[ΥΣ09] SOFTWARE TECHNOLOGY

ONLINE PLATFORM FOR MONITORING REAL-TIME SHIPPING ACTIVITY

11152100043 - KAJACKA ERIK

11152100045 - KALAMPOKIS EVAGGELOS

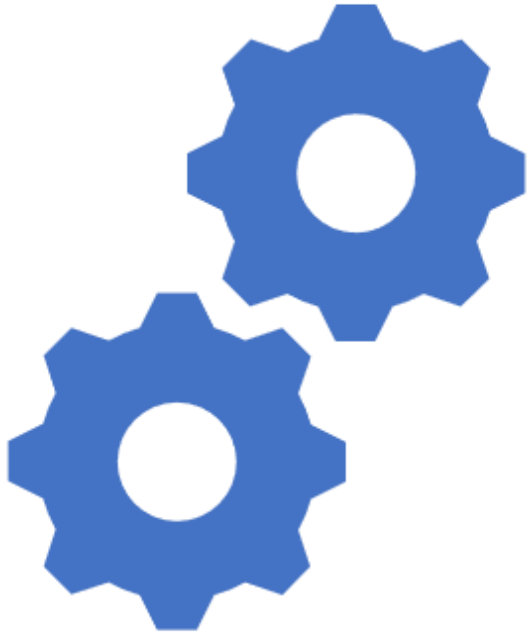
11152100108 - MOUMOULIDIS ANASTASIOS

11152100192 - TSELIKAS PANAGIOTIS

11152100275 - CHRYSOS DIMITRIOS

DEPARTMENT OF INFORMATICS & TELECOMMUNICATIONS

HELLENIC REPUBLIC
National and Kapodistrian University of Athens
EST. 1837



System Overview

Subsystems:

- Frontend (**React**)
- Backend (**Spring Boot**)
- Apache Kafka (**Producer Python, Consumer Java, One broker/topic**)
- Database (**MSSQL – Docker – Flyway Migration Scripts**)
- Git - Version Control (**GitHub**)

Real time visualization of ships using Kafka

01

Python Kafka producer reads CSV and sends JSON messages containing ship info.

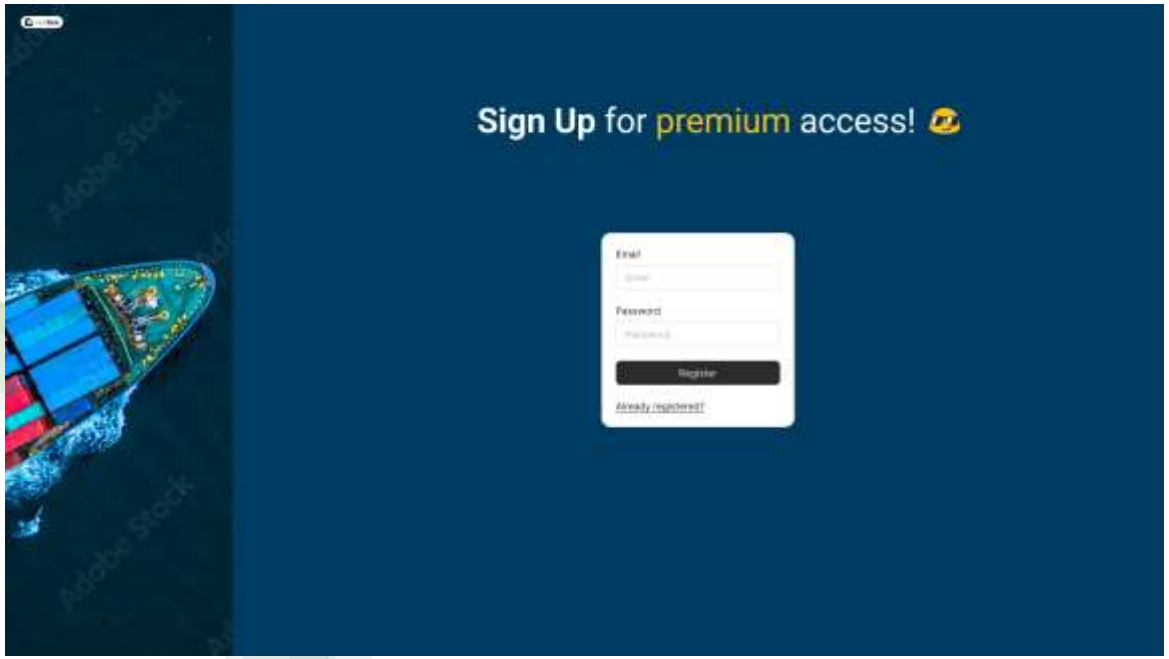
02

A Java Kafka Consumer receives the vessel messages and forwards them to the frontend in real time via WebSocket.

03

Frontend (React) receives the message and displays the ship on the map.

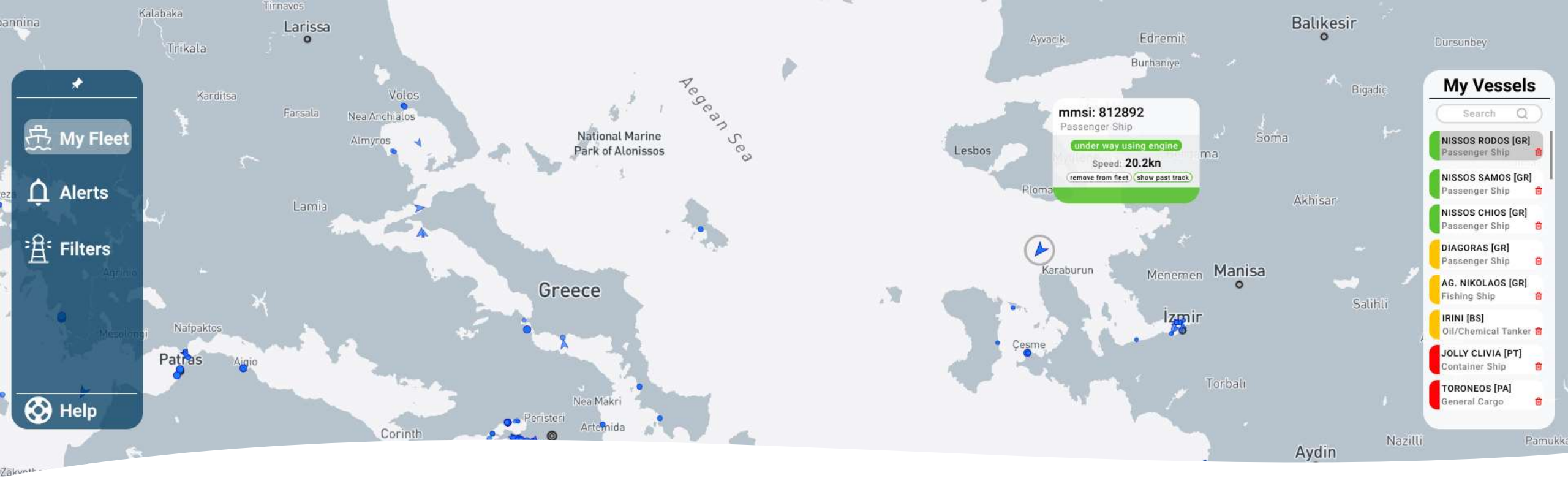
User authentication and authorization



- The application will use JWT tokens to handle secure, stateless sessions.
- All users will be saved in the database, and roles will be used to control access based on permissions.
- Roles will work by inheriting a simple User class and using validators in both the frontend and the backend.

Tracking the previous routes of ships

- The system will **persist consumed vessel messages into a database** table named **vessel_history_data**, potentially in batches for efficiency.
- When a **user accesses** the vessel **history feature**, the **backend** will **retrieve** and **return** all **messages** from the **last 12 hours** for the specified ship.
- The **frontend** will then **visualize** this **data**, allowing users to review the vessel's recent movement history.

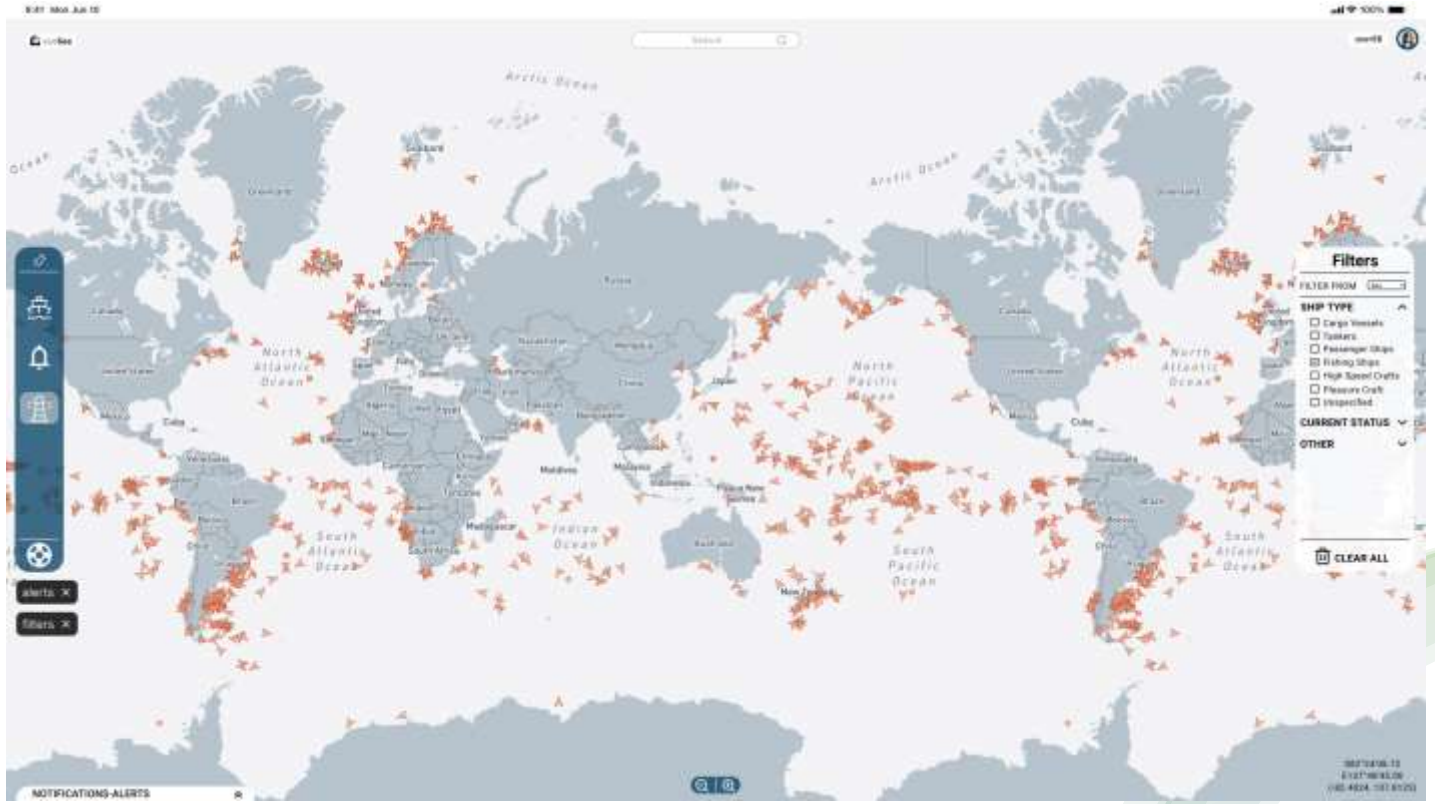


Allowing Registered users to save ships of interest to their fleet.

- The **implementation** will rely on **role-based access control** and **Java validators**, ensuring that **only** users with the appropriate **RegisteredUser role** are **allowed** to use this functionality.
- The **addition** or **removal** of a **vessel** from the **fleet** will be **communicated** from the **frontend** to the **backend** via **two endpoints**.
- The data will be persisted using registered_user and vessel tables in the database, through a many-to-many relationship, allowing each user to associate with multiple vessels and vice versa.
- The frontend will provide a list of the vessels saved on their fleet when MyFleet option is enabled

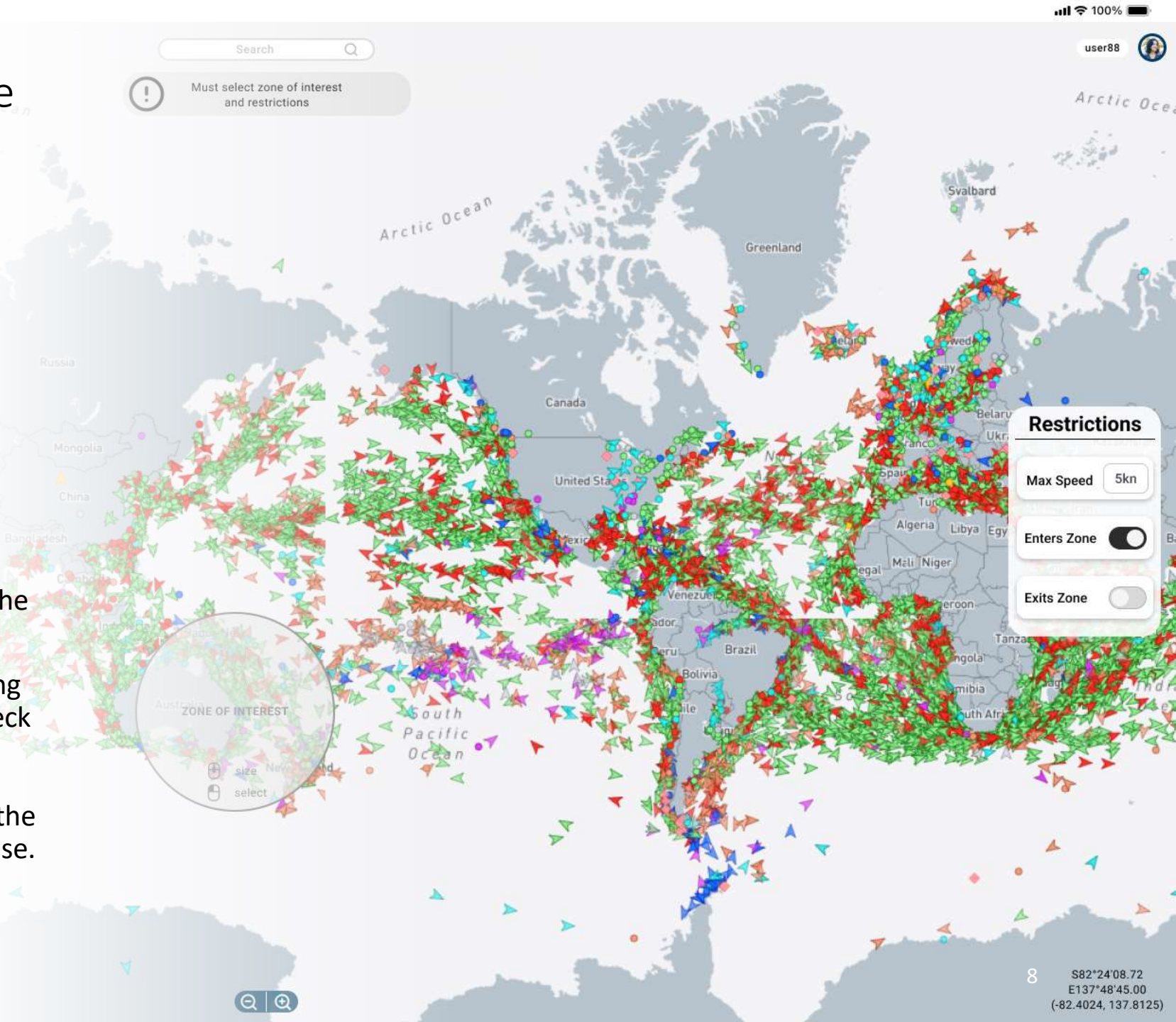
Allowing registered users to display ships on the map based on filters

- Users will be able to **filter** vessels by **type, status** and if they are part of **MyFleet.**
- The **frontend** will provide a user-friendly interface with dropdowns (My Fleet) and checkboxes (type & status) for filter options.
- The **backend** will use the **getMap()** operation to retrieve the latest data per vessel from the vessel_history_data table when filters are applied or reset. The operation will work by passing the **FiltersDTO** argument to the operation to specify the filters.



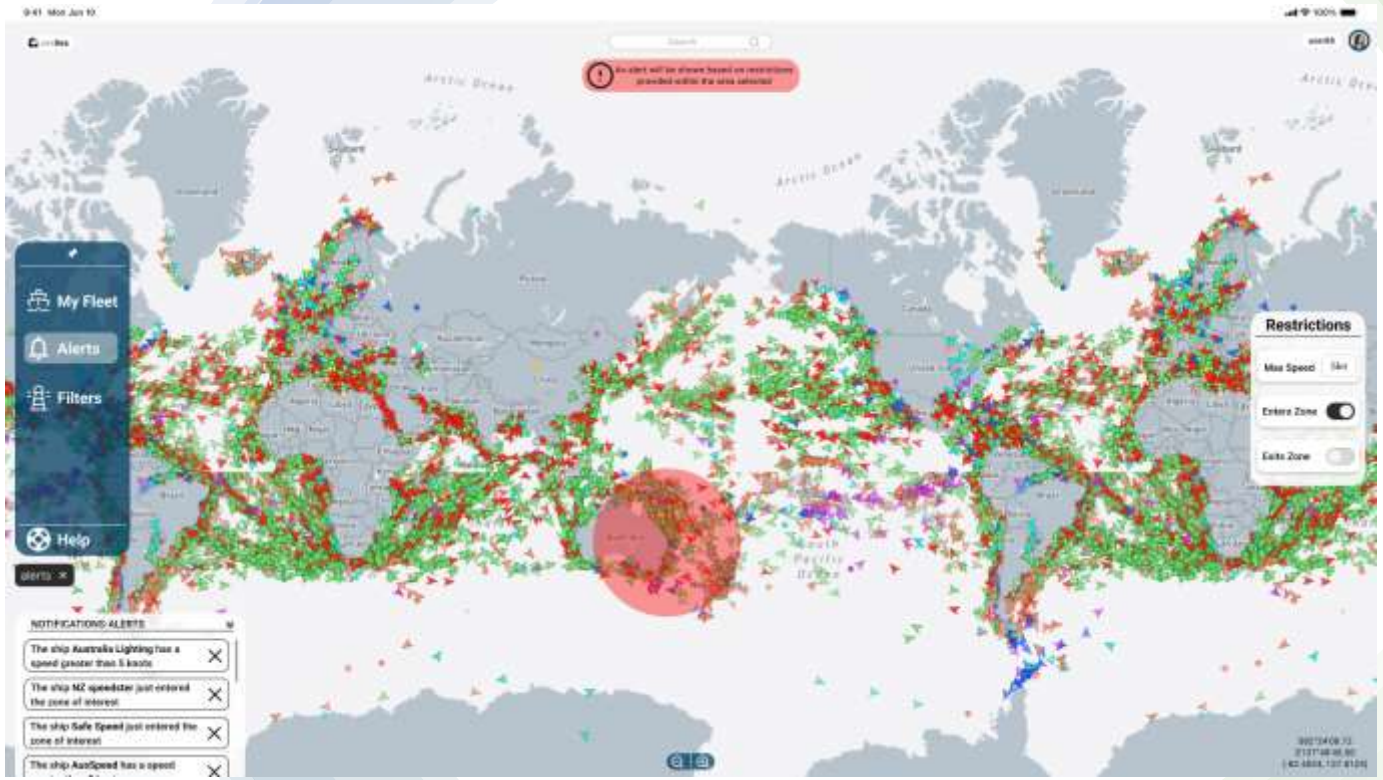
Registered users can create zones of interest with restrictions for which a notification will be sent when violated

- Users will be able to receive alerts by **Max Speed** and if the vessels **Enter** or **Exit** the **Zone.**
- The frontend will provide a **text-input** (Max Speed) and **toggle switches** (Enters/Exits Zone) for restriction options. It will also use a circle to set the zone.
- The backend will process each incoming message (via Kafka streaming) and check it against users' zone restrictions, returning notifications when relevant conditions are met. It will also persist the per user zone of interest in the database.



Registered users will have access to a notification list

- The alerts we talked about before will be also listed as notifications.
- The frontend will provide the list of the notifications and buttons to remove them.
- The backend will persist the user alerts to the notification table and have endpoints to retrieve and delete them.

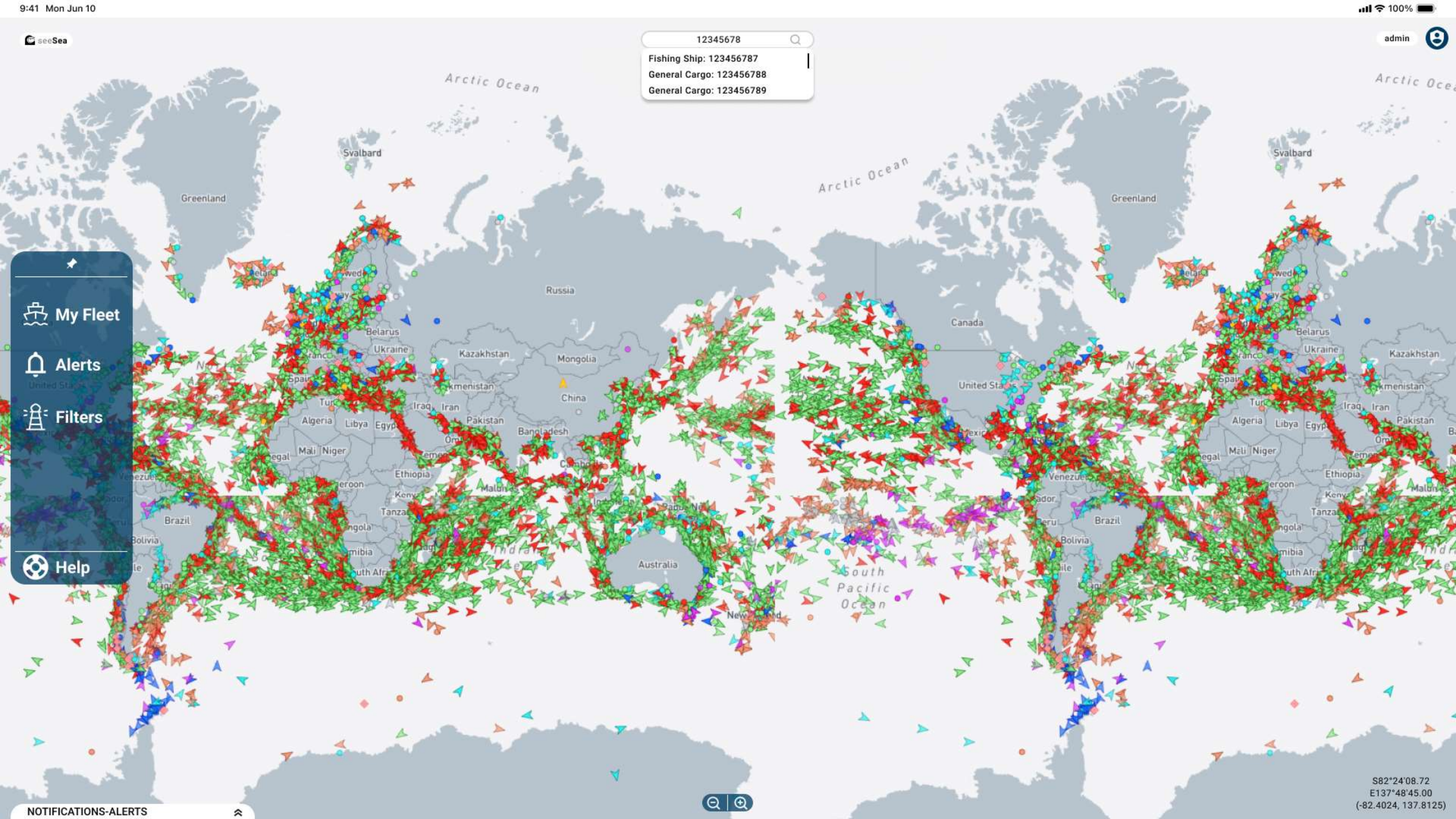


Administrators can edit static data about the ships

- Admin users will have the ability to change the vessel type.
- The frontend will provide an edit button on the vessel info popup, to change the type of a vessel when the user is an Administrator.
- The backend will persist this change to the database using the editVesselInfo() operation in the Administrator class that will be called by a respective endpoint.

Search Bar

- The frontend will display a search bar that filters ships by mmsi in real time. When a ship is selected, the map will automatically zoom to its location and highlight it.
- The backend will provide an endpoint that returns basic information for all ships, including their mmsi and coordinates, allowing the frontend to perform the search and zoom functionality efficiently.


9:41 Mon Jun 10
100%
seeSea
12345678
Fishing Ship: 123456787
General Cargo: 123456788
General Cargo: 123456789
admin
My Fleet
Alerts
Filters
Help
Arctic Ocean
Svalbard
Greenland
Russia
Canada
Belarus
Ukraine
Kazakhstan
Mongolia
China
Iraq
Iran
Pakistan
Bangladesh
Algeria
Libya
Egypt
Mali
Niger
Ethiopia
Brazil
Bolivia
Australia
South Pacific Ocean
NOTIFICATIONS-ALERTS
S82°24'08.72
E137°48'45.00
(-82.4024, 137.8125)

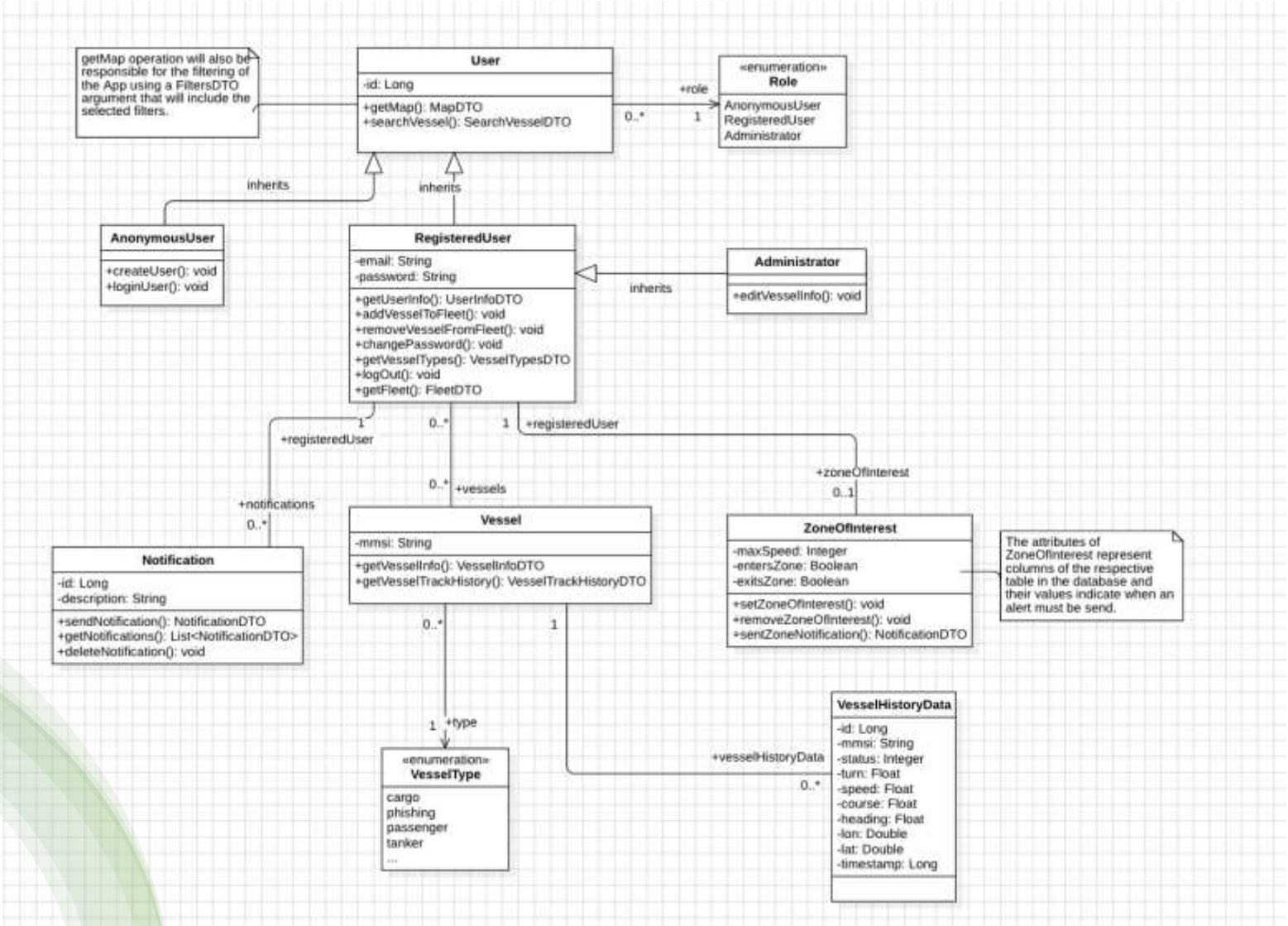
User authentication and data transmissions must be secured using HTTPS

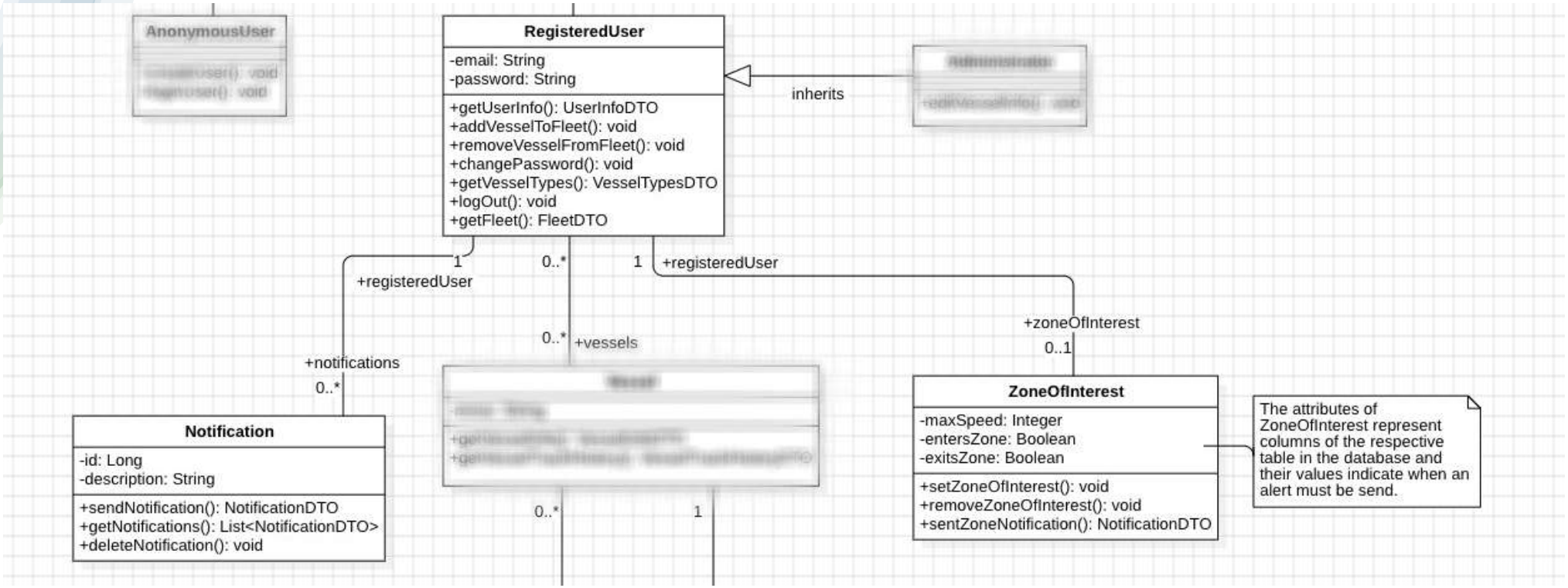
- A **self-signed certificate** will be generated and configured in the Spring Boot backend to enable encrypted connections.
- Key generation: `gen_key.sh` (creates `.p12` keystore)
- Certificate export: `export_cert.sh` (exports `.crt` file)

GDPR compliance must be adhered regarding user data storage and handling

- We will ensure **GDPR** compliance by securely storing user data in the database, only collecting information necessary for account functionality.
- Users will have the ability to access, update, or delete their personal information upon request. All sensitive data will be handled with care, using encryption and role-based access control.

UML Class Diagram

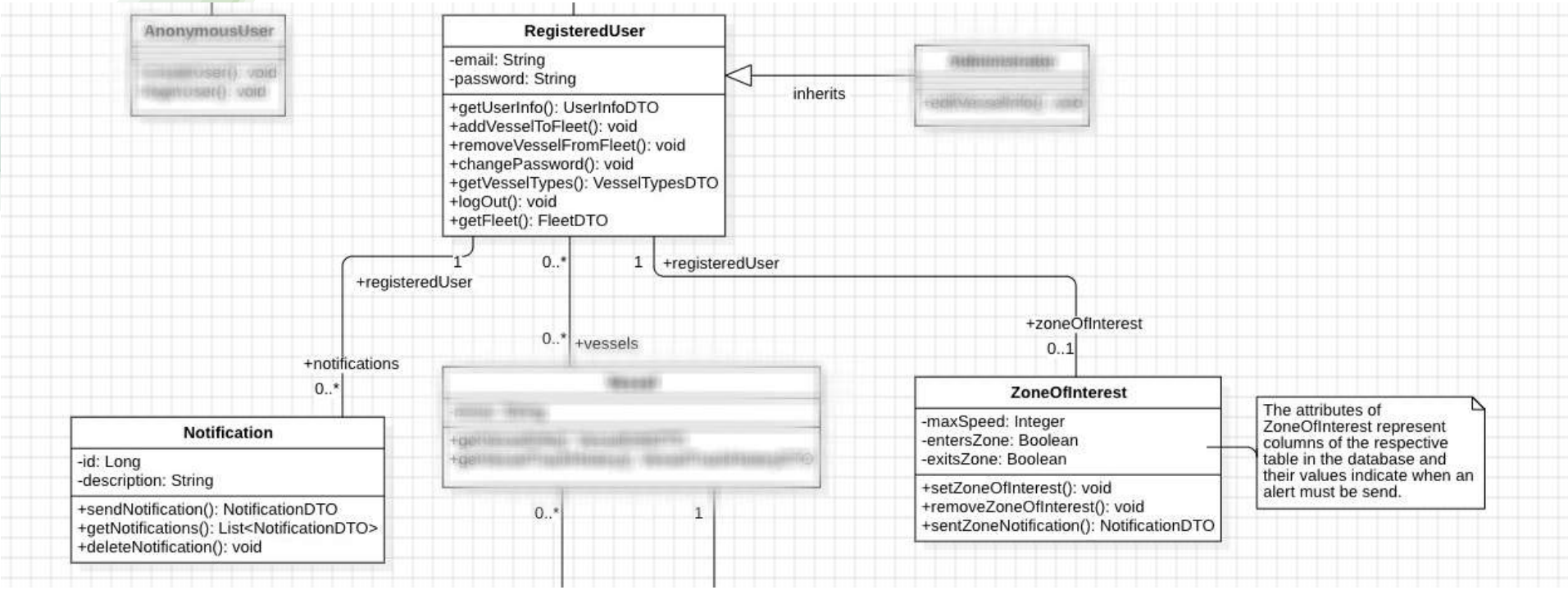




This feature is implemented by the classes **ZoneOfInterest**, **RegisteredUser** and **Notification**.

Zone Of Interest

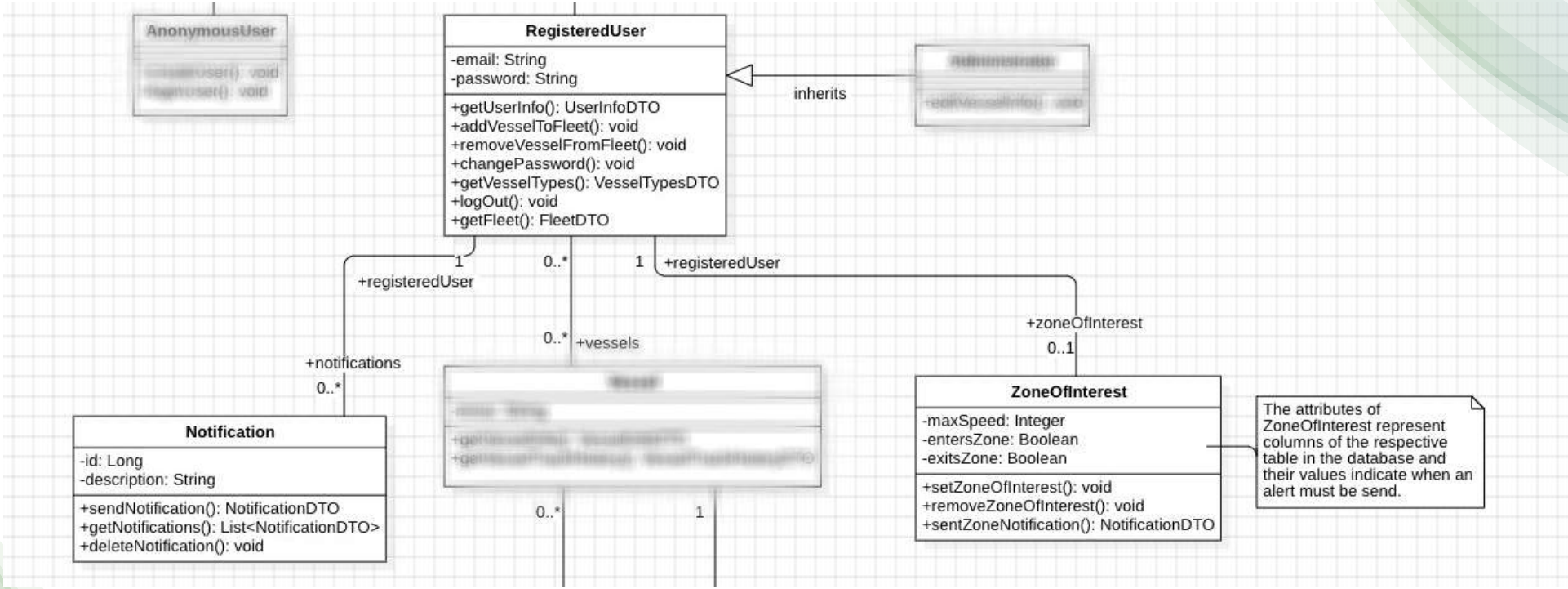
Zone Of Interest



Attributes of ZoneOfInterest class:

- maxSpeed -> When the zone is set, alerts will be sent and notifications will be created for the vessels that have a speed greater than the one set on this parameter.
- entersZone, exitsZone -> When these attributes are set to true, alerts will be sent and notifications will be created for vessels that enter or exit the zone, respectively.

Zone Of Interest



Operations of **ZoneOfInterest** class:

- **setZoneOfInterest()**, **removeZoneOfInterest()** -> sets (removes respectively) the zone for a user and is triggered via an endpoint from the frontend.
- **sentZoneNotification()**, responsible for generating and storing a notification, as well as delivering a real-time alert via WebSocket when a vessel breaches one of the user's defined zone conditions. It is triggered automatically as part of the real-time Kafka data stream processing in the backend.

Thank you for your time!

Full Project will be uploaded on github repo:
https://github.com/erikk03/softwareTechnology