>:<kafka_port>

flink.job.manager.ip = <job_manager_ip>
flink.job.manager.port = 8084
flink.parallelism = <parallelism>
```

In particular, the Job Manager's IP should be specified and the desired application parallelism should be specified. The parallelism should not exceed the number of slots in the cluster.

Additionally, in case Kafka was running locally (on `localhost`), you might have to update its IP address to an IP in the LAN. For that, you may have to restart Kafka and add the following two lines to Kafka's `config/server.properties`:

```
listeners=PLAINTEXT://0.0.0.0:9092
advertised.listeners=PLAINTEXT://<IP>:9092
```

Where `IP` is the IP address of the computer running Kafka. These two configuration changes (adding the two lines to Kafka configuration and adjusting the two Kafka-related lines in the `kafka-connection.properties` file) allow for all machines in the Flink's cluster to connect to the Kafka server.

The final change to make sure the external cluster is used in the application is to change which Java bean for the Flink environment is used in the application. In particular, in `AnomalyDetectionPipeline.java` class constructor, the qualifier has to be changed from `localFlinkEnv` to `distributedFlinkEnv`.

If all of these steps are done successfully, when started, the application will submit a job to the external Flink cluster, which will distribute the job to Task Managers.

Appendix E

OpenAPI Specification

Track Anomaly Detection - OpenAPI 3.0

Overview

This represents the OpenAPI Specification of the backend server of the application.

Tags

ships

Ship Information management component

notifications

Notification management component

Paths

GET `/ships/details/{id}`

Retrieve the most recent information about a ship.

Parameters

Type	Name	Description	Schema
path	**id** *required*	The ID of the ship	number

Responses

Code	Description	Links
200	Successful operation *Content* **application/json**	No Links
404	Ship not found	No Links
425	Backend pipeline still starting	No Links
500	Internal server error	No Links

GET `/ships/details`

Retrieve the current details of all the ships in the system.

Responses

Code	Description	Links
200	Successful operation *Content* **application/json**	No Links
425	Backend pipeline still starting	No Links
500	Internal server error	No Links

GET /ships/history/{id}

Retrieves all the CurrentShipDetails intances of the respective ship.

Parameters

Type	Name	Description	Schema
path	**id** *required*	The ID of the ship	number

Responses

Code	Description	Links
200	Successful operation *Content* **application/json**	No Links
500	Internal server error	No Links

GET /notifications

Gets all the notifications stored in the database

Responses

Code	Description	Links
200	Successful operation *Content* **application/json**	No Links

GET /notifications/{id}

Gets a certain notification from the database.

Parameters

Type	Name	Description	Schema
path	**id** *required*	The ID of the notification to be fetched	number

Responses

Code	Description	Links
200	Successful operation *Content* **application/json**	No Links
404	Notification not found	No Links

GET `/notifications/ship/{id}`

Gets all notifications of a particular ship.

Parameters

Type	Name	Description	Schema
path	**id** *required*	ID of the ship	number

Responses

Code	Description	Links
200	Successful operation *Content* **application/json**	No Links

Components

Schemas

CurrentShipDetails

Properties

Name	Description	Schema
currentAnomalyInformation *optional*		AnomalyInformation
currentAISSignal *optional*		AISSignal

Name	Description	Schema
maxAnomalyScoreInfo *optional*		MaxAnomalyScoreDetails

Notification

Properties

Name	Description	Schema
id *optional*	ID of the notification	number (long64)
shipId *optional*	ID of the corresponding ship	number (long64)
currentShipDetails *optional*		CurrentShipDetails

AnomalyInformation

Properties

Name	Description	Schema
score *optional*	Computed Anomaly Score	number (float64)
explanation *optional*	Corresponding explanation of the computed score	string
correspondingTimestamp *optional*	Corresponding timestamp	string (date-time)
id *optional*	ID of the corresponding ship	number (long64)

AISSignal

Properties

Name	Description	Schema
id *optional*	ID of the corresponding ship	string (long64)
speed *optional*	Speed of the vessel	number (float64)
longitude *optional*	Longitudinal coordinates of the vessel	number (float64)
latitude *optional*	Latitudinal coordinates of the vessel	number (float64)

Name	Description	Schema
course *optional*	Course of the ship	number (float64)
heading *optional*	Heading of the ship	number (float64)
timestamp *optional*	Timestamp of the signal	string (date-time)
departurePort *optional*	Departure port	string
receivedTime *optional*	Received time	string (date-time)

MaxAnomalyScoreDetails

Properties

Name	Description	Schema
maxAnomalyScore *optional*	Maximum anomaly score assigned to the respective ship	number (float64)
corresponding Timestamp *optional*	Timestamp corresponding to the maximum acheived anomaly score	string (date-time)

Appendix F

User Manual

F.1 Overview

This manual aims to explain how the users should use the website part Track Anomaly Detection application.

The anomaly score given by the application should only be taken as the recommendation which ships to look at. Final choice should be made by also evaluating the situation manually.

F.1.1 Terminology

- **Track** - the ship's path from the departure port to the destination.
- **AIS** - Automatic Identification System
- **AIS signal** - the signal that ships must frequently send. Such a signal contains information about the vessel: its position, rotation, destination, etc.
- **Anomaly score** - the score between 0% and 100% given by the application to a certain track. It tries to represent how suspiciously the ship is behaving in its track. However, note that it should not be taken as some "ground truth", and should only be taken as the suggestion what ships to look at, but further evaluation should be done manually.

F.1.2 General View of the Website

When the website is opened the view looks as in the Figure F.1. The components that are visible are marked in the Figure F.2.

The components that you see are:

- **Map**. The main component of the website. It displays the current ship positions, orientation, and the anomaly score (denoted by colour).
- **Information button**. The button for getting more information about what is displayed currently on the screen.
- **Sidebar**. The component containing functionality for displaying ships and notifications, as well as software errors occurred.

More information on how to use all the components is presented below.

Figure F.1: *The view of the website when started.*

F.2 Using the Map

Map allows the following functionality:

- **Zooming**. Zooming can be done using the mousewheel or the laptop touchpad. The map zooms based on the position of the mouse cursor. To avoid cluttering the map, some ships are grouped into the clusters.
- **Moving**. You can move in the map by clicking left mouse button and dragging the mouse (analogously, using the laptop touchpad).
- **Selecting the ship**. When the ship is clicked, the map zooms to the ship and ship's details are displayed near the sidebar. If the user does not zoom or move, the website keeps following the selected ship.
- **Clicking on the Cluster**. When the ship cluster is clicked, the map zooms to this cluster, and the view is refreshed.

Note that based on the options that developers set when launching the website, the map displays only the selected amount of most anomalous ships on the screen. To see more ships, the user needs to zoom closer to the region they want.

F.3 Displayed Ships

F.3.1 Individual Ship Markers

The ship markers represent the locations of ships based on the last sent AIS signal. If the ship's current speed is zero, the ship is displayed as a dot (Figure F.3a). Otherwise the ship is displayed as an arrow (Figure F.3b). Arrow is pointing according to the ship's current heading. If the heading informatino was not given, then the arrow is pointing according to the ship's course. The colour of the ship marker represents the anomaly score calculated for that ship. The colour is linearly interpolated from green (anomaly score 0%) to red (anomaly score 100%).

Figure F.2: *The view of the website when started with the main visible components marked.*

(a) *The ship marker for a stationary ship.*

(b) *The ship marker for a moving ship.*

Figure F.3: *Ship markers indicate the speed: dot is for stationary ship, and arrow is for moving ship.*

F.3.2 Ship Clusters

To avoid clutter on the map, ships are grouped into the clusters. An example for that can be seen in Figure F.4. **The number** on the cluster shows the maximum of the calculated anomaly scores for the ships inside the cluster. **The colour** of the cluster represents the same maximum anomaly score among the ships in the cluster. The colour value is calculated similarly to the ship colours.

F.3.3 Ship Trajectories

When a ship is selected (either by clicking on it in the map, or in the ships or notifications list), its track trajectory is displayed (for an example see Figure F.5). The points on the trajectory have the colour representing the anomaly score calculated for the moments when ship was on those points.

If the notifications list is opened (via the sidebar), and some notification is clicked, then the ship trajectory will highlight the point where the ship was at the moment of the notification.

F.4 Using the Sidebar and Information window

The sidebar (Figure F.6) contains the following sections, all of which open the corresponding information window when clicked:

- Ships (Section F.4.1)

(a) *The ship cluster which contains ships. The number 25% means that the maximum anomaly score of the ships in the cluster is 25%.*

(b) *The view zoomed into the area of cluster visible in F.4a. The three ships visible here were grouped into the cluster.*

Figure F.4: *Ship markers indicate the speed: dot is for stationary ship, and arrow is for moving ship.*

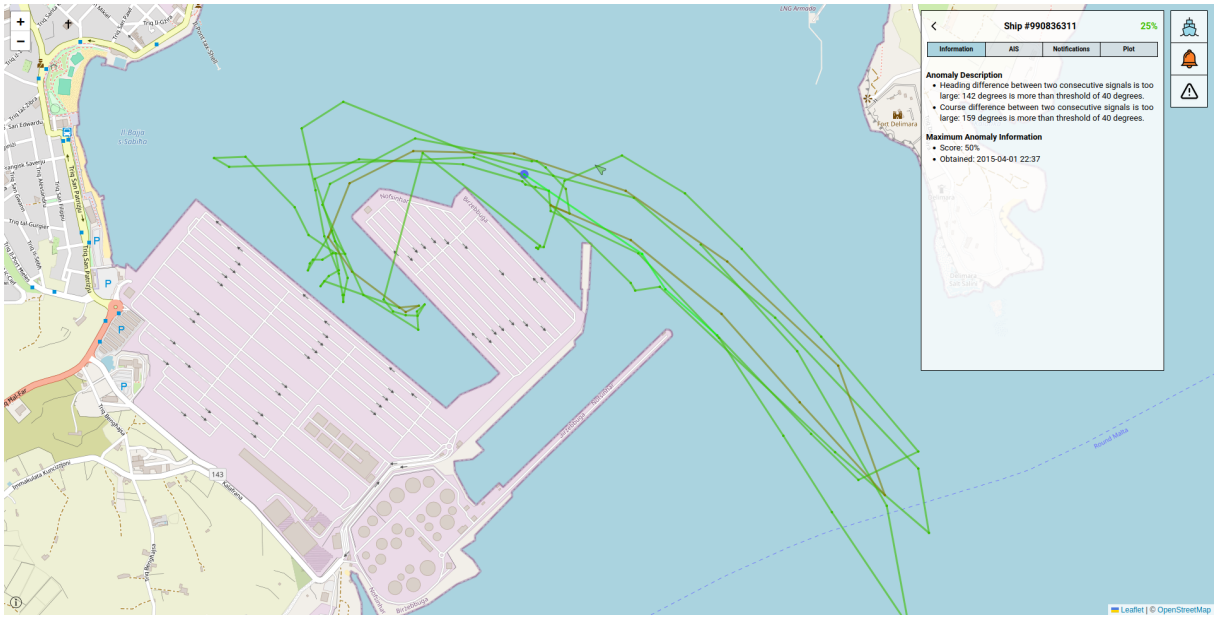


Figure F.5: *The trajectory of the selected ship. Colour represents the anomaly at each point of the trajectory.*

- Notifications (Section F.4.2)
- Software errors (Section F.4.3)

F.4.1 Ships

When Ships section is selected from the sidebar, the ship list is seen (Figure F.7). The shown ships are sorted based on their calculated anomaly score. Using the threshold slider you can select the threshold you want to use. Ships that have the anomaly value below the threshold will not be shown.

When a ship is clicked in the list, that ship is centered in the map, and its details are shown. In the details window you see the following sections:

- **Information** section (Figure F.8a) contains the description for the current anomaly score, and the information about the maximum anomaly score obtained throughout ship's journey.
- **AIS** section (Figure F.8b) contains details about the ship's last AIS signal data. It contains information, such as the time of the last signal, the departure port, the course and the heading, the latitude and the longitude, and the speed of the ship.
- **Notifications** section (Figure F.8c) contains a list of notifications sent about the selected ship.
- **Plot** section (Figure F.8d) shows a plot of the current ship's anomaly score changes throughout the journey. Points where notifications were sent are also marked in this plot.

Figure F.6: *The sidebar. It contains buttons for opening anomaly list, notifications list, and software errors list.*

Note on ship anomaly score explanations

Once the anomaly score is calculated, the explanation is generated as well. It can be found in the Information section of the ship details (Figure F.8a). The explanations contain the relevant data why the score was given.

F.4.2 Notifications

When the bell icon is clicked, the notification list appears (Figure F.9). When clicked on the notification, its corresponding ship is centered in the map.

Notifications are sent when the calculated anomaly score becomes bigger than 90%. Notifications can be marked as read when clicked on them, or the user can mark all notifications as read by clicking on the checkmark button.

F.4.3 Software Errors

Similarly to the notifications, when the warning icon is clicked, the software error list appears (Figure F.10). These software error messages contain the information about the errors occurred in the software, such as lost connection to the web server.

These software error messages are of three severity levels: errors, warnings and information. If only a warning or information is got, it usually indicates that only some minor issue appeared, and the application can continue to function normally. However, if the error appears, it indicates that a serious software issue happened that impacts the application.

F.5 Using the Information Button

On the left bottom corner of the screen, there is an information button. When you click it, it opens the information text (an example of this can be found in Figure F.11). This information text gives information about the components that are seen on the screen.

The information text is adaptive. If the user selects a different sidebar item, the information text changes to give information about that item.

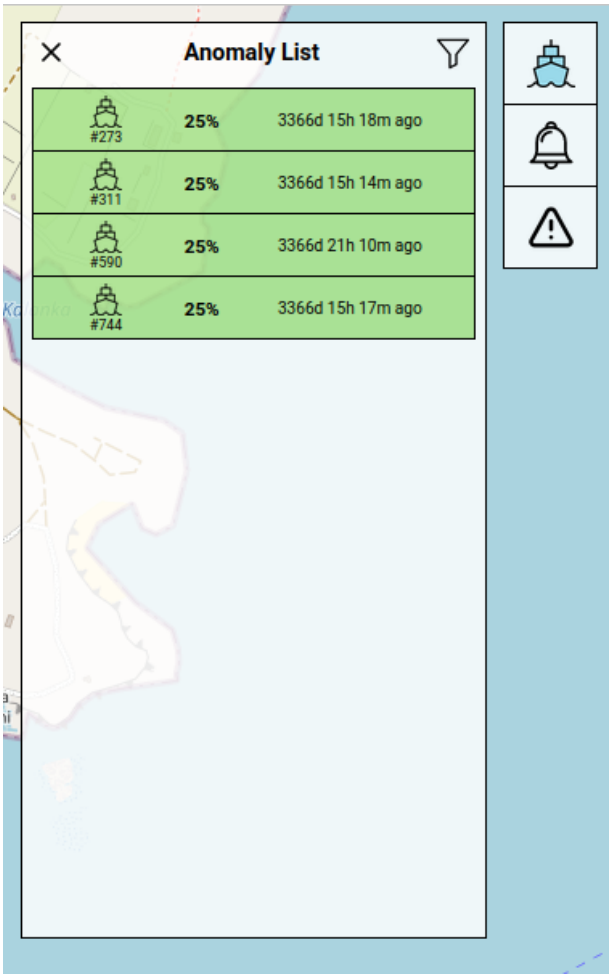
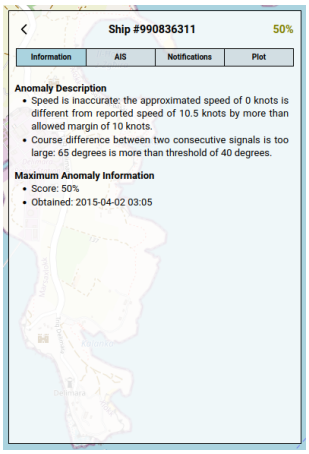
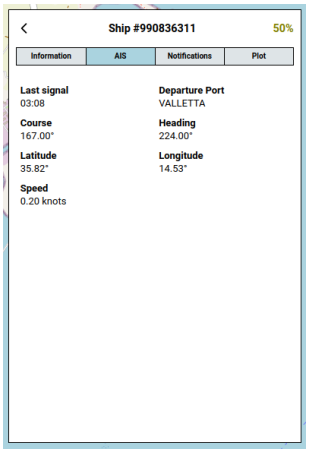


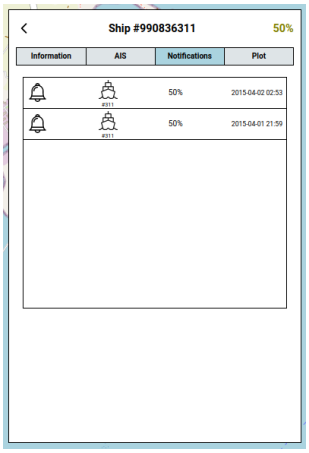
Figure F.7: *Opened ship anomaly list. The ships are shown in decreasing order of calculated anomaly scores.*



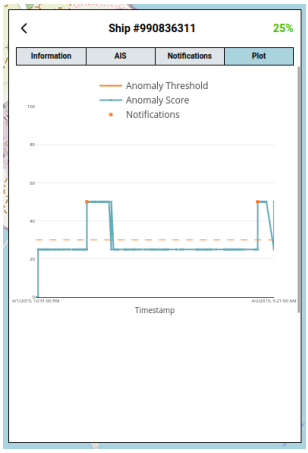
(a) *The anomaly description and maximum anomaly information section.*



(b) *AIS signal information section.*



(c) *Notifications section.*



(d) *Anomaly plot section.*

Figure F.8: *Different sections in the opened ship's details.*

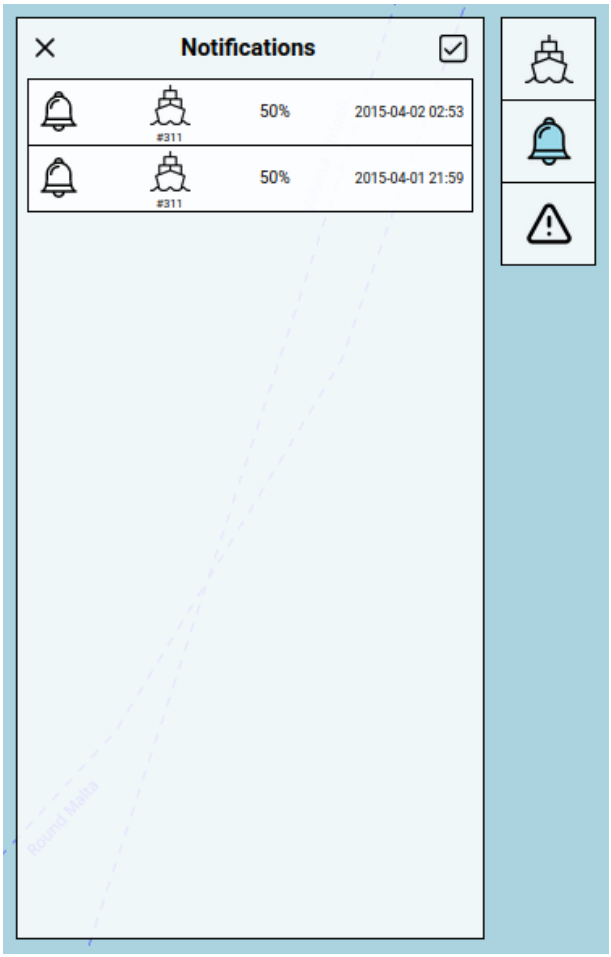


Figure F.9: *The opened notifications list.*

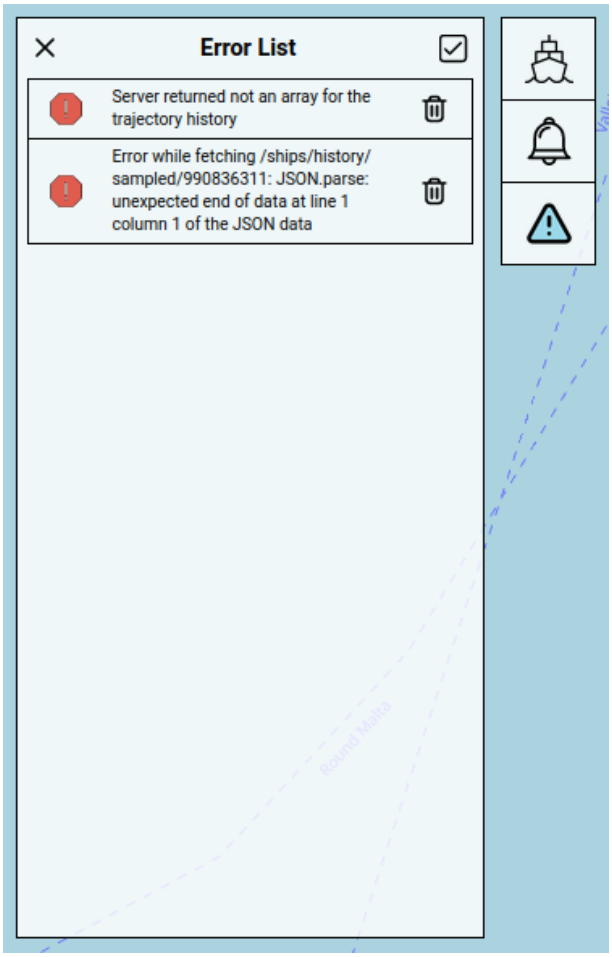


Figure F.10: *The opened software errors list.*

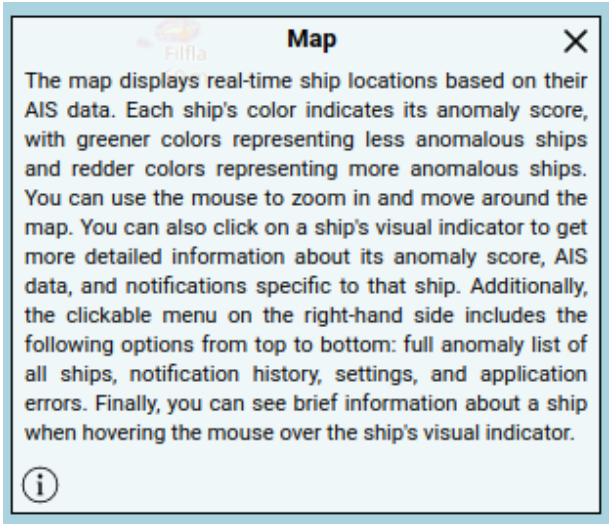


Figure F.11: *The opened information container.*

Appendix G

Division of Work

During the whole duration of the project, the team strived to achieve as equal work distribution as possible. That was accomplished as closely as possible. Everyone in the team contributed to both the frontend and the backend of the application. However, due to natural consequences, some people worked more on certain parts than others. For instance:

- Aldas worked more on Gitlab pipelines and DevOps.
- Augustinas worked more on distributability issues.
- Victor focused more on database-related issues and research (anomaly detection).
- Marius worked more on anomaly detection and accuracy testing part.
- Justinas worked more on the frontend and design problems.

Appendix H

Internal Rubric

Certain rules were set up that the team had to follow throughout the course to ensure complete fulfillment of the goals. The rules are listed as follows:

Exploration

- *Requirements engineering.*
  - The requirements are structured in a consistent way. Each item in the list of requirements is written in the same or very similar format (e.g., "The user is able to ...").
  - The requirements have been exhaustively discussed with the client until the client is satisfied and the team has no more questions.
- *Research / design methodology.*
  - For each major decision in the development process, a medium-sized document document is prepared, comparing alternatives and justifying the choices. Such documents are compiled *before* proceeding with the implementation of the piece of software under discussion. Only when all relevant parties agree with the ideas outlined in the document, the implementation can proceed. Examples of choices requiring such justification are technology stack, architectural choices, anomaly detection algorithm choices, etc.
  - All such justification documents are added to the final report.

Application

- *Software quality.*
  - No merge request is allowed to merge before passing the pipeline. The pipeline contains checks for Java checkstyle, front-end and back-end tests, test coverage, dependency vulnerability checks, and PMD.
  - No unnecessary Java warnings are present in the backend application.
  - At all times, unit tests cover at least 85% of the non-UI code of the application.
  - Some integration tests have to be written after W5.
  - Some UI testing has to be done after W5.
- *Design quality.*
  - As mentioned above, each serious design choice requires written justification reviewed by team members and other relevant parties.

    * In case ML techniques will be used, the document with proposed techniques is shown to the coach for a discussion.

- *Meeting expectations.*
    * On-demand meetings are set up with the client. Whenever the team or the client requires information, either a meeting or a written correspondence is initiated.
    * At all times, the project plan is followed. If significant additions or changes need to occur to the plan or requirements, they are first discussed with the client to make sure both parties agree.
- *Quality of documentation.*
    * Each method, both in the frontend and the backend, has a comment documenting it.
    * As mentioned, the important design choices are described in a document.
    * A final diagram of the overview structure of the application is drawn at the end of the project.
    * As new components are added to the project, README files are immediately updated. For each part (backend and frontend) there is a separate readme file. Each of them contains at least 2 sections: (1) building the project and (2) running the project. The part about building the project has to contain guidelines for installing the dependencies of a project. These two sections are updated whenever relevant changes are made.
    * An OpenAPI documentation is prepared for the endpoints exposed by the backend.
    * Changes are also well documented in Gitlab issues and merge requests.
    * Each issue and merge request is written in a manner that is understandable to a less-informed reader. It should be fully understandable to the person grading.
    * Each issue/merge request is done properly: clear (capitalized) title, concise but not too short description, attached labels, time tracking added, milestone attached.

Process

- *Planning.*
    * Each change to the project has (a) corresponding issue(s) and merge requests.
    * All non-MR/Issue-template changes are first merged to dev branch.
    * Issues are assigned at the beginning of each sprint, usually according to the initial priority order of the requirements.
    * Gitlab statistics are used to track the progress of the team. If one observes that the team keeps missing the deadlines, necessary interventions are made: the issues may be differently distributed, or their prioritization might change (after discussion with the client).
    * Initial plan is followed: finishing the MVP in 4 weeks, finishing SHOULD requirements in 7 weeks.
    * Each week, a Sprit retrospective is filled out.
- *Initiative, creativity & reflection.*
    * Once again, each significant improvement is documented in the report.
- *Team interaction and communication.*
    * The agenda for the TA meeting is sent a day prior to the meeting, at the latest 20:00.
    * The minutes for the TA meeting are uploaded on the same day of the meeting to the Gitlab repository and also sent to the TA.