BT5110 Data Management & Warehousing
AIS Data Warehouse Project
Group 28

Group members:
Alex Chen Chen (A0280580E)
Laura Ngoc Ha Do (A0280548X)
Lim Ciwen Brendan (A0216513N)
Vikram Sharma (A0280540M)
Wong Cheuk Wah (A0280543H)

Outline of this report

1. Design phase

1.1. Use case for AIS-based data warehouse

The Automatic Identification System (AIS) is a tracking system for ships which is used by e.g. port and harbour authorities to improve the safety, compliance and efficiency of marine traffic. Participating ships frequently transmit details about their vessel, position, speed, time, destination etc., enabling authorities to monitor and intervene in maritime activities in real-time.

Considering further use cases, AIS data can be valuable for supply chain matters or port operations: Supply chain stakeholders can leverage AIS data to track cargo shipments to derive more accurate ETA (estimated time of arrival) and allow them to adjust consecutive supply chain steps if necessary. Other events such as logistic disruption, shipment delays and additional costs due to stock shortages or longer storage time etc. can be estimated, minimised or even mitigated. In terms of port operations, AIS data can be used to gain an overview of historic and current port operations, optimise the arrival and departure schedule of vessels to fully leverage the port's capacity, and understand how external factors such as weather and sea conditions can affect vessels' travel, and consequently, their port arrival or departure time.

Due to publicly available data, we have chosen to build a minimum viable product (MVP) of an AIS-based data warehouse for a port operations manager to demonstrate the value of one. The MVP runs on September 2023 AIS data for a part of Singapore waters and was retrieved from Data@Liánchéng's platform. Further web-scraped data on weather, sea conditions and vessel details have been added to enable more insightful analysis.

1.2. Design of MVP dashboard and queries

The MVP is designed for the port operations manager to get a general understanding of vessel traffic in the Singapore waters and conduct analysis on port operations and utilisation. For ease of use and fast adoption, the interface of the data warehouse is intended to be an interactive dashboard displaying several insightful graphs and a map.

As we assume that the manager's objective is to optimise port operations and utilisation, which we measure by the number of vessels arriving to and departing from the port per day, we have defined the interface of the dashboard and the queries based on insights that will be helpful from the user's perspective. Hence, our user can look and interact with the dashboard regularly, e.g. once a week, to understand what happened during the week, identify any trends or correlation as a starting point for root cause analysis.

Hence, we designed the UI (user interface) and content of the dashboard with the port manager's perspective, pain points and needs in mind. Ultimately, the MVP dashboard is divided into three sections.
The first section consists of static graphs which provide a general understanding of vessel traffic in the month of September. The graphs and queries for this static section are as follows:

1. In which countries are the vessels licensed that were present in the Singaporean territorial (port and waters)?
2. How many ships were headed to which destinations if they were departing from Singapore?
3. How many ships of which vessel type were present in Singapore territorial waters and ports?
4. On average, how many ships were in the Singapore port in the top 10% busiest days vs. top 25% vs. the mid 50% vs the bottom 25%?

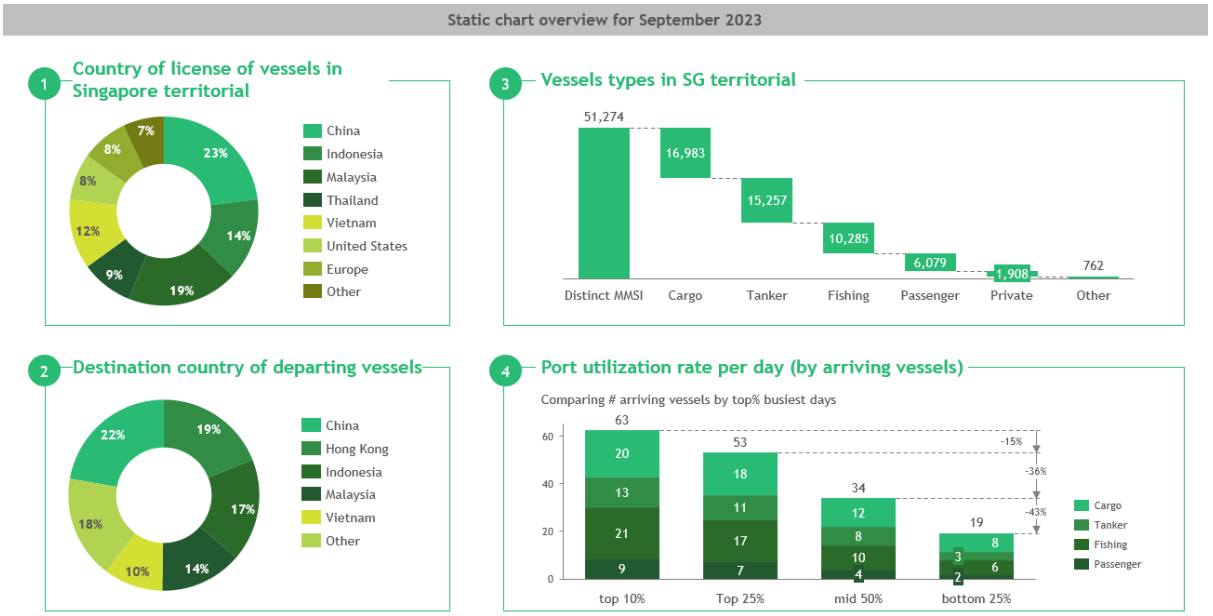

Figure 1: Dashboard section 1 User Interface (UI) design

In the second section, the port operations manager can deep-dive into specific time periods, vessel types, origin countries and the effect of weather and sea conditions. To do so, interaction buttons are used to customise the graphs. These interactive graphs are based on the following queries:

5. How many distinct vessels of which vessel type are present in Singapore territory each day?
6. For each day, how many vessels of which type are in the port? And compared to the average number of ships during the top 10% busiest days (benchmark), what is the capacity utilisation difference between that specific day and the benchmark?
7. What are the weather or sea conditions on each day?

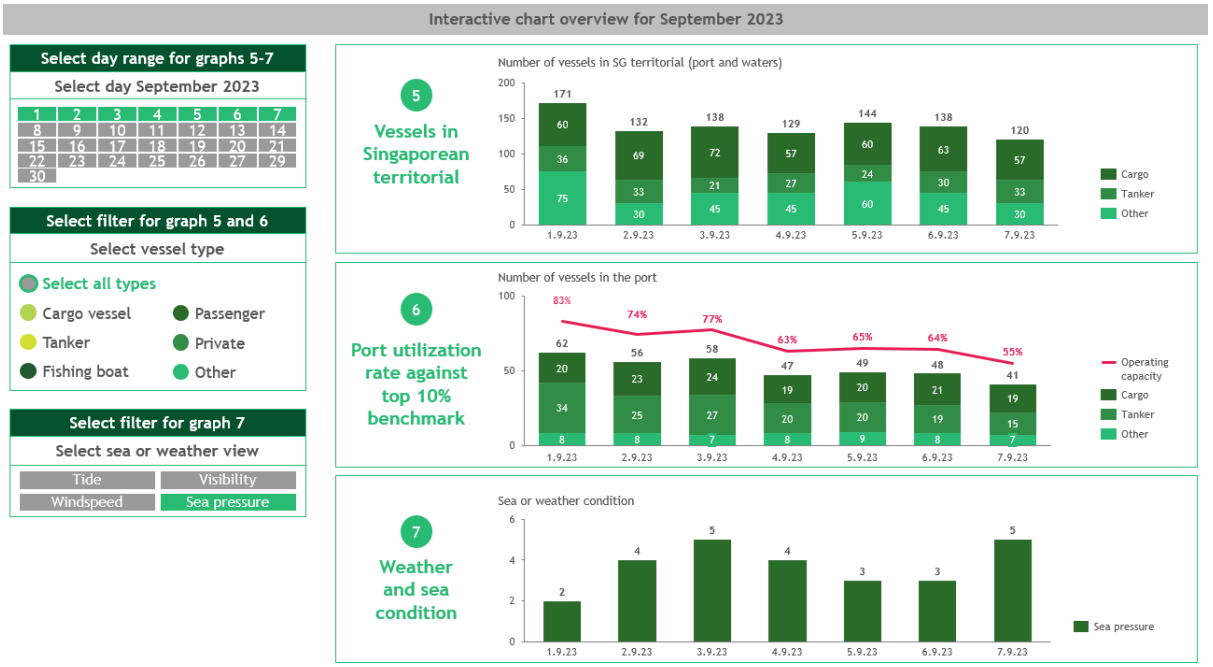

Figure 2: Dashboard section 2 User Interface (UI) and Interaction design

The third and last section consists of a map of the Singapore water area. This map view enables the user to gain a visual overview of vessels and their position at a selected point in time. By default, the map will show the positions of 20 vessels on 2023-09-01 at 12:00 pm. Starting from this position, the user can use the interaction buttons to choose date and time, country of origin and vessel type to display on the map. This part of the dashboard was designed based on the following interactive queries:

8. On a specific day at a specific time, where are all cargo vessels located?
9. What are the positions of vessels licensed from Malaysia at a specific date and time?

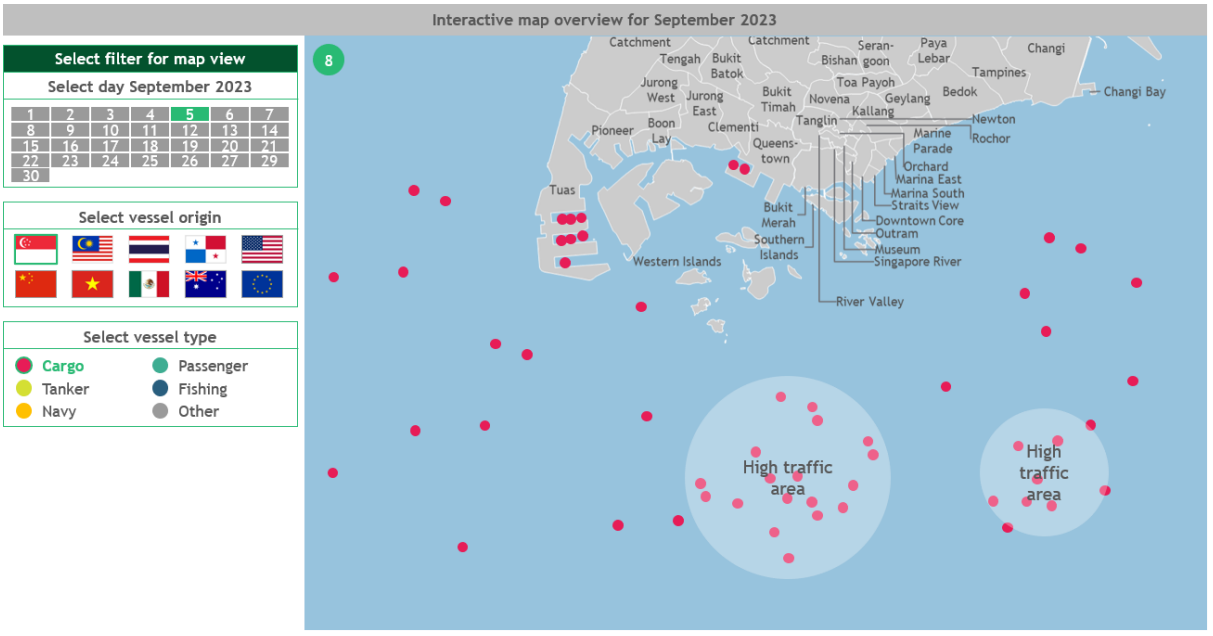

Figure 3: Dashboard section 3 User Interface (UI) and map interaction design

In summary, all three sections of the dashboard will provide the port operation manager with an sufficient overview of maritime traffic and port capacity utilisation to enable them to optimise port activities based on historic and current analyses.

2. Development phase

2.1. Data collection

2.1.1. Data collection: AIS extraction

The monthly data for September [https://data.liancheng.science/ais_logs.html] is far too huge to be opened in our machines, so we use various text editors to view the JSON. EMACS was used to view the data and after performing an incremental search we see that we have around 25 million documents.
Approaches used:

- We first tried a JSON parser to parse the file line by line to convert this into a CSV.
- We were unable to save the state and thus running it in batches was infeasible.
- We tried using JQ to convert this but were unsuccessful due to the uneven nature of the data.
- We tried replacing {,[and other elements so it looks like a CSV in plaintext which we can then import but due to the huge size this was not feasible in most regex based editors with the time and memory constraints.
- The approach that finally worked was querying for data in MongoDB and then extracting these results to CSV. We then imported this CSV file into our Postgresql.

2.1.2. External data scraping

We used BeautifulSoup as our main WebScraping library, however, the individual scraping strategies for each website were different. What is common is that all read data is converted into a Dataframe and then into a CSV. The strategies are as follows:

- AIS Ship Types
- https://api.vtexplorer.com/docs/ref-aistypes.html
    - The table of values is stored in a table that is found inside the container class and we iterate over each tr to get each value
- Country Codes
- https://documentation.spire.com/ais-fundamentals/vessel-flag-codes/
    - The first 3 digits of the MMSI are the MID codes which are none other than country codes that signify the country where the MMSI of the ship was issued. There are multiple tables on this page and thus to locate the correct table we identify the table id and then scrape the data into a dataframe
- VesselFinder to get ship dimensions
- https://www.vesselfinder.com/vessels/details/9382528
    - There are little to no free websites that offer ship dimensions in a pattern that can be fed into a web scraping algorithm. The link above has a pattern however in the address of the webpage. The sample link above finds a ship with the MMSI : 9382528. Each ship can be navigated to on this website

using either the MMSI or the IMO number at the end of the link. This details page provides a lot of valuable information including the size of the ship and the various other dimensions (however some information is behind a Paywall).
    - Each page has multiple tables and we read the one with dimensions with its respective table id. After getting a list of MMSI that are valid (i.e. follow the standard convention). We iterate through each link with different MMSI values at the end of the link.
    - Failsafes
        - As our data requests are large, we added a time interval of 3 seconds between each read
        - For MMSI that is not detected we add an error handling mechanism for 404 codes and we skip past those
        - For ships that do not have any tables to read we skip those links.
        - Data in the table is relative to one ship and so we transpose the table to get rows corresponding to each ship
    - Due to data inconsistencies and paywalls we were unable to scrape enough information to integrate this with our database.(Less than 1% matching ship MMSI found)
- Tide Information
- https://www.citipedia.info/en/tides/singapore/sembawang_singapore/m/september
    - Tide data is in a table and thus it is read the same as country codes
- Weather information
- https://www.visualcrossing.com/weather/weather-data-services#
    - Weather information had hourly data on fields like temperature, humidity, precipitation wind direction and cloud cover.
    - Data was in a table and downloaded as a CSV file
- Navigation status
- https://documentation.spire.com/ais-fundamentals/how-to-interpret-navigational-status-codes/
    - Navigation status provided the code and description of the different situations a ship can be under, such as “Underway using engine” or “Anchored”
    - Data was in a table so it was read by referencing the table ID

2.2. Data exploration and assumptions

From the data in September, the majority of the messages are found to be type 1, 2 and 3, which refer to position messages from class A vessels. This information mainly reports the navigation status, longitude and latitude coordinates and ship identifiers like MMSI . The next most common type of message is type 5 messages, which reports static information, like ship dimensions, callsign and name. The last and smallest type of messages are type 18 and 19 which are mainly used for class B vessels.

Data exploration reveals that in some scenarios, different ships (even with different dimensions) can share the same MMSI, IMO or ship name. Data inconsistencies can either arise from dynamic changes, such as the change of ownership of a ship which necessitates a change in MMSI, or a reassignment of MMSI to a different physical entity which necessitates a change in IMO. It is also possible for fraudulent or inaccurate information to

be present as individual ships can choose what to broadcast. While these cases are possible in a realistic scenario, we choose to omit these cases and assume that ship ownership etc. remains static, as we do not have information otherwise to verify changes in MMSI , IMO or call sign as well as when the changes occur.

To deal with this data inconsistency, we choose to solely rely on MMSI as a ship identifier, and mainly use distinct MMSIs in the count of ships, as it is consistent information that resides in all messages regardless of type. This overall gives us a deflated number of ships than actual (where multiple ships are counted as once), but due to the localised scope of this project off the south-western of Singapore, we believe that it is reasonable to assume that changes in ship details within the short span of a month, in a small area is negligible.

Further data exploration reveals that the majority of the ships have a reported destination of Singapore, which is not surprising given that the information originates from a location very close to Singapore.

Next, in order to classify a ship as having entered a port, we researched the location of the major container terminals in Singapore where container ships engage in loading and unloading. Given the coordinates of the port, we demarcate a square region around the coordinate of around +/- 0.01° (approximately +/- 1.1km). If a ship sends its location within the 2km by 2km grid, we define that as having entered a port. This process is repeated for the following ports. For the purpose of queries, we do not distinguish within ports but look at the number of ships in all the ports cumulatively.

Port	Longitude	Latitude
Tanjor Pagar	103.8480° E +/- 0.01°	1.2607° N +/- 0.01°
Keppel Harbour	103.8225° E +/- 0.01°	1.2612° N +/- 0.01°
Keppel Terminal	103.8380° E +/- 0.01°	1.2690° N +/- 0.01°
Marina at Keppel Bay	103.8347° E +/- 0.01	1.2664° N +/- 0.01°
Brani	103.8347° E +/- 0.01	1.2664° N +/- 0.01°
Pasir Panjang Terminal 1	103.7652° E +/- 0.01°	1.2839° N +/- 0.01°
Pasir Panjang Terminal 2	103.7800° E +/- 0.01°	1.2644° N +/- 0.01°
Pasir Panjang Terminal 3	103.7848° E +/- 0.01°	1.2749° N +/- 0.01°
Pasir Panjang Terminal 4	103.7896° E +/- 0.01°	1.2754° N +/- 0.01°
Pasir Panjang Terminal 5	103.7944° E +/- 0.01°	1.2759° N +/- 0.01°
Pasir Panjang Terminal 6	103.7992° E +/- 0.01°	1.2764° N +/- 0.01°
Tuas Port	103.6119° E +/- 0.01°	1.2443° N +/- 0.01°
Jurong Port	103.7172° E +/- 0.01	1.3098° N +/-0.01

Table 1: List of ports and coordinates demarcating port area

2.3. Data processing: Data transformation and cleaning

Firstly, the AIS ID from each message is dropped as it provides no useful information. Next, any invalid MMSI values were removed. MMSI is defined to be a 9 digit number; to process this, any values smaller than 9 digits were removed. As the data already existed in integer form, no further processing was required. Data was then split into the following 3 categories and further cleaned separately: type 1 to 3, type 5 and type 18 to 19. Only fields which were not null (that exist in the message) were extracted to reduce the size of the data.

For type 1 to 3 and type 18 to 19 data, no further processing was done other than to verify data types which mainly existed in integers or floats. Timestamp data was retained in datetime format for future processing. For type 5 data, the destination field for type 5 messages was cleaned by matching country codes or aliases for country names and manually converted to the actual country name. For some rows, the destination was reported as an exact location, such as “PEBGA” which refers to Eastern Boarding Ground “A” in Singapore. In such cases, port locations were searched to determine their home country and assigned as such. This was necessary due to the very varied data present in the column, which seems to be prone to data entry errors. Type 5 data was then normalised and split into fact and dimension tables. The fact table would contain the dynamic information such as timestamp, draught and destination, while the dimension table contained static information on the ship, like its dimensions, type, name, IMO and call sign. This process minimises any repetition of information and keeps the schema as close to Boyce Codd Normal Form (BCNF) as possible.

After all cleaning, the data retained had the following size:

Data/fact table	**Type**	**Rows**	**Columns**
Class A Position	1,2,3	23,223,013	10
Class A Static	5	1,417,136	4 (after normalisation)
Class B Position	18,19	2,373,177	7

Table 2: Data and fact table sizes

2.4. Schema definition: Fact and Dimension tables

2.4.1 Definitions

Schemas are utilised in data warehousing to organise data efficiently by separating fact and dimension tables. The fact tables provide transactional data such as the ship’s destination and time while the dimension tables each provide supporting additional static information about each transaction. Separating the fact table and dimension table through a schema allows optimised data storage and allows data analysts to query specifically for the information needed.

A star schema is a schema where the dimension tables have primary keys that are referenced by foreign keys in the fact tables. As the fact tables are generally placed in the middle and branch out to reference the dimension tables, a resemblance of a star shape can

be formed. A snowflake schema follows the concept of the star schema, but some dimension tables might contain foreign keys that reference other dimension tables, allowing relationships between dimension tables. The granularity of the data can be obtained from the detail and specifications in the dimension tables, as they obtain the finest grain (most granular data) in the whole dataset.

2.4.2 AIS Dimension and Fact Tables

The fact tables are composed purely using data from the September 2023 AIS data retrieved from Data@Liánchéng's platform. They consist of three tables:

- Type 1,2,& 3 messages: status, speed, longitude, latitude, and other dynamic data
- Type 5 messages: time, draught, destination, and MID code
- Type 18 & 19 messages: speed, longitude, latitude, and other dynamic data

It is worth noting that type 5 messages in the fact table only include the data that is dynamic from the original type 5 messages in the raw AIS data file, while the static information is included in a separate dimension table called 'ship_dimension'. The rest of the dimension tables used in our schema were web scrapped from the sites mentioned earlier.
The dimensions tables consist of the following:

- Ship Dimension: IMO, callsign, shiptype #, and other static specifications of the ship
- Shipcode: Identifies the vessel type (tanker, cargo, etc..) based on the shiptype #
- Weather: Wind speed, visibility and Sea Level Pressure on each day in September
- Country Code: Identifies the vessel flag country based on the Maritime Identification Digits (MID), extracted from the first 3 digits of the MMSI.
- Wave Tide: wave tide height and high/low binary based on different times of the day

2.4.3 AIS Schema

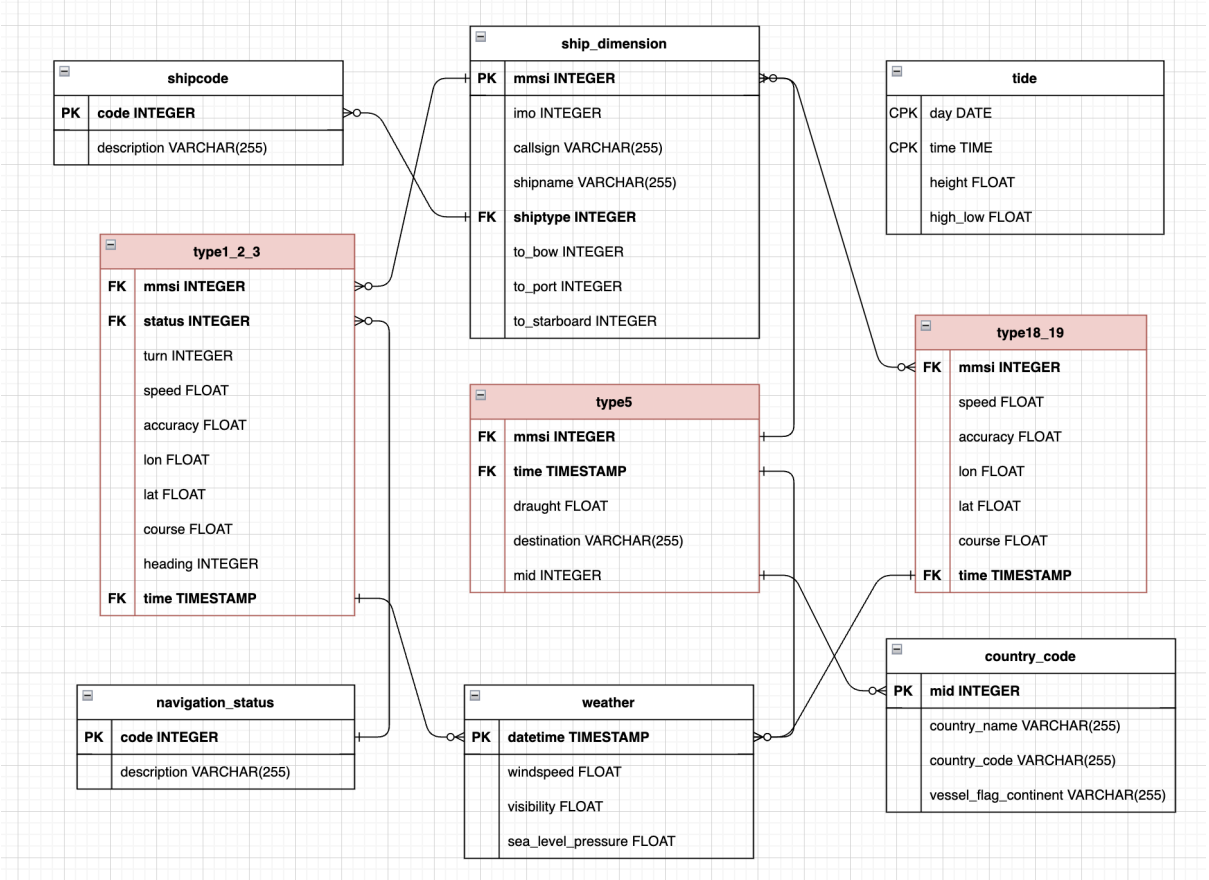

Figure 4. Data warehouse schema

In the schema shown above, the **bold** rows represent the primary keys in the tables and how they are referenced by foreign keys in the other tables, mostly by the fact tables. Although the schema closely resembles a star schema, there is a relationship between two dimension tables (ship dimension and ship code). The reasoning behind this extra branch is that shiptype integer (1-99) and the corresponding description (tanker, cargo, passenger, etc..) are both static messages, which requires the shiptype integer column from the ship dimension table to get information about the vessel's purpose. This creates a star schema with the exception of one single extra branch more characteristic of a snowflake schema.

The relationships are as follows:

- Type 1,2,& 3 messages use MMSI (foreign key) to reference the ship dimension MMSI (primary key)
- Type 5 messages use MMSI (foreign key) to reference the ship dimension MMSI (primary key)
- Type 18 & 19 messages use MMSI (foreign key) to reference the ship dimension MMSI (primary key)
- Type 5 messages use Maritime Identification Digits (MID) (foreign key) to reference the country code MID (primary key)
- Type 5 messages
- Ship Dimension table uses shiptype (foreign key) to reference the shipcode table's code column (primary key).

- All three fact tables use time (foreign key) to reference the weather datetime (primary key.)

The only table with no relational connection is the Tide table, used to measure the wave heights at different times of each day in September. The reason it is a dimension table with no relationship is that it is simply used to graph out different data based on the day and comparing to the ranges of those times in the rest of the dataset. The table has a composite primary key of (day,time), as it needs two attributes to uniquely identify each record in the table.

The overall schema allows the user to query based on the type of message or a combination of different messages, and only utilising the dimension tables relevant to the individual needs. In a dataset with millions of messages, a schema reduces significantly the time and storage of the data as well as simplicity in visualisation of search results.

2.5. Dashboard development

The dashboard interface is built in Python with library packages namely, Dash, Plotly and Folium.

2.5.1. Web implementation and data loading

From the Python script, we first connect to the AIS data warehouse using Psycopg2, a PostgreSQL database adapter, which ensures the interaction between Python and PostgreSQL. With Psycopg2, we then can pass SQL queries from Python to the pgAdmin server and extract the result tables back to Python for further analysis and visualisation.

We have decided to use Dash to develop an interactive dashboard for our user. It is a python framework written based on Flask, Plotly.js and React, and its analyses' results are displayed on a local web browser. Hence, in our MVP data warehouse, we retrieve information from our connected PostgreSQL database, conduct analyses with SQL queries and Python libraries, and finally visualise our results and insights in a Dash interface.

Dash also enables our users to interact with the input components of the interface. This way, our port operation manager can customise the deep-dive graphs and map according to their interest and needs.

In our MVP dashboard, the dash application is deployed inside JupyterLab, in which an external URL is displayed as a hyperlink. By clicking it, we can open the dashboard in a new browser tab.

2.5.2. Data visualisation and interactive queries

Following our analyses in Python, we visualise trends and patterns with graphs using Plotly. In addition, we have created a map to display the positions of vessels using Folium based on their longitude and latitude data.

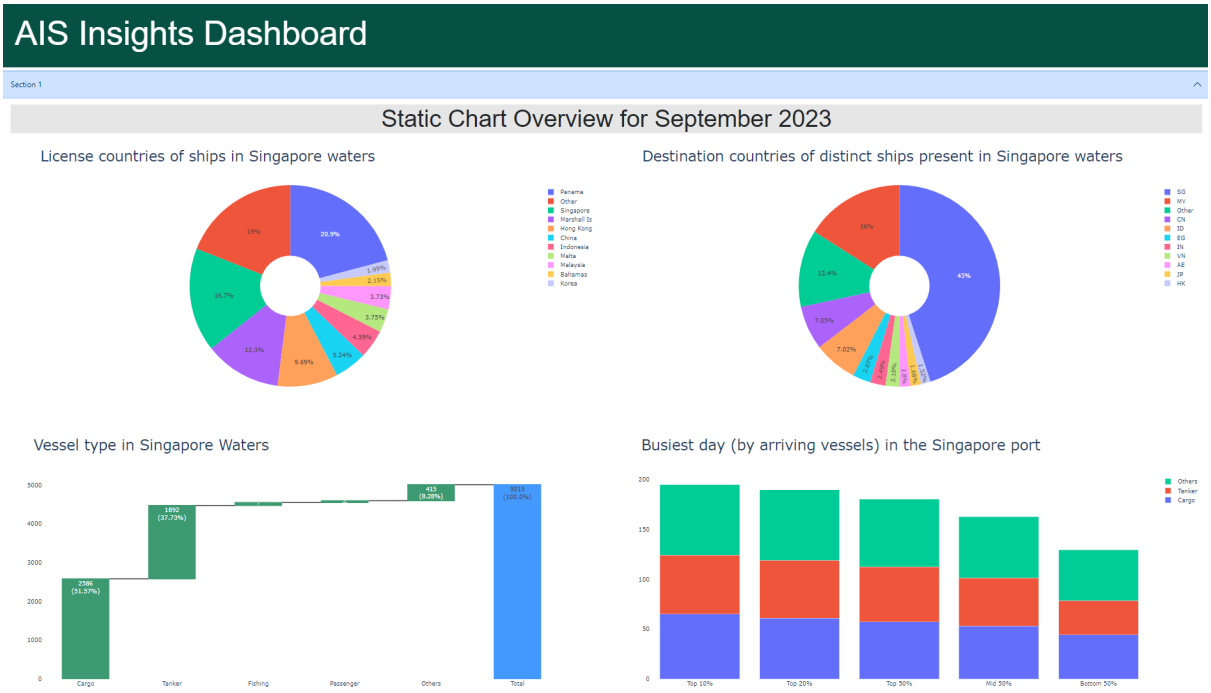

Figure 5: Interface of the Static Chart Overview

Figure 4 showcases section 1 of the dashboard with the static chart overview based on the 4 hard-coded queries. This section's purpose is to provide the user with a general understanding of the vessels in the Singaporean territory altogether, as well as a first glance on port utilisation over the whole month.

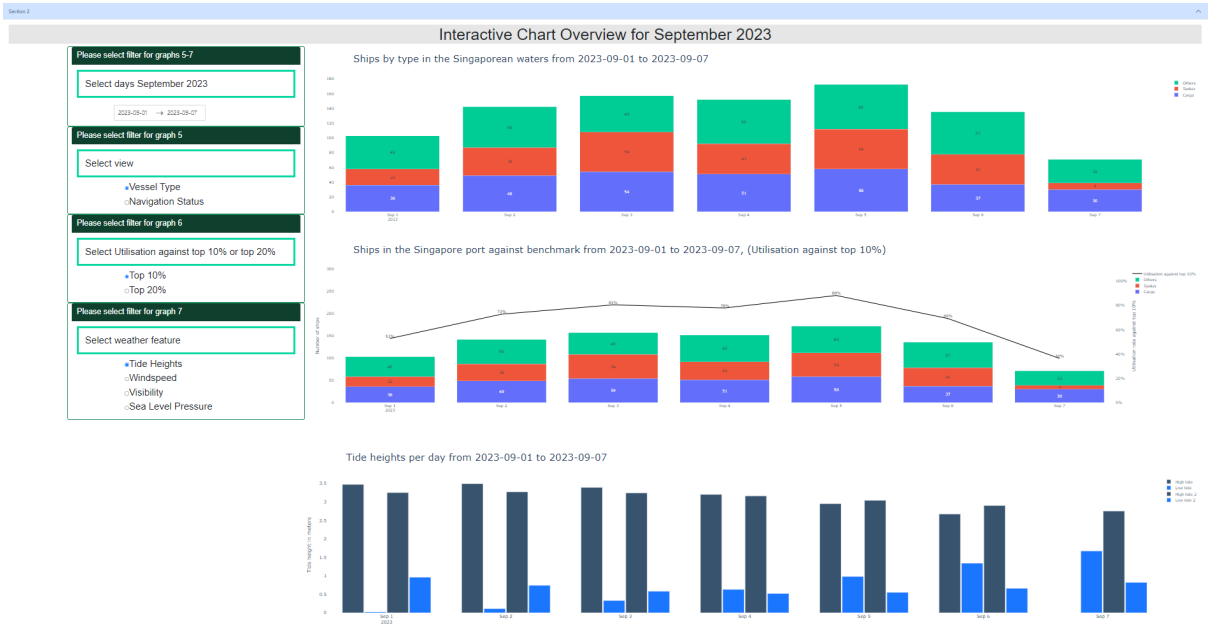

Figure 6: Interface of the interactive Chart Overview

To facilitate interactive trend analysis for the user, we incorporate interaction buttons in the dashboard's section 2, as shown in Figure 5. In the frontend interface, users can actively engage in the data exploration by selecting the specific time periods, vessel types, navigation status, origins, and weather or sea conditions. When they select a button that represents their option, the corresponding value will be passed to a callback decorator. The function that the decorator wraps will be triggered because of the change in its input property. By default, no filtering is applied in which all records will be displayed in the visuals. Upon the function's execution, the associated SQL query will be updated with a new filtering condition that extracts the relevant data. With the newly extracted dataframe, the figure will also change accordingly. It will be returned to the output property in the callback decorator. The updated output graph will then be passed to the component property it belongs to and displayed in the interface. Appendix B showcases a piece of code example about how the decorator triggers its corresponding callback function.

There are four interactive buttons and their component identification numbers (IDs) in Section 2 of the dashboard,

- Datepicker Range (ID: part2-left-DatePickerRange-input): It allows users to choose a range of days in the month of September, which is applied to all the interactive graphs
- View Selector (ID: part2-left-ViewRadioItems-input): Users can choose to view the number of ships in the Singapore waters by either vessel type or navigation status.
- Benchmark Selector (ID: part2-left-TopRadioItems-input): Users can compare the number of different types of vessels against the top 10 % of 20% busiest days in the port
- Weather Feature Selector (ID: part2-left-WeatherRadioItems-input): With options of tide heights, wind speed, visibility and sea level pressure, users can select the weather condition based on their exploration interest

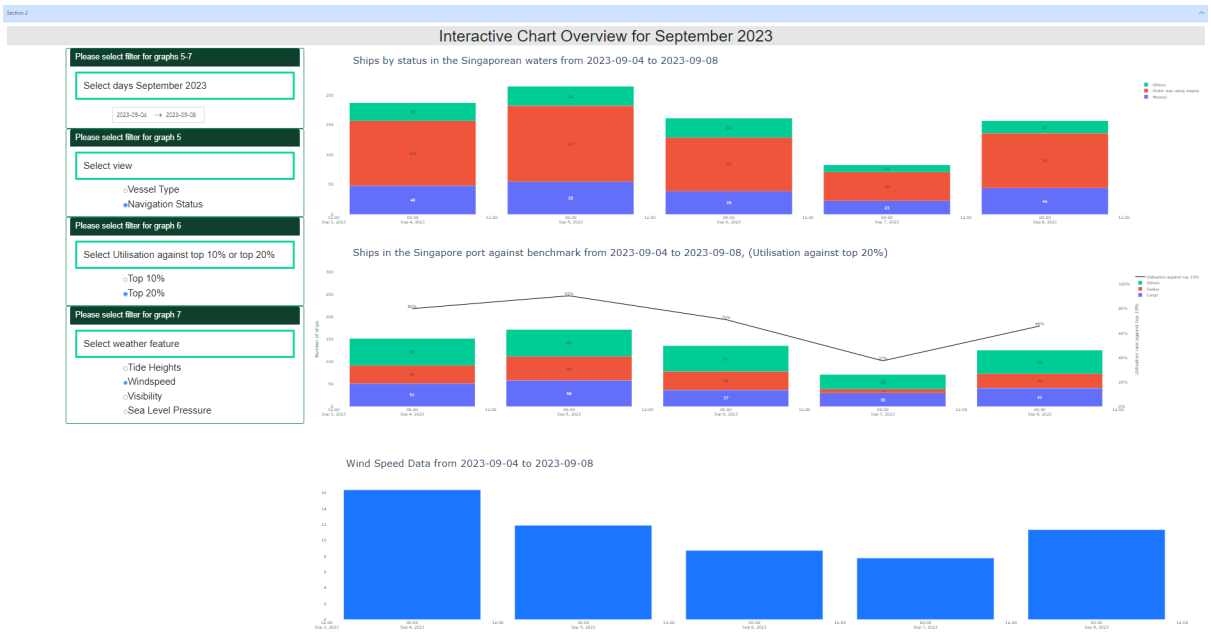

Figure 7: Demonstration of Interaction

Figure 6 illustrates the display result if the user is interested in comparing the port utilisation rate per day from September 4th to 8th, 2023, against the top 20% busiest days of the month. As displayed, the user will find that port utilisation is especially low on September 7th, with a utilisation rate of only 17% if benchmarked against the top 20% busiest port days in September. Comparing this result to the above and below graph, this low port utilisation rate can be explained by two factors: (1) there were in general fewer vessels in the Singaporean territorial, and looking at wind speed, (2) this result seems to be correlated to lower wind speed.

This insight can be helpful to the user as it enables further analysis into whether and why wind speed might affect vessel presence in the area and arrival to port. If found positive, the port manager can use current wind speed data to approximate port utilisation and increase or decrease the number of on-duty port staff, e.g. to save labour cost, or redirect labour efforts towards other activities that day.

Furthermore, this port utilisation rate analysis also enables the user to compare busy to non-busy days in order to evaluate what port-driven factors might be. This might help to make key decisions in how to market and cater port services to cargo and tanker vessels to increase the port utilisation in the future. Increasing port operations can increase the port's profitability, hence, these analyses and insights are assumed to be of highest value for a port manager. In contrast, without these graphs and insights from our dashboard, these root cause analyses cannot be conducted or even kick-started.

2.5.3. Interactive Map

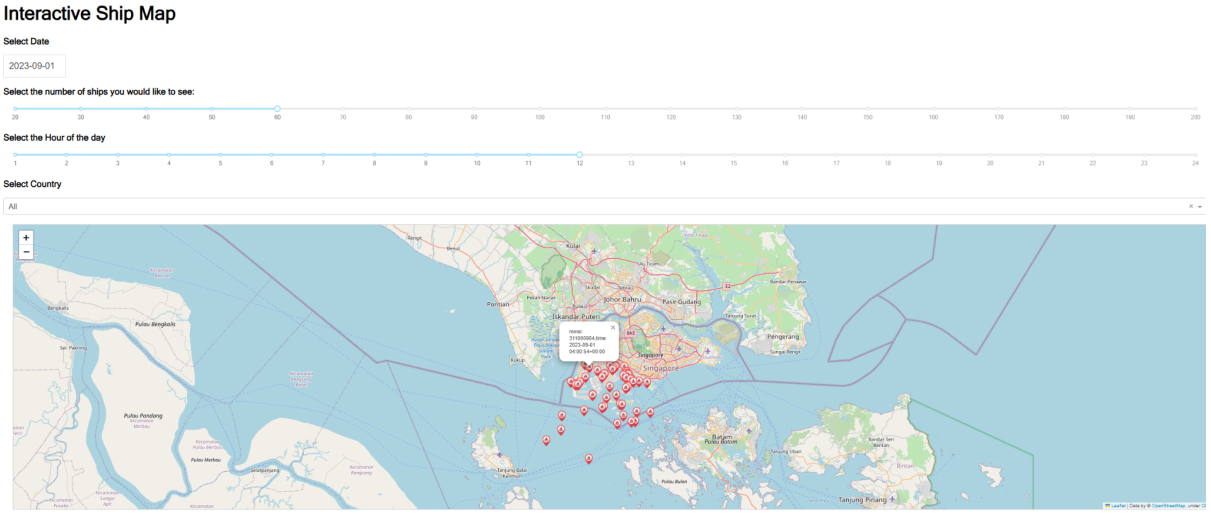

Figure 8: Interactive Map

The libraries used to create and then push the html object onto our Dash WebApp are folium, tempfile and dcc.html.

Interactive Buttons

- DatePicker (ID: date-picker): we have a datepicker that allows users to choose a date from the month of september
- The hour slider (ID: hour-slider): The slider object starts at 1 and goes up to 24 so that users can choose the hour of the day

- The number of ships (ID: ship-limit-slider): The slider object starts at 20 and goes up to 200 with a step increment of 10 ships
- Origin Country Selector (ID:country-dropdown): Dropdown with a list of the top 113 countries which the user has an option to choose from. The 113 countries are the ones with the most ships and this data is stored as a dataframe via a CSV which is then passed as a list to our dropdown values. (Figure 9)

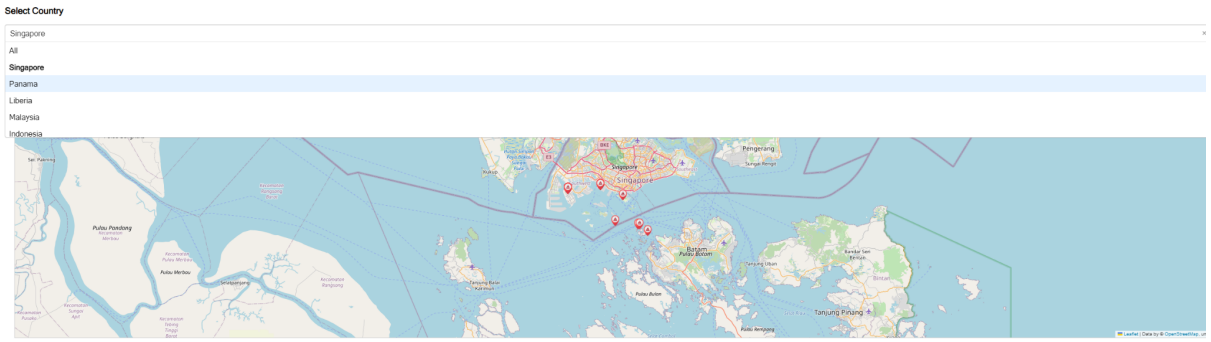

Figure 9: Country Selector Dropdown

If the value of any of the above changes then the callback function is called with the update map function which updates the query and thus our data.

Workflow

- We first use folium to create a Folium Map object by querying from our local postgres database via Psycopg2. The query we used is a join of table type1_2_3 and type 5 to get the MMSI, latitude and longitude
- The map uses a default date of 1/09/2023 at 12:00 Pm for viewing 20 Ships.
- We then get a populated map which we then convert into an html object that is stored in a tempfile and passed onto Dash
- The map then dynamically passes values into our multiline string query to change it accordingly
- The Update function changes the state of the Dash webpage to now reflect the updated values

Extra features

- We use custom markers by importing a ship png into our environment.
- Dynamically changing the icon size as the user increases the number of ships to display, so that the map is not cluttered
- Clicking on the markers shows you the exact timestamp as well as the MMSI of that ship

Queries and iterations

- *Query for all countries:- query1 = """*
  SELECT t1.MMSI, t1.time, t1.lat, t1.lon
  FROM type1_2_3 t1
  WHERE t1.MMSI IN (SELECT DISTINCT MMSI FROM type5)
  AND DATE(t1.time) = %s
  AND EXTRACT(HOUR FROM t1.time) = %s
  AND t1.MMSI IS NOT NULL
  AND t1.lon IS NOT NULL

AND t1.lat IS NOT NULL
LIMIT %s;
"""

- *Query for selected countries:- query1 = """*
  SELECT t1.MMSI, t1.time, t1.lat, t1.lon, c.country_name
  FROM type1_2_3 t1
  LEFT JOIN country_code c ON t1.MMSI / 1000000 = c.mid
  WHERE t1.MMSI IN (SELECT DISTINCT MMSI FROM type5)
  AND DATE(t1.time) = %s
  AND c.country_name = %s
  AND EXTRACT(HOUR FROM t1.time) = %s
  AND t1.MMSI IS NOT NULL
  AND t1.lon IS NOT NULL
  AND t1.lat IS NOT NULL
  LIMIT %s;
  """
- The parameters are updated as the interactive buttons change their state and so these new parameters are passed into the query in place of '%s' in our update map function, thus refreshing the data
- The mid is the first 3 digits and thus we perform the necessary division for this join.
- The 2 queries change based on if the user selects "All" or something else in our list of top 113 countries. If a selection is made then query1 gets overwritten to the one with the added country_name filter and vice versa

The interactive components are summarised in the callback graph below:

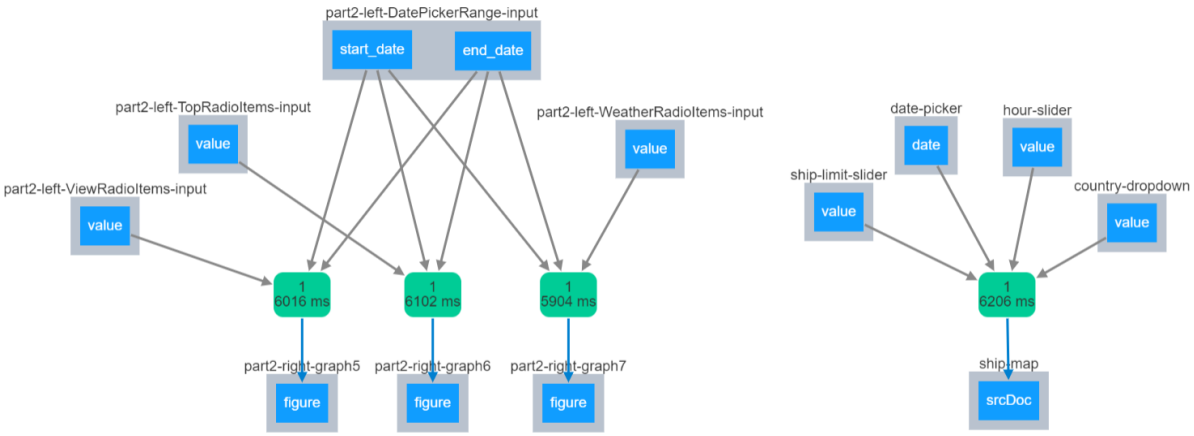

Figure 10: Callback Graph

The components at the top represent the interactive buttons that require users' inputs, denoted by their IDs and input properties. The green oblongs record the number of clicks and the execution duration for each interaction. After the callback functions are executed, the outputs will be passed to the graphs they belong to according to their component IDs.

As shown in Figures 5 and 6, there are 4 interactive visualisations in the dashboard,

- part2-right-graph5 refers to the graph "Ship by type in the Singaporean waters"
- part2-right-graph6 refers to the graph "Ships in the Singapore port against benchmark"
- part2-right-graph7 refers to the graph "Weather Condition per day"
- ship-map refers to the interactive map

As such, the graphs and map will be updated accordingly when any buttons are pressed.

2.5.4. Ship tracking

We also have an added feature of tracking ships if the user has a list of MMSI in a CSV. We first read the csv and store the MMSI in a list and then pass this into our query as follows:

```
query1 = """
        SELECT t1.MMSI, t1.time, t1.lat, t1.lon
        FROM type1_2_3 t1
        WHERE t1.MMSI IN %s
        AND t1.MMSI IS NOT NULL
        AND t1.lon IS NOT NULL
        AND t1.lat IS NOT NULL
    """
location_df = pd.read_sql_query(query1, conn, params=(tuple(lat_lon_df["MMSI"]),))
```

If we get back results then the ship is shown on the map.

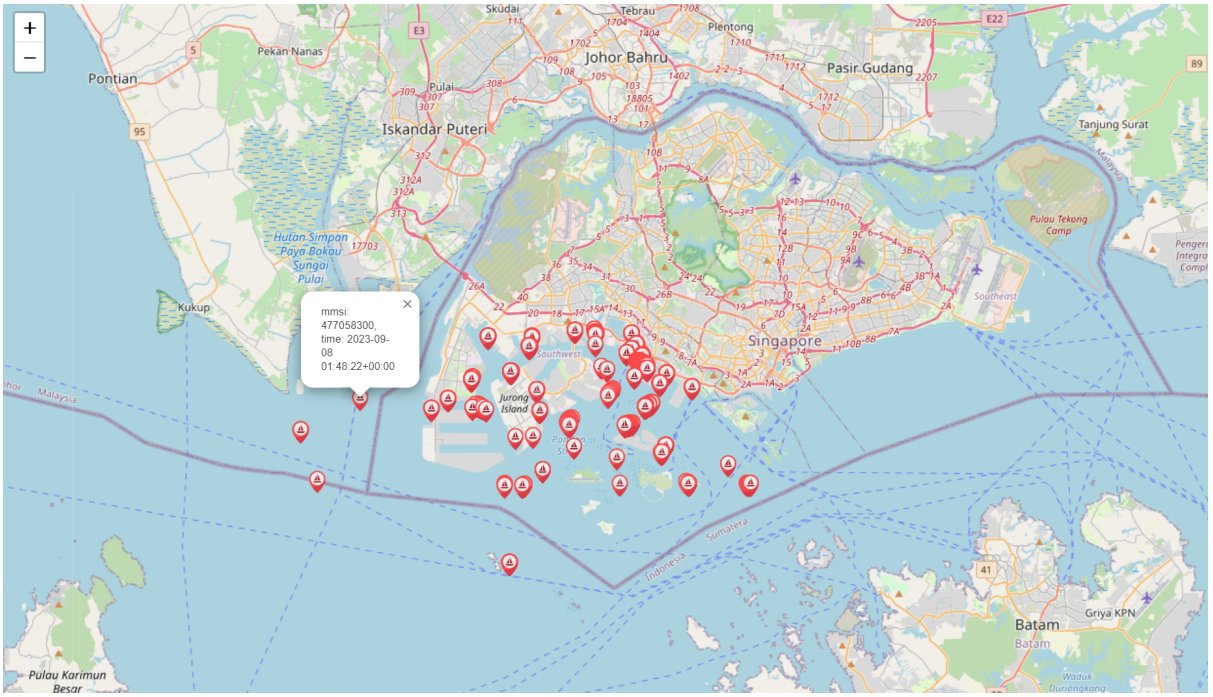

Figure 11: Map Tracking from CSV file

2.5.5. Query formulation

From the data we examined, we are aware that the list of MMSI in type 1 to 3 messages differs from the list of MMSI in type 5 messages. This means that not all ships that sent type 5, also sent type 1, vice versa. However our queries do require information from both type 1 (longitude and latitude) and type 5 (ship type) messages. Hence in order to retain a uniform

benchmark between queries, such that analysis between queries remains meaningful, we only retain information from messages that had the same MMSI in both the type 1 to 3 and type 5 message fact tables.Hence we only look at the intersection between position and static messages.

Categorising ships “in port” was additionally implemented with the knowledge that we could not achieve 100% accuracy. To address this, we used the coordinates in Table 1, which we believe provides a reasonable estimate for the boundaries of the ports.

Lastly, in order to deal with the large variety of ship types, CASE WHEN was used to segregate between the types of interest (cargo and tanker) from the others. An example of this query can be found in Appendix A which showcases both the CASE WHEN syntax and the method used for port coordinate demarcation.

2.5.6. Query optimisation

Due to the large nature of data with rows in the millions, queries had to be written with performance measures in mind. Initially when base cross joins were utilised on each message fact table, it yielded queries that would run for more than 5 minutes at a time which is not ideal for on-the-go ad-hoc OLAP analysis. Hence the queries were reviewed and only DISTINCT information was extracted from each table in order to reduce the query join sizes, from initial table size of 20+ million, down to the thousands. Only information that is required for each query was extracted, such as MMSI, time and type, in order to be able to perform the query. Using this method on both fact 1 to 3 and fact 5 tables, yielded a resulting query duration of below 5 seconds. This duration is deemed reasonable for daily use for ad-hoc queries that remain compatible with our web application. Examples of the use of the subqueries for distinct results can be found in Appendix A.

3. Dashboard demonstration

Figure 12. Roles for product demonstration

In our dashboard presentation, we are re-enacting a product demonstration and sales pitch. The goal of this 10 minute pitch is to present and spark the interest of our client “Jurong Port” in buying the new AIS Dashboard from “AIS Insights LLC”.

To watch the 10 minute product demonstration, please refer to the video: [https://drive.google.com/drive/folders/1eiQ3S8Sih6qSpuEge_xriQFjK1K_f_wH?usp=drive_link].

For our entire Codebase, please refer to: https://drive.google.com/drive/folders/1G6ckc4urAMeHBmgVz_YN04zmXbaX0nan?usp=sharing

4. References

1. *Ais Fundamentals*. Spire Maritime Documentation. (2022, September 22). https://documentation.spire.com/ais-fundamentals/vessel-flag-codes/

2. *AIS messages*. Home. (n.d.). https://www.navcen.uscg.gov/ais-messages

3. *AIS ship types*. AIS Ship Types - AIS Data - VT Explorer. (n.d.). https://api.vtexplorer.com/docs/ref-aistypes.html

4. GeeksforGeeks. (2023, May 8). *Star schema in Data Warehouse Modeling*. GeeksforGeeks. https://www.geeksforgeeks.org/star-schema-in-data-warehouse-modeling/

5. Kyaagba, S. (2018, September 8). *Integrating folium with dash*. Medium. https://medium.com/@shachiakyaagba_41915/integrating-folium-with-dash-5338604e7c56

6. *Tide times and tide charts Sembawang*. Tide Times and Tide charts in September sembawang. (n.d.). https://www.citipedia.info/en/tides/singapore/sembawang_singapore/m/september

7. *Vesselfinder*. VesselFinder. (n.d.). https://www.vesselfinder.com/vessels/

8. Louart, M., Szkolnik, J., Boudraa, A., Lann, J. L., & Roy, F. L. (2023). Detection of AIS messages falsifications and spoofing by checking messages compliance with TDMA protocol. Digital Signal Processing, 136, 103983. https://doi.org/10.1016/j.dsp.2023.103983

5. Appendices

5.1. Appendix A: SQL Queries used

Query for ship origin country

```
SELECT c.country_name as country, count(DISTINCT t1.mmsi) as count
FROM country_code c,
(SELECT DISTINCT mmsi FROM type1_2_3) as t1,
(SELECT DISTINCT mmsi, MID FROM type5) as t5
WHERE t1.mmsi = t5.mmsi
AND c.MID = t5.MID
GROUP BY c.country_name
ORDER BY count DESC
```

	country character varying (255)	count bigint
1	Panama	1047
2	Singapore	837
3	Marshall Is	616
4	Hong Kong	486
5	China	263
6	Indonesia	230
7	Malta	188
8	Malaysia	187
9	Bahamas	108
10	Korea	100

Total rows: 68 of 68 Query complete 00:

Query for ship destination countries

```
SELECT c.country_name as country, count(DISTINCT t1.mmsi) as count
FROM country_code as c,
(SELECT DISTINCT mmsi FROM type1_2_3) as t1,
(SELECT DISTINCT mmsi, destination_country FROM type5) as t5
WHERE t1.mmsi = t5.mmsi
AND LOWER(t5.destination_country) = LOWER(c.country_name)
GROUP BY country
ORDER BY count DESC
```

	country character varying (255)	count bigint
1	Singapore	2604
2	Malaysia	924
3	China	408
4	Indonesia	406
5	Egypt	166
6	India	144
7	Vietnam	126
8	UAE	104
9	Japan	97
10	Korea	88

Total rows: 40 of 40 Query complete 00

Query for types of ship

```
SELECT CASE
WHEN code = 30 THEN 'Fishing'
WHEN code BETWEEN 60 AND 69 THEN 'Passenger'
WHEN code BETWEEN 70 AND 79 THEN 'Cargo'
WHEN code BETWEEN 80 AND 89 THEN 'Tanker'
ELSE 'Others' END AS bins, count(mmsi) as count
FROM (
SELECT DISTINCT t1.mmsi as mmsi, sc.code as code
FROM ship_dimension as s, shipcode as sc,
(SELECT DISTINCT mmsi FROM type1_2_3) as t1,
(SELECT DISTINCT mmsi FROM type5) as t5
WHERE t1.mmsi = s.mmsi
AND t1.mmsi = t5.mmsi
AND s.shiptype = sc.code
    ) t
GROUP BY bins
ORDER BY count DESC
```

	bins text	count bigint
1	Cargo	2586
2	Tanker	1892
3	Others	415
4	Fishing	75
5	Passenger	47
Total rows: 5 of 5		Query co

Query for ship in ports, by types by date

```
SELECT DATE(t1.time) as date, CASE WHEN sc.description LIKE '%Cargo%' THEN 'Cargo'
 WHEN sc.description LIKE '%Tanker%' THEN 'Tanker' ELSE 'Others' END as type, count(DISTINCT t1.mmsi) as count
FROM ship_dimension as s, shipcode as sc,
(SELECT DISTINCT mmsi, time, lon, lat FROM type1_2_3) as t1,
(SELECT DISTINCT mmsi FROM type5) as t5
WHERE t1.mmsi = s.mmsi
AND t1.mmsi = t5.mmsi
AND s.shiptype = sc.code
AND ((t1.lon BETWEEN 103.838022 AND 103.852221 -- tanjong pagar
AND t1.lat BETWEEN 1.257772 AND 1.269024) -- tanjong pagar
OR (t1.lon BETWEEN 103.8225-0.01 AND 103.8225+0.01 -- keppel harbour
AND t1.lat BETWEEN 1.2612-0.01 AND 1.2612+0.01) -- keppel harbour
OR (t1.lon BETWEEN 103.8380-0.01 AND 103.8380+0.01  -- keppel terminal
AND t1.lat BETWEEN 1.2690-0.01 AND 1.2690+0.01) -- keppel terminal
OR (t1.lon BETWEEN 103.8347-0.02 AND 103.8347+0.02 -- marina
AND t1.lat BETWEEN 1.2664-0.02 AND 1.2664+0.02) -- marina
OR (t1.lon BETWEEN 103.7752-0.01 AND 103.7752+0.01 -- ppt1
AND t1.lat BETWEEN 1.2739-0.01 AND 1.2739+0.01) -- ppt1
OR (t1.lon BETWEEN 103.7800-0.01 AND 103.7800+0.01 -- ppt2
AND t1.lat BETWEEN 1.2744-0.01 AND 1.2744+0.01) -- ppt2
OR (t1.lon BETWEEN 103.7848-0.01 AND 103.7848+0.01 -- ppt3
AND t1.lat BETWEEN 1.2749-0.01 AND 1.2749+0.01) -- ppt3
OR (t1.lon BETWEEN 103.7896-0.01 AND 103.7896+0.01 -- ppt4
AND t1.lat BETWEEN 1.2754-0.01 AND 1.2754+0.01) -- ppt4
OR (t1.lon BETWEEN 103.7944-0.01 AND 103.7944+0.01 -- ppt5
AND t1.lat BETWEEN 1.2759-0.01 AND 1.2759+0.01) -- ppt5
OR (t1.lon BETWEEN 103.7992-0.01 AND 103.7992+0.01 -- ppt6
AND t1.lat BETWEEN 1.2764-0.01 AND 1.2764+0.01) -- ppt6
OR (t1.lon BETWEEN 103.61191878972717-0.01 AND 103.61191878972717+0.01 -- tuas
AND t1.lat BETWEEN 1.244386225597597-0.01 AND 1.244386225597597+0.01)) -- tuas
GROUP BY date, type
ORDER BY date, type, count DESC
```

	date date	type text	count bigint
1	2023-09-01	Cargo	36
2	2023-09-01	Others	45
3	2023-09-01	Tanker	22
4	2023-09-02	Cargo	49
5	2023-09-02	Others	55
6	2023-09-02	Tanker	38
7	2023-09-03	Cargo	54
8	2023-09-03	Others	49
9	2023-09-03	Tanker	54
10	2023-09-04	Cargo	51

Total rows: 93 of 93 Query complete 00:

Query for ship types within date range

```
SELECT DATE(t1.time) as date, CASE WHEN sc.description LIKE '%Cargo%' THEN 'Cargo'
 WHEN sc.description LIKE '%Tanker%' THEN 'Tanker' ELSE 'Others' END as type, count(DISTINCT t1.mmsi) as count
FROM ship_dimension as s, shipcode as sc,
(SELECT DISTINCT mmsi, time, lon, lat FROM type1_2_3) as t1,
(SELECT DISTINCT mmsi FROM type5) as t5
WHERE t1.mmsi = s.mmsi
AND t1.mmsi = t5.mmsi AND s.shiptype = sc.code
AND DATE(t1.time) BETWEEN '1/9/23' and '3/9/23' -- sample input
AND ((t1.lon BETWEEN 103.838022 AND 103.852221 -- tanjong pagar
AND t1.lat BETWEEN 1.257772 AND 1.269024) -- tanjong pagar
OR (t1.lon BETWEEN 103.8225-0.01 AND 103.8225+0.01 -- keppel harbour
AND t1.lat BETWEEN 1.2612-0.01 AND 1.2612+0.01) -- keppel harbour
OR (t1.lon BETWEEN 103.8380-0.01 AND 103.8380+0.01  -- keppel terminal
AND t1.lat BETWEEN 1.2690-0.01 AND 1.2690+0.01) -- keppel terminal
OR (t1.lon BETWEEN 103.8347-0.02 AND 103.8347+0.02 -- marina
AND t1.lat BETWEEN 1.2664-0.02 AND 1.2664+0.02) -- marina
OR (t1.lon BETWEEN 103.7752-0.01 AND 103.7752+0.01 -- ppt1
AND t1.lat BETWEEN 1.2739-0.01 AND 1.2739+0.01) -- ppt1
OR (t1.lon BETWEEN 103.7800-0.01 AND 103.7800+0.01 -- ppt2
AND t1.lat BETWEEN 1.2744-0.01 AND 1.2744+0.01) -- ppt2
OR (t1.lon BETWEEN 103.7848-0.01 AND 103.7848+0.01 -- ppt3
AND t1.lat BETWEEN 1.2749-0.01 AND 1.2749+0.01) -- ppt3
OR (t1.lon BETWEEN 103.7896-0.01 AND 103.7896+0.01 -- ppt4
AND t1.lat BETWEEN 1.2754-0.01 AND 1.2754+0.01) -- ppt4
OR (t1.lon BETWEEN 103.7944-0.01 AND 103.7944+0.01 -- ppt5
AND t1.lat BETWEEN 1.2759-0.01 AND 1.2759+0.01) -- ppt5
OR (t1.lon BETWEEN 103.7992-0.01 AND 103.7992+0.01 -- ppt6
AND t1.lat BETWEEN 1.2764-0.01 AND 1.2764+0.01) -- ppt6
OR (t1.lon BETWEEN 103.61191878972717-0.01 AND 103.61191878972717+0.01 -- tuas
AND t1.lat BETWEEN 1.244386225597597-0.01 AND 1.244386225597597+0.01)) -- tuas
GROUP BY date, type
ORDER BY date, type, count DESC
```

	date date	type text	count bigint
1	2023-09-01	Cargo	36
2	2023-09-01	Others	45
3	2023-09-01	Tanker	22
4	2023-09-02	Cargo	49
5	2023-09-02	Others	55
6	2023-09-02	Tanker	38
7	2023-09-03	Cargo	54
8	2023-09-03	Others	49
9	2023-09-03	Tanker	54

Total rows: 9 of 9 Query complete 00:00:

Query for ship navigation status by date range

```
SELECT DATE(t1.time) as date, ns.description as status, count(DISTINCT t1.mmsi) as count
FROM ship_dimension as s, navigation as ns,
(SELECT DISTINCT mmsi, time, status, lat, lon FROM type1_2_3) as t1,
(SELECT DISTINCT mmsi FROM type5) as t5
WHERE t1.mmsi = s.mmsi AND t1.mmsi = t5.mmsi
AND ns.code = t1.status
AND DATE(t1.time) BETWEEN '1/9/23' and '3/9/23' -- sample input
AND ((t1.lon BETWEEN 103.838022 AND 103.852221 -- tanjong pagar
AND t1.lat BETWEEN 1.257772 AND 1.269024) -- tanjong pagar
OR (t1.lon BETWEEN 103.8225-0.01 AND 103.8225+0.01 -- keppel harbour
AND t1.lat BETWEEN 1.2612-0.01 AND 1.2612+0.01) -- keppel harbour
OR (t1.lon BETWEEN 103.8380-0.01 AND 103.8380+0.01  -- keppel terminal
AND t1.lat BETWEEN 1.2690-0.01 AND 1.2690+0.01) -- keppel terminal
OR (t1.lon BETWEEN 103.8347-0.02 AND 103.8347+0.02 -- marina
AND t1.lat BETWEEN 1.2664-0.02 AND 1.2664+0.02) -- marina
OR (t1.lon BETWEEN 103.7752-0.01 AND 103.7752+0.01 -- ppt1
AND t1.lat BETWEEN 1.2739-0.01 AND 1.2739+0.01) -- ppt1
OR (t1.lon BETWEEN 103.7800-0.01 AND 103.7800+0.01 -- ppt2
AND t1.lat BETWEEN 1.2744-0.01 AND 1.2744+0.01) -- ppt2
OR (t1.lon BETWEEN 103.7848-0.01 AND 103.7848+0.01 -- ppt3
AND t1.lat BETWEEN 1.2749-0.01 AND 1.2749+0.01) -- ppt3
OR (t1.lon BETWEEN 103.7896-0.01 AND 103.7896+0.01 -- ppt4
AND t1.lat BETWEEN 1.2754-0.01 AND 1.2754+0.01) -- ppt4
OR (t1.lon BETWEEN 103.7944-0.01 AND 103.7944+0.01 -- ppt5
AND t1.lat BETWEEN 1.2759-0.01 AND 1.2759+0.01) -- ppt5
OR (t1.lon BETWEEN 103.7992-0.01 AND 103.7992+0.01 -- ppt6
AND t1.lat BETWEEN 1.2764-0.01 AND 1.2764+0.01) -- ppt6
OR (t1.lon BETWEEN 103.61191878972717-0.01 AND 103.61191878972717+0.01 -- tuas
AND t1.lat BETWEEN 1.244386225597597-0.01 AND 1.244386225597597+0.01)) -- tuas
GROUP BY date, ns.description
ORDER BY date, ns.description, count DESC
```

	date date	status character varying (255)	count bigint
1	2023-09-01	At anchor	3
2	2023-09-01	Moored	35
3	2023-09-01	Not defined (default)	3
4	2023-09-01	Reserved for future use	3
5	2023-09-01	Restricted manoeuverability	2
6	2023-09-01	Under way sailing	5
7	2023-09-01	Under way using engine	73
8	2023-09-02	At anchor	8
9	2023-09-02	Engaged in Fishing	1
10	2023-09-02	Moored	40

Total rows: 26 of 26 | Query complete 00:00:03.742

Queries for weather and tide

```
-- (7a) What are the tide heights for per day?
SELECT day as date, height, high_low
FROM tide
WHERE day IS NOT NULL
ORDER BY date ASC

-- (7b) What is the average windspeed per day?
SELECT date_time, wind_speed
FROM weather
ORDER BY date_time ASC

-- (7c) What is the average visibility per day?
SELECT date_time, visibility
FROM weather
ORDER BY date_time ASC

-- (7d) What is the average sea pressure per day?
SELECT date_time, sea_level_pressure
FROM weather
ORDER BY date_time ASC
```

	date date	height double precision	high_low character varying (255)
1	2023-09-01	0.02	Low
2	2023-09-01	3.25	High
3	2023-09-01	0.96	Low
4	2023-09-02	3.47	High
5	2023-09-02	0.11	Low
6	2023-09-02	3.27	High
7	2023-09-02	0.74	Low
8	2023-09-03	3.49	High
9	2023-09-03	0.33	Low
10	2023-09-03	3.24	High

Total rows: 115 of 115 | Query complete 00:00:00.096

	date_time date	wind_speed double precision
1	2023-09-01	13.454166666666666
2	2023-09-02	11.308333333333332
3	2023-09-03	14.424999999999999
4	2023-09-04	16.391666666666666
5	2023-09-05	11.9
6	2023-09-06	8.720833333333333
7	2023-09-07	7.7625
8	2023-09-08	11.354166666666666
9	2023-09-09	9.283333333333333
10	2023-09-10	9.479166666666666

Total rows: 30 of 30 | Query complete 0

	date_time date	visibility double precision
1	2023-09-01	10.058333333333334
2	2023-09-02	9.870833333333334
3	2023-09-03	9.954166666666667
4	2023-09-04	9.379166666666666
5	2023-09-05	9.85
6	2023-09-06	8.483333333333333
7	2023-09-07	9.658333333333333
8	2023-09-08	9.616666666666667
9	2023-09-09	9.5375
10	2023-09-10	9.658333333333333

Total rows: 30 of 30 | Query complete 0

	date_time date	sea_level_pressure double precision
1	2023-09-01	1009.9416666666667
2	2023-09-02	1009.0291666666667
3	2023-09-03	1008.3166666666667
4	2023-09-04	1007.5291666666667
5	2023-09-05	1008.9499999999999
6	2023-09-06	1010.9791666666666
7	2023-09-07	1010.3625000000001
8	2023-09-08	1010.1833333333334
9	2023-09-09	1010.2624999999999
10	2023-09-10	1010.3666666666667

Total rows: 30 of 30 | Query complete 0

5.2. Appendix B: Callback Functions

Callback function for "Ship by type in the Singaporean waters"

```
#Graph 5
@app.callback(
    dash.dependencies.Output('part2-right-graph5', 'figure'),
    [dash.dependencies.Input('part2-left-ViewRadioItems-input', 'value'),
     dash.dependencies.Input('part2-left-DatePickerRange-input', 'start_date'),
     dash.dependencies.Input('part2-left-DatePickerRange-input', 'end_date')])
def update_graph5(value, start_date, end_date):
    #['Vessel Type', 'Navigation Status']
    if value == 'Vessel Type':
        df = get_data_graph5_type(start_date, end_date)
        fig = plot_graph5_type(df, start_date, end_date)
    else:
        fig = get_data_graph5_satus_and_plot(start_date, end_date)
    return fig
```

Callback function for "Ships in the Singapore port against benchmark"

```
#Graph 6
@app.callback(
    dash.dependencies.Output('part2-right-graph6', 'figure'),
    [dash.dependencies.Input('part2-left-TopRadioItems-input', 'value'),
    dash.dependencies.Input('part2-left-DatePickerRange-input', 'start_date'),
     dash.dependencies.Input('part2-left-DatePickerRange-input', 'end_date')])
def update_graph6(value, start_date, end_date):
    df = get_data_graph6(start_date, end_date)
    fig = plot_graph6(df, value, start_date, end_date)
    return fig
```

Callback function for "Weather Condition per day"

```
#Graph 7
@app.callback(
    dash.dependencies.Output('part2-right-graph7', 'figure'),
    [dash.dependencies.Input('part2-left-WeatherRadioItems-input', 'value'),
    dash.dependencies.Input('part2-left-DatePickerRange-input', 'start_date'),
     dash.dependencies.Input('part2-left-DatePickerRange-input', 'end_date')])
def update_graph7(value, start_date, end_date):
    if value == 'tides':
        df = get_data_graph7_tides(start_date, end_date)
        fig = plot_graph7_tides(df, start_date, end_date)
    else:
        df = get_data_graph7_others(value, start_date, end_date)
        df.rename(columns={df.columns[0]: 'date'},inplace=True)
        df.rename(columns={df.columns[1]: 'value'},inplace=True)
        fig = plot_graph7_others(df, value, start_date, end_date)
    return fig
```

Callback function for the interactive map

```python
# Graph 8 (map)
@app.callback(
    Output('ship-map', 'srcDoc'),
    [Input('date-picker', 'date'), Input('ship-limit-slider', 'value'), Input('hour-slider', 'value'), Input('country-dropdown', 'value')]
)
def update_map(selected_date, ship_limit, Shour,selected_country):
    global ShipLimit
    ShipLimit = ship_limit
    country_name = selected_country

    if country_name == "All":
        query1 = """
            SELECT t1.mmsi, t1.time, t1.lat, t1.lon
            FROM type1_2_3 t1
            WHERE t1.mmsi IN (SELECT DISTINCT mmsi FROM type5)
            AND DATE(t1.time) = %s
            AND EXTRACT(HOUR FROM t1.time) = %s
            AND t1.mmsi IS NOT NULL
            AND t1.lon IS NOT NULL
            AND t1.lat IS NOT NULL
            LIMIT %s;
        """
        location_df = pd.read_sql_query(query1, conn, params=(selected_date, Shour, ShipLimit))
        lats1 = location_df['lat'].tolist()
        lons1 = location_df['lon'].tolist()
        mmsi1 = location_df['mmsi'].tolist()
        time1 = location_df['time'].tolist()
        country1 = []
    else:
        query1 = """
            SELECT t1.mmsi, t1.time, t1.lat, t1.lon, c.country_name
            FROM type1_2_3 t1
            LEFT JOIN country_code c ON t1.mmsi / 1000000 = c.mid
            WHERE t1.mmsi IN (SELECT DISTINCT mmsi FROM type5)
            AND DATE(t1.time) = %s
            AND c.country_name = %s
            AND EXTRACT(HOUR FROM t1.time) = %s
            AND t1.mmsi IS NOT NULL
            AND t1.lon IS NOT NULL
            AND t1.lat IS NOT NULL
            LIMIT %s;
        """
        location_df = pd.read_sql_query(query1, conn, params=(selected_date, country_name, Shour, ShipLimit))
        lats1 = location_df['lat'].tolist()
        lons1 = location_df['lon'].tolist()
        mmsi1 = location_df['mmsi'].tolist()
        time1 = location_df['time'].tolist()
        country1 = location_df['country_name'].tolist()

    map_file_path = create_map(lats1, lons1, mmsi1, country1, time1)
    with open(map_file_path, 'r') as map_file:
        map_html = map_file.read()

    return map_html
```

5.3. Appendix C: Screen Copies of Interface