Bachelor's Thesis

Business Information Technology

2020

Antti Rae

PRODUCING AND STORING MARITIME DATA USING SHIP BRIDGE SIMULATORS

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Business Information Technology

2020 | 27 pages

Antti Rae

PRODUCING AND STORING MARITIME DATA USING SHIP BRIDGE SIMULATORS

The ships and boats of the maritime sector are a growing source of big data. This maritime data consists of multiple different data formats such as AIS and NMEA that provide information about the ships at sea, such as their location and current in-depth status of the vessel. Sharing this maritime data to software developers and data analytics could have a wide-ranging impact on the maritime sector and the way maritime data is treated. Giving application developers and data analytics an easy and open access to maritime data would open the doors for many innovations and new solutions to old problems in the maritime sector.

The goal of this thesis is to explore ways to collect maritime data from ship bridge simulators located in Novia University of Applied Sciences. Furthermore, how to store maritime data in a database and share the stored data to other parties interested in maritime data. The thesis also does research on what benefits can be derived from maritime data and what information can be decoded from said data. This thesis is a case study using constructive research method.

The main result of this thesis is a platform that can be used for recording maritime data from ship bridge simulators. The recorded data simulator sessions are stored in a database from which interested parties can retrieve the data for use cases such as software development and data analysis. This thesis also proposes some future development ideas for the platform. The thesis results contain a brief look into maritime systems and their data formats.

The results of this thesis can be utilized when thinking about future development of the platform. Novia University of Applied Sciences also benefits from the resulted platform when recording and storing maritime data.

KEYWORDS:

Big data, Maritime data, AIS, NMEA Standards, Django Framework

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietojenkäsittely

2020 | 27 sivua

Antti Rae

MERENKULKUDATAN LUOMINEN JA SÄILÖMINEN LAIVAN KOMENTOSILTASIMULAATTORIEN AVULLA

Merenkulkualan veneet ja laivat ovat kasvava lähde massadatalle. Tämä merenkulundata sisältää useaa eri dataformaattia, joita ovat muun muassa AIS ja NMEA. Kyseinen data kertoo paljon vesillä olevista aluksista, kuten aluksen sijainnin ja yksityskohtaista tietoa aluksen tilasta. Tällaisen tiedon jakamisella sovelluskehittäjille ja data-analyytikoille voi olla laajoja vaikutuksia merenkulkuun ja melenkulkudatan käsittelyyn. Kun sovelluskehittäjille ja data-analytikoille annetaan helppo ja avoin pääsy merenkulkudataan, se mahdollistaisi tehokkaampaa merellä kulkemista esimerkiksi polttoaineen kulutusta analysoimalla.

Tämän opinnäytetyön tarkoituksena oli selvittää, kuinka merenkulkudataa voidaan ottaa talteen toimeksiantajan Novia ammattikorkeakoulun laivan komentosiltasimulaattoreista ja kuinka tallennettua tietoa voidaan jakaa kiinnostuneille osapuolille. Työn tarkoituksena oli myös tutkia, mitä hyötyä tallennetusta tiedosta on ja mitä asioita siitä voidaan tulkita. Opinnäytetyö toteutettiin tapaustutkimuksena konstruktiivista tutkimusotetta käyttäen.

Tämän opinnäytetyön tuloksena oli alusta, jolla voidaan käyttää hyväksi laivan komentosiltasimulaattoreita merenkulkudatan luomiseen ja tallentamiseen. Tieto tallennetaan tietokantaan simulaattorisessioina. Kiinnostuneet osapuolet voivat noutaa kyseisiä simulaattorisessioita omia tarkoituksiaan varten. Esimerkiksi tutkimukseen tai sovelluskehitykseen. Koska alustaa ei saatu täysin valmiiksi, opinnäytetyön tuloksista selviää myös jatkokehitysideoita alustaa varten. Opinnäytetyön teoriaosuuden tuloksia oli selvitys merenkulkujärjestelmien toiminnasta, merenkulkudatan formaaattien tulkinnasta ja hyötykäyttökohteiden tunnistuksesta.

Työn tuloksia voidaan hyödyntää alustan jatkokehitystyössä. Toimeksiantaja voi hyödyntää kehitettyä alustaa merenkulkudatan luomiseen ja tallentamiseen.

ASIASANAT:

massadata, merenkulkudata, AIS, NMEA standardit, Django-viitekehys

CONTENTS

FIGURES

LIST OF ABBREVIATIONS

AIS	Automatic Identification System is a system for tracking ships. AIS technology is based on satellites and it provides identification and other useful data of ships.
API	Application Programming Interface. A computing interface for different computing intermediaries.
ASGI	Asynchronous Server Gateway Interface
NMEA 0183	Dominant communications standard for maritime electronics used in commercial shipping. Connects ship's sensors and display units. It is slowly being phased out in favor of NMEA 2000 (National Marine Electronics Association, 2002)
NMEA 2000	The newest communications standard for maritime electronics used in commercial shipping. Connects ship's sensors and display units. (Spitzer, Luft, & Morschhauser, 2009)
NMEA	National Marine Electronics Association is responsible for NMEA standards, among other things.
NoSQL	Not only SQL. A term used for describing databases that differ from the traditional relational databases
ORM	Object-Relational Mapper
REST	Representational State Transfer, a software architecture style with a set of constraints used when creating Web services
SQL	Structured Query Language, the main language used in relational databases
WS	WebSocket is a protocol allowing two-way communication between a client and a server without opening multiple HTTP connections.(Internet Engineering Task Force, 2011)
WSGI	Web Server Gateway Interface

1 INTRODUCTION

Big data is a relatively new topic in the maritime sector. Currently many vessels at sea generate large amounts of data that could be used for achieving a benefit for the maritime sector. Given access to the maritime data, software developers and data analytics could increase the efficiency of vessels at sea and create other benefits for the whole maritime sector with new solutions and innovations.

This thesis will propose a way to utilize ship bridge simulators to generate maritime data and store it using a database for future uses, such as app development and research. Using already existing ship bridge simulators located in Novia University of Applied Sciences one is able to cost effectively generate and collect the data without the need of costly real-life vessels. The thesis will focus on the development of a platform for recording maritime data in formats such as AIS and NMEA and storing it in a database. Clients can then retrieve the data in various ways from the platform for their own use cases. The platform uses HTTP and WebSocket protocols for data transfer between clients and the platform. HTTP for sending bulks of data and WebSocket for sending data in real-time in the same manner that it was recorded from the simulators.

The thesis will also go into some details about the maritime data formats AIS and NMEA. Describing what one can learn from the data and what uses there theoretically could be for the data. Lastly some web development tools, and their use cases will be discussed. In the closing chapters some future suggestions for the platform are described along with a conclusion.

2 MARITIME DATA – STANDARDS AND PROTOCOLS

2.1 NMEA Standards and Protocol Data

Almost every modern vessel has some sort of marine electronics these days. Marine electronics are devices such as marine radios, sonars, and marine radars. These devices are used for example, in steering the vessel and navigating. Modern vessels have a large number of marine electronics that are connected to each other. The marine electronics use specifications defined by NMEA to communicate between each other. The National Marine Electronics has introduced two standards, the older NMEA 0183, and the NMEA 2000 for data communications among shipboard electronic devices. (Luft, Anderson, & Cassidy, 2002)

NMEA, the national marine electronics association is an association committed to enhancing the technology and safety of marine electronics. The association offers services such as training alongside with its standards. NMEA owns the NMEA 2000 and NMEA 0183 interface standards as its' intellectual property and published updates to the standards from time to time. (NMEA, 2020)

2.1.1 NMEA 0183 Standard

The National Marine Electronics (NMEA) first released the NMEA 0183 standard in March of 1983. NMEA 0183 is a voluntary industry standard and updated from time to time. (Betke, 2000) NMEA 0183 can operate at 4800 bits/second allowing approximately up to ten messages per second. This is adequate for a single device broadcasting to others to use. However, a limit is quickly reached when systems start to combine, and more devices are added. Nevertheless NMEA 0183 is still expected to be used well into the future for certain simpler use cases such as, backup data connections and direct device-to-device connections. (Luft et al., 2002)

All NMEA 0183 data is transmitted in the form of ASCII character sentences. The sentences start with a "$" character and end with <CR><LF> tags. NMEA 0183 has three kinds of basic sentences are talker sentences, proprietary sentences, and query sentences. Talker sentences are sentences sent by talking devices to listeners. The sentences contain information generated by the talking device. (Betke, 2000)

```
$HCHDM,238,M<CR><LF>
```

Figure 1. Example NMEA 0183 talker sentence.

Figure 1 contains a talker sentence from a magnetic compass. The sentence's first two letters following the ”$” character are the talker identifier, in this example specifiying that the message is from a magnetic compass. The next three characters are the sentence identifier telling that a magnetic heading message follows. The sentence identifier is followed by a number of data fields seperated by commas. In the example in figure 1 the heading is 238 and the ”M” designates that the heading is magnetic. (Betke, 2000)

Proprietary sentences are allowed in the standard for manufacters to define their own sentence formats. These sentences start with the ”$P” characters followed by a 3 letter manufacturer ID and data that the manufacturer wishes. Query sentences are sentences for requesting data from a talker device. (Betke, 2000)

```
$CCGPQ,GGA
```

Figure 2. Example query sentence.

Figure 2 illustrates a query sentence sent to a GPS device ”GP” from ”CC” talker, requesting for a ”GGA” sentence. Upon receiving the sentence the GPS will transmit the requested sentence once per second until a new query sentence is received by the GPS. The sentence's first two characters consists of talker identifier of the requester. Then the next two characters define talker identifier of the device being queried (listener) and the fifth character ”Q” defines the message as a query. (Betke, 2000)

2.1.2 NMEA 2000 Standard

NMEA 2000 is a low-cost serial data network taking advantage of the Controller Area Network (CAN) that was originally developed for use in cars. NMEA 2000 allows for multiple electronic devices to be connected to each other via a common channel for communication purposes between devices. The key difference between NMEA 0183 and NMEA 2000 apart from speed is that NMEA 0183 is an interface and NMEA 2000 is a network. (Luft et al., 2002)

NMEA 2000 messages are predetermined parameter group numbers (PGN). PGNs are constantly being added to NMEA 2000. For example, PGN 128275 is a distance log parameter group (message). The distance log PGN provides the cumulative voyage distance traveled since last reset, along with timestamps of the distance measurements. The PGN has a number of parameters. In this example there are four parameter fields. The fields contain the information, information such as, the total distance, time of measurement and distance since last reset. (Luft et al., 2002)

2.2 Automatic Identification System – AIS

The automatic identification system (AIS) is widely used system in commercial shipping and other big vessels for identification and collision avoidance. AIS acts like a transponder, operating in the maritime VHF radio band. From AIS data one can learn many useful aspects about the vessel in question. Ship name, course and speed, classification, and other information. Each automatic identification system consist of different VHF receivers and transmitters also marine electronic communication links are used to communicate to ship display and sensor systems. Most information broadcast by AIS is obtained electronically from shipboard equipment through standard marine data connections. (U.S. Coast Guard Navigation Center, 2020)

AIS works autonomously and continuously regardless where the vessel is located currently. Each ship reports messages to AIS stations with information such as location. The stations have their own transmission schedule. The schedule is based on previous communications and known future actions by other stations. A position report from a vessel fits into one of 2250 time slots into the transmission schedule established by the AIS stations. The stations continuously synchronize themselves with each other, this way overlap of slot transmissions is avoided. As the vessels transmit their information to the stations, they also reserve another time slot for future messages. (U.S. Coast Guard Navigation Center, 2020)

AIS consist of multiple categories of data. Dynamic, static, voyage and safety related data. The data from the categories is encoded into the AIS data that is broadcasted to other AIS stations. The categories contain optional and required data. Dynamic data is data coming from different sensors and encoded things like heading and rate of turn are encoded from this category. Static data is data that is encoded at the time of installation of the AIS. It contains information such as vessel type and call sign. Voyage related data is information about the current voyage the vessel is taking. Encoded data for example is estimated time of arrival (ETA) and destination. Lastly safety-related text messaging is used for concise exchange of safety-related information messages. (USCG, 2020)

2.2.1 AIS Data Examples

Ship types, for example, a typical oil tanker ship type is 81. The first digit, 8, telling that the ship is a tanker and the last digit 1 telling that it is carrying a hazard or a pollutant of certain kind. Being static data, the ship type is set at the time of installation of the system. (USCG, 2020) Navigation status should always be up-to-date information about the navigational status. For example, at anchor, underway or moored. The information is encoded with number denoting different statuses. 1 meaning the vessel is at anchor, 0 meaning the vessel is under way using engine. There are also some more specific values, for example, 11 meaning that the vessel is towing astern. (MarineTraffic, 2018)

```
!AIVDM,1,1,,B,177KQJ5000G?tO`K>RA1wUbN0TKH,0*5C
!AIVDO,1,1,,B,H1c2;qDTijklmno31<<C970`43<1,0*28
```

Figure 3. Example AIVDM and AIVDO (AIS) sentences

Figure 3 contains a AIVDM and AIVDO sentences emitted by a AIS receiver. AIVDM and AIVDO sentences are NMEA 0183 sentence variants. There are seven fields in a typical AIVDM data packet. The fields are seperated by commas. In the figure 3 example the first field 1 identifies the packet as AIVDM packet. The next fields two and three are message fragment numbers for cases where the message is too long for a single message. Field 4 contains the message id that is used for multi-sentence messages. Field 5 contains the radio channel code. Field 6 contains the actual payload data that can be decoded into human readable information. Lastly field 7 is the number of fill bits used to pad the payload to a 6 bit boundary. (Raymond, 2019)

When the example sentence's payload in figure 3 is decoded a lot of parameters containing in-depth information are revealed. Most notable pieces of information are included in parameters such as longitude, latitude, true heading, and navigational status as seen in figure 4 where one of the example sentences was decoded. (Maritec Solutions, 2020)

Parm#	Parameter	Value	Description
01	Message ID	1	
02	Repeat indicator	0	No repeat (default)
03	User ID (MMSI)	477553000	
04	Navigational status	5	Moored
05	Rate of turn ROTAIS	+0.0	
06	SOG	0.0	
07	Position accuracy	0	Low (> 10 m) (default)
08	Longitude	122.3458333	
09	Latitude	47.5828333	
10	COG	51.0	
11	True heading	181	
12	Time stamp	15	
13	Special manoeuvre indicator	0	
14	Spare	0	
15	RAIM-flag	0	RAIM not in use (default)

Figure 4. Decoded AIVDM sentence from figure 3.

2.3 Maritime data - Possible Use Case Examples

Big data could play an important role in the maritime business. By analyzing data obtained from previously used vessels' sensors benefits could be achieved in ship design, decision making and many other areas. There are multiple key application areas for maritime big data usage. In chartering data analytics could use automatic identification system information for finding alternative routes and freight forecasts. This provides ship owners more options, potentially improving efficiency. Benefits in ship operations such as maintenance can also achieved by analyzing vessel performance. (Marine Digital GmbH, 2020)

3 IMPLEMENTING A PLATFORM FOR MARITIME DATA

3.1 Tools Used

Various IT-tools were used in the development and deployment of the proof of concept platform. Primarily the tools were open source software, and their source code is open for everyone to see. The fact that most tools and technologies used were open source, helped the project in numerous ways. Good and publicly available documentation and the free to use aspects of open source software were compelling arguments for the use of open source technologies. The tools varied in their use cases, from running the actual platform to accessing the server used for the platform deployment.

3.1.1 Django Framework and MongoDB

Django is an open source Python web framework that enables developers to build web applications rapidly thanks to its' clean and pragmatic approach to development and the extensive documentation available online. (Django Software Foundation, 2020) Django framework was the main tool used in the development of the platform and it proved to be a good choice for developing the platform for it was easy to pick up and learn and its' database management ORM was beneficial in many ways. Django's ORM system makes database interfacing simpler by mapping data models to database tables.

MongoDB is a database based on JSON-like documents. MongoDB is more flexible than the traditional SQL databases. (MongoDB, 2020) MongoDB allows the data structure to be changed over time and the documents in the database can be different. The data does not have to be uniform between the documents. MongoDB is an open source project. Choosing the database for the project was a choice between a SQL and NoSQL database. The choice was made in favor of MongoDB because the nature of the incoming data was not fully known at that point and there were preservations about the data's structure being uniform. Also, the planned volume of the data was so large that mongo DB was a natural choice.

3.1.2 Gunicorn and Daphne

Gunicorn and Daphne are HTTP servers that were used to service the incoming requests coming to the server. Gunicorn is a WSGI HTTP server that can handle traditional HTTP requests. (Chesneau, 2020) Daphne is also an HTTP server but in addition it supports WebSocket protocol. (Gibson, 2020) The WebSocket protocol support was important since the project needed to serve data through web sockets. Both Gunicorn and Daphne were used at one point or another during the project. Due to technical difficulties and time limitations the solutions related to the used HTTP servers were not optimal. In the end the platform was running Gunicorn to server basic HTTP requests and Daphne to handle WebSocket traffic. Daphne supports both WSGI and ASGI applications so Gunicorn would have been possible to remove from the platform to simplify the design.

3.1.3 Supervisor and Other Tools

Supervisor is a client and server system that allows its users to monitor and control a number of processes on UNIX-like operating systems. (Supervisor Developers, 2018) For the project supervisor was used for its process monitoring and its ability to handle the possible crashes that would ensue by unhandled errors or power outages. There were other possible choices of process monitoring and handling software but in retrospect Supervisor seemed a good choice for its ease of use and it used the Python programming language as almost every other tool. Other choices were PM2 and systemd which are similar programs but made with different programming languages and development approaches.

In addition to the bigger tools mentioned before there were also some minor tools used that are worth a mention. Nginx being one of them. Nginx is a HTTP server and reverse proxy. (F5, 2020) In the project Nginx was used as a proxy server to forward the incoming and outgoing traffic to and from Gunicorn. Also, Nginx acted as the edge for the platform, connecting to the internet and acting as a HTTPS server. In order to enable HTTPS for connections SSL certifications were generated and used in Nginx.

The developed platform was put on production server with a Debian based Linux server operating system distribution. This was largely due to what Novia University of Applied Sciences had to offer and Linux in general is a stable choice for software that needs to

be running around the clock in a server environment. Also, Linux has a robust security by default that requires minimal configuration. The production server was accessed via VPN (Virtual Private Network) and PuTTY SSH (Secure Shell) client using a password and username. The production server was restricted to an internal network this provided the project with a secure place to test the platform without going public but still retaining the option to open the service for public clients later on.

3.2 Development Process

The development of the platform took place in 3-month period. During which the main functionalities of the platform were designed and implemented. Some of the designed features did not see development during the 3 months due to time limitations. But these missing features and functionalities did not hinder the final product since the main and most important functionalities were developed and present at the end of the project. And should the project be continued in the future all the source code and basic documentation was archived for later use and development.

The development process can be divided into three distinct phases, planning phase, development phase and deployment phase. Each taking different amounts of time and effort. These phases overlapped when developing the platform. The reason for this was design oversights that came up during development or minor change requests from the client that required more development time. In hindsight these problems would have been easy to avoid but its' the nature of software development for these kinds of problems to arise and avoiding them might not always be possible due to inexperience from the developers, lack of time or changing requirements by the client.

3.3 Planning Phase

The design process for the platform consisted of multiple different aspects that needed consideration. Mainly the needs of the client, Novia University of Applied Sciences, were in the spotlight. Taking the requirements Novia provided into the designs proved to be a challenge when thinking about the specifics of each part of the platform. Multiple questions needed answers. Questions such as what the data would look like, how the data should be stored and redistributed forward. Multiple meetings took place to discuss these questions and seek suitable solutions.

3.3.1 The Platform

During the planning multiple figures and pictures were drawn to illustrate how the platform would work in practice to help in the development phase. These illustrations included architectural designs of the whole platform and its' pieces, as seen in Figure 5 where an overview of the whole platform is pictured. The platform consists of a few key components. These components include the Ship bridge simulator as the main source of incoming data for the rest of the platform. The forwarding script that acts as a middleman between the main backend and the simulator. The forwarding script listens to the simulator via Wi-Fi connection and collects all the incoming data as a simulator session. The simulator session consists of NMEA and AIS data in text format and some basic information about the session such as time of recording, name and a description of the recorded session. The last major piece of the platform is the Django REST backend. The backend receives data from the forwarding script and after validation writes the data into a NoSQL MongoDB database. Clients can retrieve the stored data from the database through the backend or by using the frontend user interface. The data can be retrieved via a WebSocket connection or a traditional HTTP request through the REST API.

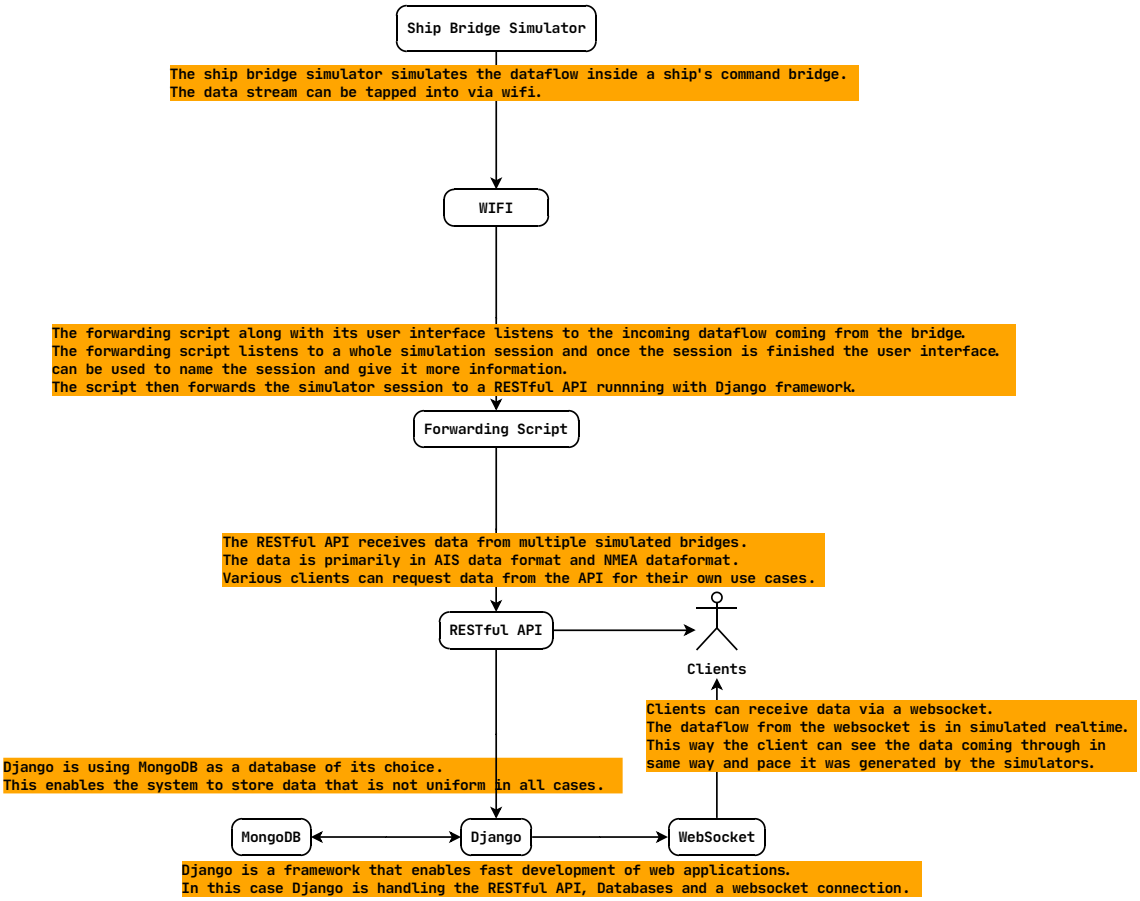

Figure 5. Architectural design of the platform.

3.3.2 Backend design

Database Design

In order to accommodate all the incoming data a suitable database was needed. Since the data was not always uniform in its format a database was needed that could handle changing data formats. Traditional SQL databases that use tables and relations between tables wouldn't work very well since the incoming data would not necessarily conform to the designed tables. This is why MongoDB was a good choice for the platform. MongoDB works in a way that allows for flexible data storage regardless of the data types included in the data. MongoDB uses documents as a way to store information. These documents' data can be widely different from each other. In addition, the documents do not have relational mapping allowing them to exist independently. Documents do not impose any limitations on the types of data. Also, the choice of MongoDB would mean that the actual

database design would be simple since MongoDB does not use tables and relations that would require designing.

Choosing the Web Framework

When choosing the technologies for developing a RESTful API several factors were taken into consideration. Main factors for determining the suitability for the project were database support, coding language, support for the needed platform features and availability of documentation. Choosing a proper and suitable framework for the platform was crucial in the long term, since some features would be added later and support for these features should be manageable without having to refactor too much of the already existing code. In the end Django framework was chosen for its' excellent online documentation, ease of use and extensive selection of features. Also, in addition of Django framework the project's rest API was built with Django rest framework.

3.3.3 Frontend design

In figure 6 there is a Mockup made with Adobe XD showcasing what the user interface could look like. In the interface users could see their own recorded simulator sessions and prerecorded scenarios containing data of simulated ships undergoing different navigational scenarios. The mockup seen in figure 6 was an early mockup that would not see development due to time constraints. But the mockup is something that could be implemented over the designed backend. Due to the time constraints the project opted for using Django template engine to create the frontend user interface. This way the user interface was simpler and quicker to design and implement since it was incorporated into the backend. A separate frontend would have cost too much time and effort for the project's allocated time.

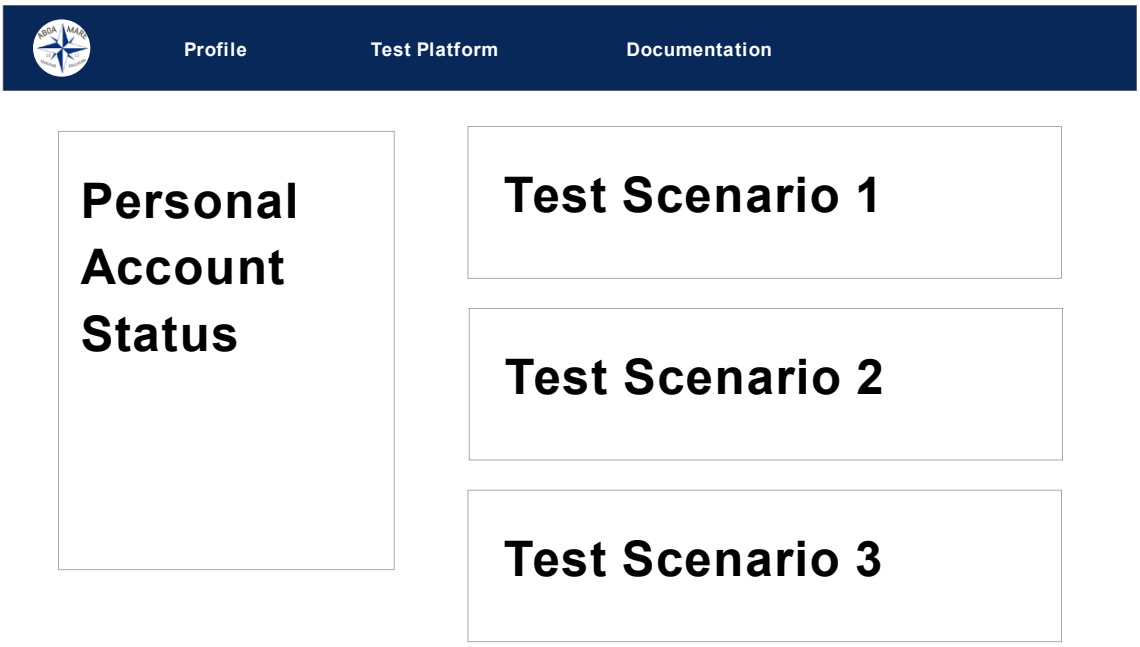

Figure 6. User interface mockup.

3.4 Development Phase

The development phase was the longest the primary phase for the project. During this time most of the coding and development was done. The first major part of the platform that took shape was the Django backend. The Django backend consists of multiple parts of code that is separated into files containing parts of the back-end's functionalities. These files lean on each other and rely on the other files for their functions and classes. These files can be created by the developers as they will, but Django will include some default files when generating projects and apps. The default files include files such as models, views and tests that make up a bare bones app that contains the most basic functionalities enabling the app's functionalities to work. The generated files act as a starting point for further development.

3.4.1 Django and Databases

The models file includes model classes that are used to define models for Django's ORM system. The model class contains attributes that are defined using Django's models class' field methods. For example, CharField() and IntegerField() are used to define a database character and integer attribute fields. The field methods take arguments such

as max length and default for more specific definition of the attributes and enables developers to restrict user input to certain parameters and avoid unwanted data causing problems.

The class represents a SQL database table and the class' attributes are the table's attributes. Due to the project using MongoDB, which is not a SQL database that Django supports by default, an additional python package called Djongo was used to enable support for MongoDB. Djongo acts as a connector between Django ORM and MongoDB. Djongo works in a way that requires no changes to the Django ORM. The only change in the code Djongo requires is the use of Djongo's own DjongoManager class instead of Django's own manager. The manager is used in other parts of the code for interacting with the model for actions such as writing to the database or reading from it.

```
class SimulationDataEntry(models.Model):
          description = models.CharField(max_length=255)
          rating = models.IntegerField(
                    default=1,
                    validators=[
                              MaxValueValidator(10),
                              MinValueValidator(0)
                    ])
          session_start = models.DateTimeField()
          session_name = models.CharField(max_length=50,
          unique=True)
          data = models.ArrayModelField(
                    model_container=SimulationData,
                    model_form_class=SimulationDataForm
                    )
          manager = models.DjongoManager()
```

Figure 7. Example code snippet of a simulation data model.

As seen in figure 7 the data model includes basic fields like CharField() and IntegerField() for session description, name, and rating. But also, some more complicated fields such as an ArrayModelField() for embedding a child document inside a parent document. The JSON snippet in Figure 8 corresponds to the example code snippet's class in the format it would be in the MongoDB. The data field contains an array of data sets. The data sets each have a timestamp and the actual data. The data can be in NMEA or AIS format. Other fields include information about the recorded simulation session such as start time, name, rating, and description.

```
{
    "session_start": "2019-11-16T10:45:00.391000Z",
    "session_name": "example session name",
    "description": "description of the session",
    "rating": 5,
    "data": [{
        "event_date": "2019-11-16T10:45:00.391000Z",
        "simulation_data": "$GPZDA,050508,05,07,2019,05,00*4D\n"
        }]
}
```


Figure 8. JSON snippet from MongoDB of a data entry.

3.4.2 Django with WebSocket Protocol

In order to get the requested real time flow of data for the clients, WebSocket connections were used for their ability for two way and real time communication between the client and server. WebSocket connection allowed for streaming recorded simulator session data to clients in the same order and pace as it did when it was recorded from the simulator. Django framework does not support WebSocket connections by default due to Django's synchronous nature. Django Channels makes some changes to Django's synchronous code allowing for asynchronous actions and enables protocols that require long-running connections, such as WebSocket in this case. The WebSocket connections are used for serving data to clients in real-time. (Godwin & Django Software Foundation, 2020)

The developed WebSocket functionality uses a class called consumer. This consumer is an abstraction that allows for easier ASGI application development. The consumer consists of a series of functions that are called whenever an appropriate event happens in the program. For example, in the Figure 9 the consumer function websocket_disconnect is called when a WebSocket is disconnected from the server. Other consumer functions for example are called when a WebSocket connection is created or a WebSocket receives messages. In the functions, code is written for handling each of the events appropriately. The code and logic written for receiving messages through the WebSocket reads the incoming messages and if the message contains the correct information Django will start streaming the requested data set to the client back through the same WebSocket connection.

```
async def websocket_disconnect(self, event):
     print("disconnect", event)
```

Figure 9. Consumer function websocket_disconnect.

3.5 Deployment Process

At some point during the development one should start considering where and how to deploy. For the platform, several points were taken into consideration, such as the deployment environment, how to run the code and handle runtime errors when they arise without the whole platform becoming inoperable. The deployment phase consists of installing and configuring a suitable server for the platform and also uploading the source code to the production server and having it run in a proper manner, that is, all the time and if crashes occur the code runtime should be restored automatically.

For running a Django based application in a production environment one usually needs to use a HTTP server such as Gunicorn or Daphne. They are also recommended in the Django's online documentation. (Django Software Foundation, 2020) Making them a good and supported choice for a HTTP server. For the case of the platform a proxy server was used to act as an edge point between the internal network and public network. For the choice of the proxy server NGINX was used. NGINX handles incoming traffic and forwards it to the HTTP server for handling. NGINX encrypts and decrypts HTTPS traffic to enable secure connections to clients requesting and receiving data. The last piece of software used for handling the platform in production is Supervisor. Supervisor is a process controlling and monitoring system that allows for handling processes on UNIX-like operating systems.

3.5.1 Production Server

For a production environment a single server was used. The server machine was running a Debian GNU/Linux operating system. To prepare the server for production multiple objectives were set out initially. For a proper accessing method to the server a SSH connection was used. Also, a VPN was required since the server was closed in the school's private internal network. For the proposed platform to function properly multiple pieces of software needed to be installed and configured. Django and all its'

dependencies needed to be installed. For this python's package manager (PIP) was used inside Python a virtual environment. A python virtual environment is a self-contained directory tree that contains a Python installation and additional packages. (Python Software Foundation, 2020) Django would be running inside the virtual environment and other programs would communicate to Django inside the virtual environment. For software not available via the PIP package manager an alternative package manager called APT was used. Advanced Package Tool (APT) provides a high-level command line interface for managing packages. (Debian Project, 2019) APT was used for installing MongoDB and NGINX among other things.

4 CONCLUSIONS

One of the main goals of this thesis was to develop a solution to a problem with ship bridge simulators. The problem involved storing maritime data from the ship bridge simulators and redistributing said data to other parties with a interest in maritime data. Redistributing maritime data opens many new possibilities for the maritime sector allowing for data analytics and software developers to improve the effiency of sea vessels

This thesis has introduced a platform for distributing and recording data from ship bridge simulators located in Novia University of Applied Sciences. The platform allows for recording maritime data into a database from ship bridge simulator sessions and distribute it forward to other parties interested in said data. The platform was deployed on a GNU/Linux server machine and it is contained inside a private network for testing. The developed platform worked as a proof-of-concept and proved that such project was possible and practical. The tools used in the making of the platform proved after some research to be suitable for developing the platform. The thesis identified some qualities of the used tools and what they are good for.

In this thesis also a brief look into different maritime data formats was included. The data formats and their standards proved to be complex and lengthy. Due to the complexity, the thesis focused on the more relevant parts of the standard and data formats. The relevant parts are such as reading the data and what information can be decoded from the data. Also, some maritime data use case examples are brought up. The maritime business could benefit greatly from analyzing data from vessels and using the results in practice for navigation and other key areas.

For future development for the platform there are many aspects to consider. For increased security, the authentication for WebSocket connections should be reviewed and made sure that unauthenticated personnel do not get access to any data. A whole separate front-end client could be beneficial for the future development. This would allow for in-depth visualization of the maritime data and more advanced features that Django’s template engine does not necessarily support. In addition, some known bugs should be fixed. The platform also needs more testing and obtaining some interested partners to test out the platform would be valuable. Since the platform is mainly a proof of concept,

one could also consider rebuilding the platform with more scalable technologies than Django framework or reworking the old code to be more efficient.

REFERENCES

Betke, K. (2000). The NMEA 0183 protocol. Retrieved November 2, 2020, from https://www.mnspoint.com/nmeadescription.pdf

Chesneau, B. (2020). Gunicorn documentation. Retrieved October 29, 2020, from https://readthedocs.org/projects/gunicorn-docs/downloads/pdf/latest/

Debian Project. (2019). Apt(8). Retrieved November 5, 2020 from https://manpages.debian.org/buster/apt/apt.8.en.html

Django Software Foundation. (2020). Django documentation. Retrieved October 29, 2020, from https://buildmedia.readthedocs.org/media/pdf/django/latest/django.pdf

F5, I. (2020). Nginx documentation. Retrieved October 29, 2020, from https://nginx.org/en/docs/

Gibson, C. (2020). Daphne. Retrieved October 29, 2020, from https://github.com/django/daphne

Godwin, A., & Django Software Foundation. (2020). Channels documentation . Retrieved October 29, 2020, from https://channels.readthedocs.io/en/latest/

Internet Engineering Task Force. (2011). RCF 6455 the WebSocket protocol. Retrieved November 6, 2020, from https://tools.ietf.org/html/rfc6455

Luft, L., Anderson, L. & Cassidy, F. (2002). NMEA 2000 A digital interface for the 21st century. Retrieved November 13, 2020, from https://www.nmea.org/Assets/nmea-2000-digital-interface-white-paper.pdf

Marine Digital GmbH. (2020). Big data in maritime: How a shipping company can effectively use data. Retrieved November 16, 2020, from https://marine-digital.com/article_bigdata_in_maritime

NMEA. (2020). National marine electronics association. Retrieved December 8, 2020, from https://www.nmea.org/

MarineTraffic. (2018). About the automatic identification system (AIS). Retrieved November 11, 2020, from https://help.marinetraffic.com/hc/en-us/articles/203990998-What-is-the-significance-of-the-AIS-Navigational-Status-Values-

Maritec Solutions. (2020). AIS VDM/VDO decoder. Retrieved November 17, 2020, from https://www.maritec.co.za/tools/aisvdmvdodecoding/

MongoDB, I. (2020). The MongoDB 4.4 manual. Retrieved November 5, 2020, from https://docs.mongodb.com/manual/

National Marine Electronics Association. (2002). *NMEA 0183, standard for interfacing marine electronic devices: Version 3.01*
NMEA National Office.

Python Software Foundation. (2020). Virtual environments and packages. Retrieved October 29, 2020, from https://docs.python.org/3/tutorial/venv.html

Raymond, E. (2019). AIVDM/AIVDO protocol decoding. Retrieved October 17, 2020, from https://gpsd.gitlab.io/gpsd/AIVDM.html

Spitzer, S., Luft, L. & Morschhauser, D. (2009). NMEA 2000 past, present and future. Retrieved November 2, 2020, from https://www.nmea.org/Assets/20090423%20rtcm%20white%20paper%20nmea%202000.pdf

Supervisor Developers. (2018). Supervisor documentation. Retrieved October 29, 2020, from https://readthedocs.org/projects/supervisor/downloads/pdf/latest/

U.S. Coast Guard Navigation Center. (2020). Automatic identification system overview. Retrieved October, 29 2020, from https://www.navcen.uscg.gov/?pageName=AISmain

USCG. (2020). USCG AIS encoding guide. Retrieved November 2, 2020, from https://www.navcen.uscg.gov/pdf/AIS/AISGuide.pdf