

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО»**

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

ДО ЗАХИСТУ ДОПУЩЕНО

В.о. завідувача кафедри
Олександр КОВАЛЬ

«__» _____ 2026 р.

Дипломна робота

**на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інженерія програмного забезпечення
інтелектуальних кібер-фізичних систем в енергетиці»
спеціальності 121 Інженерія програмного забезпечення
на тему: Серверний додаток адміністрування даних гідроакустичних станцій**

Виконав:

студент IV курсу, групи ТВ-22

Данечкін Микита Олександрович

(прізвище, ім'я, по батькові)

(підпис)

Керівник:

асистент кафедри ПЗЕ Швайко В.Г

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Консультант з назва розділу (за наявності):

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Рецензент:

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Засвідчую, що у цій дипломній роботі
немає запозичень із праць інших
авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2026

**Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»**

Навчально-науковий інститут атомної та теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці

Рівень вищої освіти перший (бакалаврський)

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення
інтелектуальних кібер-фізичних систем в енергетиці»

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

_____ Олександр КОВАЛЬ

«___» _____ 2026р.

ЗАВДАННЯ

на дипломну роботу студенту

_____ Данечкіну Микиті Олександровичу

(прізвище, ім'я, по батькові)

1. Тема роботи: Серверний додаток адміністрування даних гідроакустичних станцій

керівник роботи Швайко Валерій Григорович

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «28» 05 2026 р. № 1992-с

2. Строк подання студентом роботи «10» червня 2026 р.

3. Вихідні дані до роботи: Серверний додаток мовою Java написаний використовуючи фреймворк Spring Boot

4. Зміст (дипломної роботи) пояснювальної записки (перелік завдань, які потрібно розробити): проаналізувати предметну область системи, спроектувати систем локалізації та моделювання руху, розробити спроектовані системи, протестувати роботу створених систем, провести аналіз роботи та зробити висновки щодо результатів роботи та потенціального розвитку системи

5. Перелік ілюстративного матеріалу: діаграми Use Case для систем, діаграми класів, схема розгорнутого додатку, sequence діаграма додатків, рисунки графічного інтерфейсу систем, скріншоти частин методів коду

6. Дата видачі завдання «31» жовтня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Строки виконання етапів роботи	Примітка
1	Отримання завдання	31.10.2025	
2	Дослідження предметної області	01.11.2025 – 31.11.2025	
3	Дослідження існуючих рішень	01.12.2025 – 31.12.2025	
4	Постановка вимог до проектування системи	01.01.2026 – 31.01.2026	
5	Розробка програмного продукту	01.02.2026 – 03.05.2026	
6	Тестування	04.05.2026 – 10.05.2026	
7	Захист програмного продукту	11.05.2026 – 15.05.2026	
8	Оформлення дипломної роботи	18.05.2026 – 31.05.2026	
9	Передзахист	01.06.2026 – 05.06.2026	
10	Захист	15.06.2026 – 19.06.2026	

Студент

(підпис)

Микита ДАНЕЧКІН

(ім'я, ПРІЗВИЩЕ)

Керівник роботи

(підпис)

Валерій ШВАЙКО

(ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

Структура та обсяг дипломної роботи. Робота містить 58 сторінок, 16 рисунків, 2 таблиці та 2 додатки.

Метою роботи: є розробка серверного додатку для адміністрування даних гідроакустичних станцій та локалізування водних об'єктів.

Методи дослідження: аналіз предметної області, методи геоінформаційного моделювання, математичні методи тріангуляції, алгоритми обробки сигналів, методи об'єктно-орієнтованого проектування програмного забезпечення, технології мікросервісної архітектури та засоби тестування програмних систем.

Отримані результати: розроблено програмний комплекс, що складається із сервісу симуляції руху водних об'єктів та сервісу локалізації та адміністрування даних. Реалізовано механізм генерації сигналів, моделювання руху об'єктів на поверхні Землі, передачу даних між сервісами, алгоритм локалізації на основі перетину азимутів, збереження результатів у базі даних та візуалізацію даних на інтерактивних картах.

Практичне значення одержаних результатів: полягає в можливості моделювання процесів пасивного спостереження та тестування алгоритмів локалізації об'єктів у реальному часі.

Ключові слова: гідроакустична станція, локалізація об'єктів, тріангуляція, азимут, обробка сигналів, мікросервісна архітектура, spring boot, websocket, моделювання руху.

ABSTRACT

Structure and scope of the thesis. The thesis consists of 58 pages, 16 figures, 2 tables, and 2 appendices.

The purpose of the thesis is to develop a server-side application for hydroacoustic station data management and localization of waterborne objects.

Research methods: domain analysis, geoinformation modeling methods, triangulation-based mathematical methods, signal processing algorithms, object-oriented software design methods, microservice architecture technologies, and software testing techniques.

Results obtained: a software system consisting of a water object movement simulation service and a localization and data management service has been developed. The system implements signal generation mechanisms, simulation of object movement on the Earth's surface, inter-service data exchange, a localization algorithm based on azimuth intersection, data persistence in a database, and visualization of results on interactive maps.

Practical significance of the results: the developed system enables simulation of passive surveillance processes and testing of object localization algorithms in real time.

Keywords: hydroacoustic station, object localization, triangulation, azimuth, signal processing, microservice architecture, Spring Boot, WebSocket, movement simulation.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ	8
ВСТУП	9
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ФОРМУВАННЯ ВИМОГ ДО СИСТЕМИ	10
1.1 Постановка задачі	11
1.2 Вимоги до систем	12
Висновки до розділу 1	14
2 ЗАСОБИ РОЗРОБКИ	16
2.1 Засоби розробки серверної частини	16
2.1.1 Сервіс симуляції	16
2.1.2 Сервіс локалізації	17
2.2 Засоби зберігання даних	18
2.3 Засоби контейнеризації	19
2.4 Розгортання системи	20
Висновки до розділу 2	21
3 АРХІТЕКТУРА ТА ВЗАЄМОДІЯ КОМПОНЕНТІВ СИСТЕМИ	23
3.1 Архітектура та взаємодія компонентів системи	23
3.1.1 Сервіс симуляції	23
3.1.2 Архітектура основного сервісу	24
3.2 Взаємодія користувача з системою	25
3.3 Візуалізація роботи системи	27
Висновки до розділу 3	27
4 РЕАЛІЗАЦІЯ СИСТЕМИ	29
4.1 Реалізація сервісу симуляції	29
4.1.1 Модель водного об'єкта	29
4.1.2 Реалізація механізму переміщення об'єктів	30
4.1.3 Реалізація генерації та передачі сигналів	32
4.2 Реалізація сервісу локалізації	34
4.2.1 Прийом та збереження сигналів	34
4.2.2 Реалізація алгоритму локалізації	35
4.2.3 Передача даних клієнтам у режимі реального часу	37
4.3 Математична модель локалізації об'єктів	38
Висновки до розділу 4	41
5 КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС ТА РЕЗУЛЬТАТИ РОБОТИ СИСТЕМИ	43

5.1 Інтерфейс сервісу симуляції	43
5.2 Інтерфейс перегляду сигналів станції	45
5.3 Глобальна карта локалізації	46
Висновки до розділу 5	48
6 КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС ТА РЕЗУЛЬТАТИ РОБОТИ СИСТЕМИ	50
6.1 Методика тестування	50
6.2 Тестування сервісу симуляції	51
6.3 Тестування сервісу локалізації	52
6.4 Інтеграційне тестування взаємодії сервісів	53
6.5 Аналіз результатів тестування	54
Висновки до розділу 6	54
ВИСНОВКИ	56
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	58

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ	Програмне Забезпечення
JPA	Java Persistence API
SQL	Simple Query Language
ORM	Object-Relational Mapping
REST	Representational State Transfer
HTTP	Hypertext Transfer Protocol

ВСТУП

Сучасні системи моніторингу та спостереження дедалі частіше використовують методи визначення положення об'єктів на основі аналізу сигналів, отриманих мережею сенсорів або станцій спостереження. Подібні підходи застосовуються у сфері морської навігації, контролю водного простору та системах безпеки. Особливістю таких систем є необхідність працювати з поточними даними в режимі реального часу, обробляти сигнали від багатьох джерел та виконувати локалізацію об'єктів за відомими отриманими даними.

Предметною областю дипломної роботи є розробка серверного додатку для адміністрування даних гідроакустичних станцій та локалізування водних об'єктів. Система побудована за принципом взаємодії двох окремих сервісів: сервісу симуляції, який відповідає за моделювання об'єктів, їх руху та надсилання сигналів, та основного сервісу, що виконує прийом, обробку та аналіз цих сигналів. Основною ідеєю є визначення приблизного місцеположення об'єктів лише за азимутом та силою сигналу без доступу до реальних координат джерела.

У ході роботи було реалізовано механізми випадкової генерації та переміщення водних об'єктів у межах визначеної зони, асинхронне надсилання сигналів, обробку поточних даних та алгоритми локалізації об'єктів методом тріангуляції. Для серверної частини застосунку використовувались Spring Boot, Spring Scheduling, WebSocket/STOMP та JPA/Hibernate.

Практична цінність розробленої системи полягає у можливості моделювання процесів пасивного спостереження та тестування алгоритмів локалізації об'єктів у реальному часі. У подальшому систему можна розширити підтримкою складніших алгоритмів тріангуляції, фільтрації шумів, а також інтеграцією з реальними телеметричними або сенсорними даними.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ФОРМУВАННЯ ВИМОГ ДО СИСТЕМИ

При моніторингу водних об'єктів однією з основних задач є визначення їх місцеположення на основі даних, отриманих від мережі станцій спостереження. Подібні системи використовуються для контролю морського руху, спостереження за суднами, аналізу навколишнього середовища, а також дослідження принципів роботи радіолокаційних та гідроакустичних комплексів. Особливістю таких систем є необхідність роботи з потоковими даними в режимі реального часу та виконання локалізації об'єктів за параметрами сигналів, які можуть містити похибки, надходити із затримкою або втрачатися під час передачі.

Одним із поширених підходів до визначення координат об'єктів є використання мережі сенсорів або станцій спостереження, які реєструють характеристики отриманих сигналів. У таких системах координати об'єкта визначаються непрямым способом на основі аналізу параметрів сигналів, отриманих від декількох джерел спостереження. Зокрема, кожен сигнал може містити інформацію про напрямок на об'єкт у вигляді азимута та додаткові характеристики, що дозволяють виконувати його подальшу обробку.

Важливим напрямом розвитку подібних систем є використання алгоритмів локалізації на основі тріангуляції. Суть підходу полягає у визначенні приблизного положення об'єкта за точкою перетину напрямків, отриманих від декількох станцій. Такий метод дозволяє реалізовувати пасивні системи спостереження, які не потребують передачі координат об'єкта безпосередньо та можуть використовуватися для моделювання роботи реальних гідроакустичних і радіолокаційних комплексів.

Для дослідження процесів локалізації доцільним є створення програмної системи, що поєднує моделювання руху водних об'єктів, генерацію сигналів, передачу даних між компонентами та виконання алгоритмів визначення координат у режимі реального часу. Такий підхід дозволяє аналізувати ефективність

алгоритмів локалізації, оцінювати їх точність та досліджувати вплив похибок вхідних даних на результати роботи системи.

Крім того, запропонована архітектура може слугувати основою для подальшого розвитку багатосервісної платформи моделювання та дослідження систем пасивного спостереження, а також для тестування нових методів обробки сигналів і локалізації об'єктів.

1.1 Постановка задачі

Основною вимогою до системи є реалізація механізму визначення координат рухомих водних об'єктів на основі сигналів, що надходять від декількох станцій спостереження в режимі реального часу.

Система повинна забезпечувати генерацію та переміщення водних об'єктів, формування сигналів із параметрами азимута та сили сигналу, передачу цих сигналів між сервісами, а також виконання локалізації об'єктів шляхом аналізу отриманих даних. Важливим аспектом є забезпечення коректної роботи системи в умовах великої кількості сигналів та постійного оновлення координат об'єктів.

Для реалізації поставленої задачі необхідно:

- реалізувати сервіс симуляції руху водних об'єктів;
- створити систему генерації сигналів для найближчих станцій;
- реалізувати механізм передачі сигналів між сервісами;
- забезпечити асинхронну обробку сигналів;
- реалізувати алгоритм локалізації об'єктів за азимутами сигналів;
- створити механізм збереження знайдених координат об'єктів;
- забезпечити оновлення даних на карті;
- виконати тестування основних компонентів системи;
- підготувати систему до розгортання у контейнеризованому середовищі.

1.2 Вимоги до систем

Під час виконання дипломної роботи були визначені функціональні та нефункціональні вимоги ПЗ. Основна увага була зосереджена на реалізації алгоритмі локалізації об'єктів з даних отриманих гідроакустичними станціями. В таблицях 1.1 та 1.2 представлені функціональні та нефункціональні вимоги до системи.

Таблиця 1.1 — Функціональні та нефункціональні вимоги основного сервісу

Функціональні вимоги	Нефункціональні вимоги
прийом сигналів від сервісу симуляції	система повинна обробляти сигнали у режимі реального часу
збереження сигналів у базі даних	локалізація об'єктів повинна виконуватись із мінімальною затримкою
зберігання інформації про станції спостереження	система повинна підтримувати одночасну роботу кількох станцій
визначення актуальних сигналів за часовим інтервалом	WebSocket-з'єднання повинні працювати стабільно при великій кількості оновлень
аналіз сигналів від різних станцій	система повинна бути стійкою до втрати окремих сигналів;
обчислення точки перетину азимутів	алгоритм локалізації повинен враховувати можливі похибки сигналів
визначення приблизних координат об'єкта	масштабування при збільшенні кількості сигналів
збереження знайдених координат локалізованих об'єктів	усі компоненти системи повинні бути логічно розділені за відповідальністю
відбір лише валідних сигналів для локалізації	система повинна підтримувати можливість інтеграції з іншими джерелами сигналів
виключення некоректних або суперечливих комбінацій сигналів	база даних повинна забезпечувати швидкий доступ до актуальних сигналів

Продовження таблиці 1.1

збереження історії руху локалізованих об'єктів	система повинна підтримувати подальше розширення алгоритмів локалізації та фільтрації сигналів
реалізацію WebSocket-підключення для realtime-оновлення карти	
трансляцію нових сигналів підписаним клієнтам	
відображення сигналів на карті у вигляді азимутальних ліній	
відображення станцій та локалізованих об'єктів на глобальній карті	
автоматичне оновлення карти без перезавантаження сторінки	

Таблиця 1.2 — Функціональні та нефункціональні вимоги сервісу симуляції

Функціональні вимоги	Нефункціональні вимоги
генерацію водних об'єктів у межах заданої географічної області	система повинна працювати у режимі реального часу
збереження інформації про створені об'єкти	затримка генерації та відправки сигналу не повинна перевищувати встановлений інтервал симуляції
автоматичне переміщення об'єктів через задані проміжки часу	система повинна підтримувати одночасну роботу з великою кількістю об'єктів
можливість налаштування швидкості та частоти руху об'єктів	передача сигналів між сервісами повинна бути стабільною навіть при короткочасних помилках мережі
визначення найближчих станцій до кожного об'єкта	система повинна бути побудована за принципами слабкої зв'язності між сервісами

Продовження таблиці 1.2

генерацію сигналів для найближчих станцій	конфігурація параметрів симуляції повинна задаватися через <code>environment variables</code> або <code>application.yml</code> ;
розрахунок азимута сигналу	система повинна підтримувати контейнеризоване розгортання;
розрахунок сили сигналу залежно від відстані	інтерфейс карти повинен коректно працювати у сучасних веб браузерях.
асинхронне надсилання сигналів до основного сервісу	
підтримку черги сигналів для уникнення перевантаження системи	
повторну відправку сигналів у разі помилки підключення	
відображення водних об'єктів та станцій на мапі	
автоматичне оновлення положення об'єктів на карті у режимі реального часу	

Висновки до розділу 1

В результаті аналізу предметної області було досліджено принципи функціонування систем локалізації об'єктів на основі сигналів, отриманих від мережі спостережних станцій. Встановлено, що для визначення координат об'єкта можуть використовуватись параметри сигналів, зокрема азимут напрямку та рівень сигналу, без необхідності передачі фактичних координат джерела.

На основі проведеного аналізу сформульовано постановку задачі та визначено основні функціональні можливості майбутньої системи. Було прийнято рішення реалізувати програмний комплекс у вигляді двох взаємодіючих сервісів: сервісу симуляції руху водних об'єктів і основного сервісу обробки сигналів та

локалізації. Такий підхід дозволяє відокремити процес генерації даних від процесу їх аналізу та забезпечує можливість подальшого масштабування системи.

У межах розділу також було сформовано функціональні та нефункціональні вимоги до кожного із сервісів. Визначені вимоги охоплюють процеси генерації сигналів, моделювання руху об'єктів, передачі інформації між сервісами, локалізації об'єктів, зберігання даних та візуалізації результатів на інтерактивній карті. Окрему увагу приділено вимогам до роботи системи в режимі реального часу, надійності обробки сигналів та можливості подальшого розширення функціональності.

Отримані результати аналізу предметної області та сформовані вимоги стали основою для вибору технологічного стеку, проектування архітектури системи та реалізації програмного забезпечення, які розглядаються у наступних розділах роботи.

2 ЗАСОБИ РОЗРОБКИ

Для реалізації системи локалізації водних об'єктів було використано стек Java розробки, засоби роботи з базами даних, інструменти realtime-взаємодії та засоби контейнеризації. Архітектура системи побудована за принципом взаємодії двох окремих сервісів: сервісу симуляції та основного сервісу локалізації. Підхід дозволяє розділити відповідальність між компонентами системи та забезпечити можливість їх незалежного масштабування і розвитку.

2.1 Засоби розробки серверної частини

Для створення серверної частини застосунку використано мову програмування Java та фреймворк Spring Boot. Основними перевагами Spring Boot є спрощення конфігурації застосунку, підтримка модульної архітектури та наявність великої кількості готових компонентів для роботи з веб технологіями, базами даних та асинхронною обробкою даних.

2.1.1 Сервіс симуляції

Сервіс симуляції відповідає за генерацію водних об'єктів, їх переміщення та створення сигналів для станцій спостереження. Для реалізації цього сервісу використовувалися наступні технології:

- Spring Boot — основа серверного застосунку;
- Spring Scheduling — для періодичного запуску задач переміщення об'єктів та генерації сигналів;
- Spring Web — для створення REST API та взаємодії між сервісами;
- Spring Data JPA — для роботи з базою даних через ORM;
- Hibernate — для автоматичного відображення Java-сутностей у таблиці бази даних;

- Leaflet — для інтерактивного відображення водних об'єктів та станцій на карті;
- Java Concurrency API — для реалізації черги сигналів та асинхронної відправки повідомлень.

Основною задачею сервісу є імітація руху водних об'єктів у межах визначеної географічної області. Після кожного переміщення об'єкт генерує сигнали для найближчих станцій, а сервіс виконує їх асинхронну передачу до основного сервісу локалізації.

2.1.2 Сервіс локалізації

Основний сервіс призначений для прийому сигналів, їх обробки та визначення координат об'єктів за допомогою алгоритмів локалізації. Для реалізації цього сервісу використовувались:

- Spring Boot — для створення серверного застосунку;
- Spring Web — для реалізації REST API;
- Spring WebSocket та STOMP — для realtime-взаємодії між сервером і клієнтом;
- Spring Scheduling — для періодичного запуску алгоритму локалізації;
- Spring Data JPA — для роботи з даними сигналів та локалізованих об'єктів;
- Hibernate — для ORM-взаємодії з базою даних;
- Leaflet — для візуалізації станцій, сигналів та знайдених координат на карті;
- SockJS та STOMP.js — для підтримки WebSocket-з'єднань на стороні клієнту.

Сервіс аналізує сигнали, отримані від різних станцій, виконує пошук точок перетину азимутів та визначає приблизне положення об'єкта. Отримані координати зберігаються у базі даних та відображаються на інтерактивній карті у режимі реального часу.

2.2 Засоби зберігання даних

Для зберігання даних у системі використано реляційну систему керування базами даних PostgreSQL. Вона була обрана завдяки високій продуктивності, надійності та підтримці комплексних SQL-запитів.

У базі даних зберігаються:

- дані про гідроакустичні станції;
- сигнали, отримані від сервісу симуляції;
- локалізовані координати об'єктів;
- інформація про водні об'єкти сервісу симуляції;
- історія координат локалізованих об'єктів.

Для взаємодії з базою даних використовувався Spring Data JPA разом із Hibernate. Фреймворк надає можливість працювати з даними через Java сутності без необхідності ручного написання більшості SQL-запитів.

Також були використані автоматичні механізми генерації таблиць та зв'язків між сутностями що значно спрощує розробку структури бази даних.

Схему бази даних можна побачити на рисунку 2.1 та 2.2.

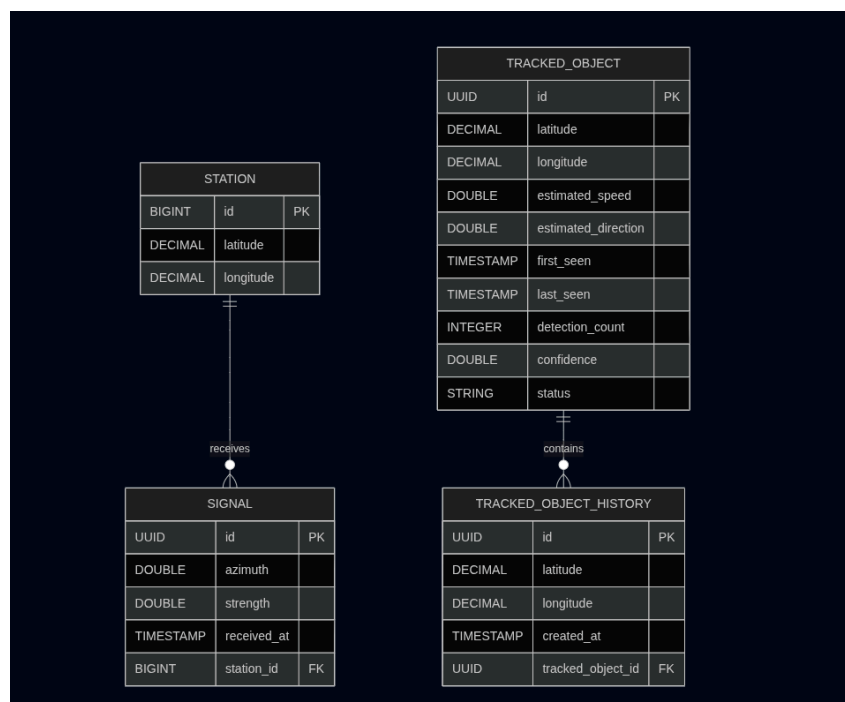


Рисунок 2.1 – Схема класів системи

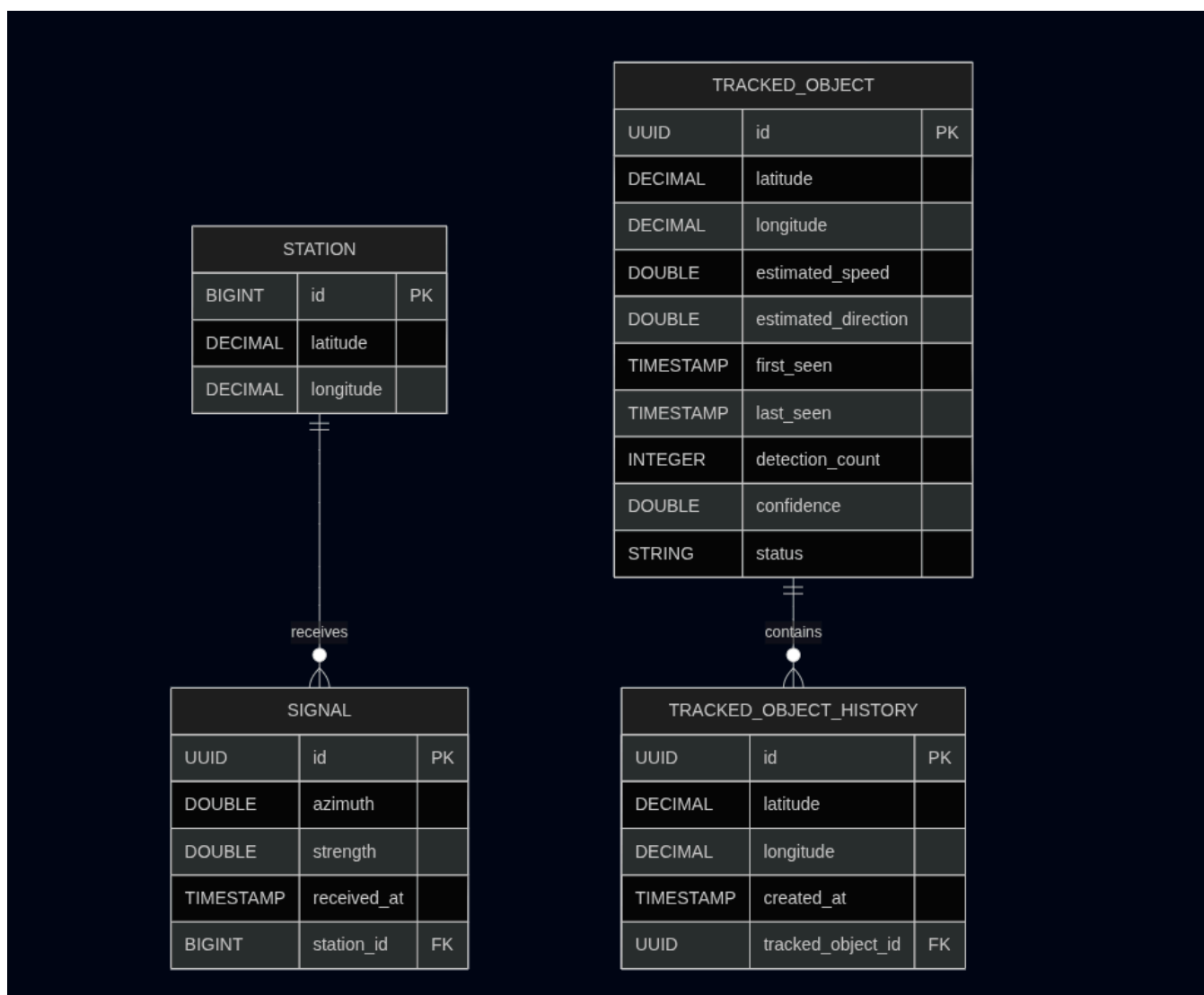


Рисунок 2.2 – Схема класів системи

Дані з наведених таблиць зберігаються у базі даних та доступні для подальшої обробки і аналізу. Дані про водні об'єкти постійно оновлюються відповідно до моделі їх руху, що реалізована у сервісі симуляції.

2.3 Засоби контейнеризації

Для контейнеризації та запуску системи використовувався Docker. Контейнеризація дозволяє ізолювати сервіси, забезпечити стабільність середовища виконання та спростити процес розгортання застосунку.

Для кожного сервісу був створений окремий Docker-контейнер:

- контейнер сервісу симуляції;
- контейнер основного сервісу локалізації;
- контейнери PostgreSQL.

Використання Docker забезпечує однакове середовище запуску на різних пристроях, спрощення налаштування залежностей, швидке розгортання системи, можливість подальшого масштабування сервісів та ізоляцію компонентів системи один від одного.

Для автоматизації запуску декількох контейнерів одночасно використовується Docker Compose, що дозволяє описати конфігурацію всієї системи в одному файлі та запускати її однією командою.

2.4 Розгортання системи

Для забезпечення можливості публікації сервісів у мережі Інтернет та їх подальшого тестування було розглянуто сучасні хмарні платформи для розгортання контейнеризованих застосунків. Як основний засіб розгортання обрано платформу Railway.

Railway є хмарним сервісом, який дозволяє виконувати автоматичне розгортання застосунків без необхідності самостійного налаштування серверної інфраструктури. Платформа підтримує інтеграцію з Git-репозиторіями, роботу з Docker-контейнерами та хмарними базами даних.

Основними перевагами Railway для даного проекту є:

- підтримка автоматичного розгортання застосунків із репозиторію;
- можливість використання контейнеризованих сервісів;
- централізоване керування змінними середовища;
- автоматичне налаштування мережевої взаємодії між сервісами;
- вбудовані засоби моніторингу та перегляду журналів роботи застосунків;
- можливість швидкого масштабування сервісів у майбутньому.

Для функціонування системи також передбачено використання окремих конфігураційних параметрів, що визначають характеристики симуляції, параметри підключення до баз даних та адреси взаємодії між сервісами. Усі такі параметри можуть зберігатися у змінних середовища, що відповідає сучасним практикам розробки хмарних застосунків.

Використання Railway разом із Docker дозволяє забезпечити однакові умови виконання системи незалежно від середовища запуску та спрощує процес подальшого розгортання і супроводу програмного комплексу.

Висновки до розділу 2

У другому розділі було розглянуто програмні засоби та технології, використані під час розробки системи адміністрування даних гідроакустичних станцій. Для реалізації серверної частини було обрано платформу Java та фреймворк Spring Boot, що забезпечують створення масштабованих застосунків із підтримкою модульної архітектури, роботи з базами даних та асинхронної обробки даних.

Окремо було проаналізовано технології, використані в сервісі симуляції та сервісі локалізації. Сервіс симуляції реалізує моделювання руху водних об'єктів, генерацію сигналів та їх передачу до основного сервісу. Основний сервіс виконує прийом сигналів, їх аналіз, локалізацію об'єктів та візуалізацію результатів. Для забезпечення роботи системи в режимі реального часу використовуються технології WebSocket та STOMP, що дозволяють оперативно передавати оновлення клієнтським застосункам.

У розділі також розглянуто організацію зберігання даних за допомогою PostgreSQL та засобів ORM, що забезпечують зручну взаємодію між програмним кодом і базою даних. Наведено структуру основних сутностей системи та описано підхід до збереження інформації про сигнали, станції та результати локалізації.

Для забезпечення можливості подальшого розгортання систем було розглянуто технологію Docker та хмарну платформу Railway. Їх використання дозволяє стандартизувати середовище виконання застосунку, спростити процес публікації сервісів та підготувати систему до роботи в умовах реальної експлуатації. Обраний набір технологій забезпечує надійну роботу системи, підтримку обробки даних у режимі реального часу та створює основу для подальшого розвитку алгоритмів локалізації й масштабування програмного комплексу.

Обраний набір технологій забезпечує надійну роботу системи, підтримку обробки даних у режимі реального часу та створює основу для подальшого розвитку алгоритмів локалізації й масштабування програмного комплексу.

3 АРХІТЕКТУРА ТА ВЗАЄМОДІЯ КОМПОНЕНТІВ СИСТЕМИ

У цьому розділі розглянуто програмну реалізацію системи додатку для адміністрування даних гідроакустичних станцій та додаток для моделювання водних об'єктів, їх архітектура та основні механізми взаємодії компонентів. Також описується процес роботи користувача із системою та основні сценарії використання програмного продукту.

3.1 Архітектура та взаємодія компонентів системи

Розроблений програмний додаток побудований за мікросервісною архітектурою та складається з двох незалежних сервісів: сервісу симуляції руху водних об'єктів та основного сервісу локалізації сигналів. Такий підхід дозволяє розділити логіку генерації даних та їх обробки, забезпечити кращу масштабованість системи та спростити подальший розвиток застосунку.

Обидва сервіси взаємодіють між собою через HTTP-запити. Сервіс симуляції генерує сигнали та надсилає їх до основного сервісу, а основний сервіс виконує аналіз сигналів, локалізацію об'єктів та візуалізацію результатів.

Архітектура системи включає: модуль симуляції руху об'єктів, модуль генерації сигналів, модуль прийому та обробки сигналів, модуль локалізації координат, модуль realtime-взаємодії через WebSocket та модуль візуалізації картографічних даних.

3.1.1 Сервіс симуляції

Сервіс симуляції відповідає за створення та рух водних об'єктів у межах визначеної географічної області. Кожен об'єкт має власні координати, які змінюються через певний проміжок часу за допомогою планувальників Spring Scheduler.

Після переміщення об'єкта сервіс:

- визначає дві найближчі станції;
- обчислює азимут до об'єкта;
- визначає силу сигналу залежно від відстані;
- створює сигнал;
- надсилає його до основного сервісу.

Для уникнення перевантаження система використовує внутрішню чергу повідомлень та асинхронну обробку сигналів.

Основні компоненти сервісу симуляції це WaterObjectService для управління водними об'єктами, PositionService для переміщення об'єктів, SenderService для генерації сигналів, QueueService для черги відправки сигналів, AsyncSignalSender для асинхронної передачі даних та GeoUtils що відповідає за географічні обчислення.

Схему Use Case діаграми можна побачити на рисунку 3.1.

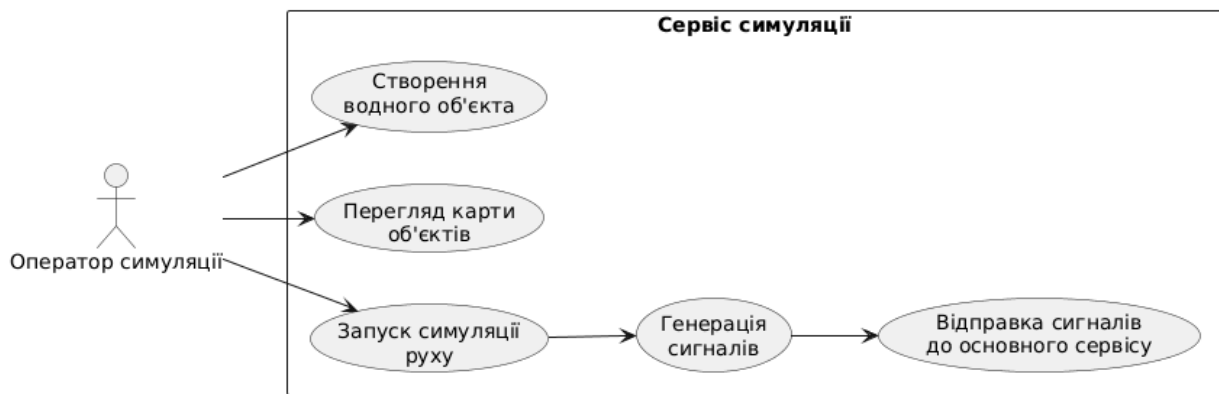


Рисунок 3.1 – Use Case сервісу симуляції

3.1.2 Архітектура основного сервісу

Основний сервіс відповідає за прийом сигналів від станцій, аналіз азимутів та визначення приблизних координат об'єктів.

Після отримання сигналу сервіс зберігає його у базі даних, групує сигнали за часовим інтервалом та виконує пошук перетину азимутів щоб обчислити

приблизне місцеположення об'єкта. Також система створює тимчасову сутність локалізованого об'єкта, що відслідковується та має збережену історію рухів. Сигнали транслюються клієнтам через WebSocket.

Схему Use Case діаграми можна побачити на рисунку 3.2.

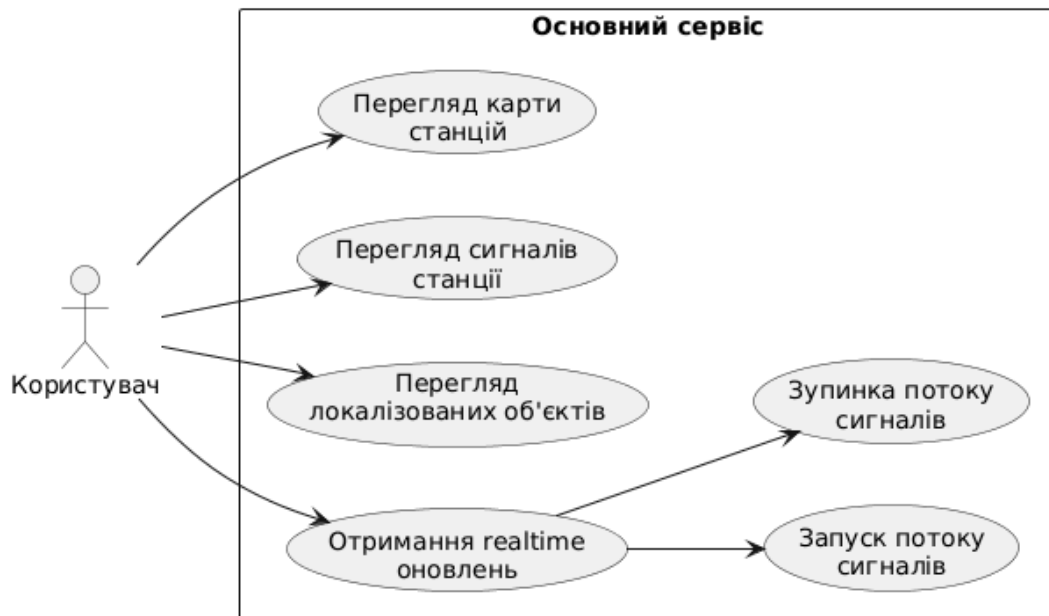


Рисунок 3.2 – Use Case основного сервісу.

Основні компоненти це SignalController для прийому сигналів, SignalService для робота із сигналами, LocatorService що відповідає за локалізацію об'єктів, LocatedEntityService для збереження результатів локалізації, SignalStreamService для realtime-передачі сигналів, HydroController з генерацією картографічного інтерфейсу та знову GeoUtils що містить математичні обчислення перетину азимутів.

3.2 Взаємодія користувача з системою

Система орієнтована на подальше підключення окремого frontend-застосунку, проте на поточному етапі містить вбудований веб інтерфейс для демонстрації роботи сервісів.

Інтерфейс у вигляді HTML-сторінок, які генеруються безпосередньо Spring Boot контролерами. Для відображення карт використовується бібліотека Leaflet та OpenStreetMap.

Користувач може:

- переглядати карту всіх станцій;
- спостерігати за рухом водних об'єктів;
- відкривати сторінку окремої станції;
- отримувати realtime-відображення нових сигналів;
- бачити приблизно локалізовані координати об'єктів.

На рисунку 3.3 продемонстровано sequence діаграму додатку.

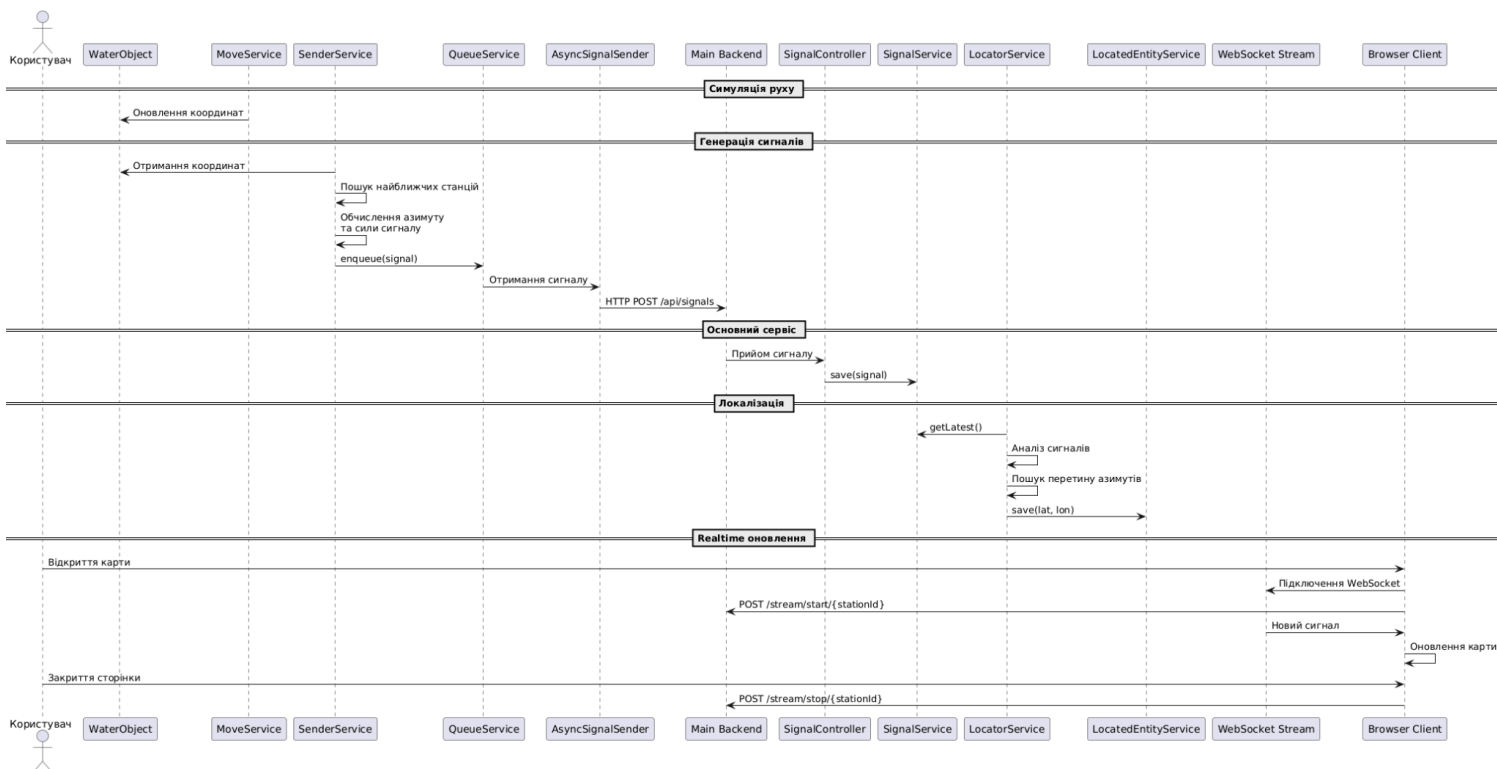


Рисунок 3.3 – sequence діаграма додатку

Для оновлення інформації в режимі реального часу використовується WebSocket-з'єднання через SockJS та STOMP. Після відкриття сторінки клієнт автоматично підписується на відповідний топік станції та отримує нові сигнали без перезавантаження сторінки.

3.3 Візуалізація роботи системи

Для демонстрації роботи розробленої системи було реалізовано вбудований веб інтерфейс з підтримкою картографічного відображення даних. Інтерфейс дозволяє в режимі реального часу спостерігати за рухом водних об'єктів, сигналами станцій та результатами локалізації.

Візуалізація побудована на основі бібліотеки Leaflet та картографічного сервісу OpenStreetMap. Дані оновлюються автоматично через WebSocket-з'єднання та періодичні HTTP-запити.

Висновки до розділу 3

У третьому розділі було розглянуто архітектуру розробленої системи локалізації водних об'єктів та особливості взаємодії її компонентів. Встановлено, що використання мікросервісного підходу дозволило розділити процес генерації даних та їх обробки між двома незалежними сервісами, що спростило структуру застосунку та забезпечило можливість подальшого масштабування окремих компонентів системи.

Було проаналізовано принцип роботи сервісу симуляції, який виконує генерацію водних об'єктів, моделює їх рух та формує сигнали для найближчих станцій спостереження. Також розглянуто архітектуру основного сервісу, що відповідає за прийом сигналів, їх збереження, аналіз та визначення координат об'єктів за допомогою алгоритму локалізації на основі перетину азимутів.

Описано взаємодію користувача із системою через вбудований вебінтерфейс, який забезпечує доступ до картографічного відображення станцій, сигналів та результатів локалізації. Для забезпечення роботи в режимі реального часу використано WebSocket-з'єднання, завдяки чому нові сигнали та результати обробки автоматично відображаються без необхідності оновлення сторінки.

Окрему увагу приділено візуалізації роботи системи. Реалізований картографічний інтерфейс дозволяє спостерігати за переміщенням водних об'єктів, роботою станцій спостереження та результатами локалізації на інтерактивній карті. Це значно спрощує аналіз роботи алгоритмів та наочно демонструє процес визначення координат об'єктів у режимі реального часу.

Таким чином, у межах розділу було сформовано повне уявлення про структуру програмного забезпечення, механізми взаємодії між сервісами та способи представлення результатів роботи системи користувачу. Отримані результати створюють основу для подальшого розгляду реалізації ключових алгоритмів локалізації та аналізу ефективності роботи розробленої системи.

4 РЕАЛІЗАЦІЯ СИСТЕМИ

Особливістю розробленої системи є використання двох незалежних сервісів, кожен з яких виконує власний набір задач. Сервіс симуляції відповідає за створення віртуального середовища, рух водних об'єктів та формування сигналів для станцій спостереження. Основний сервіс виконує прийом отриманих сигналів, їх збереження, обробку та локалізацію об'єктів за допомогою математичних алгоритмів.

У цьому розділі детально розглянуто реалізацію основних класів і сервісів системи, механізми взаємодії між компонентами, алгоритми обробки сигналів та математичну модель визначення координат об'єктів. Окрему увагу приділено реалізації алгоритму тріангуляції, який є ключовим елементом процесу локалізації.

4.1 Реалізація сервісу симуляції

4.1.1 Модель водного об'єкта

Основним елементом сервісу симуляції є водний об'єкт. Кожен об'єкт містить унікальний ідентифікатор та координати поточного розташування. Координати представлені широтою та довготою, що дозволяє виконувати подальші географічні розрахунки. Для зберігання значень координат використано `BigDecimal` для точності. На рисунку 4.1 можна побачити клас даного об'єкту.

```

@Table(name = "water_object")
public class WaterObject {

    @Id
    @GeneratedValue(strategy = GenerationType.UUID)
    private UUID id;

    @Column(precision = 9, scale = 6, nullable = false)
    private BigDecimal latitude;

    @Column(precision = 9, scale = 6, nullable = false)
    private BigDecimal longitude;

    private Double speed;

    private Double direction;

    @Column(name = "active", nullable = false)
    @Builder.Default
    private boolean active = true;
}

```

Рисунок 4.1 – клас об'єкту WaterObject

Дані про об'єкти зберігаються у базі даних та постійно оновлюються під час роботи системи.

4.1.2 Реалізація механізму переміщення об'єктів

Однією з ключових задач сервісу симуляції є моделювання руху водних об'єктів у межах заданої географічної області. Для цього було реалізовано сервіс PositionService, який виконує періодичне оновлення координат усіх активних об'єктів.

Сервіс запускається автоматично за допомогою механізму планувальників Spring Scheduling через інтервал часу, який задається конфігураційним параметром move-freq. Під час кожного запуску система отримує список активних об'єктів та обчислює їх нове положення відповідно до поточних параметрів руху.

На відміну від спрощених моделей переміщення у двовимірній площині, у розробленій системі враховується сферична форма Землі. Це дозволяє отримувати коректні результати навіть при тривалому русі об'єктів та уникнути накопичення похибок у координатах.

Кожний водний об'єкт характеризується такими параметрами:

- географічна широта;
- географічна довгота;
- швидкість руху;
- напрямок руху.

На основі швидкості обчислюється відстань за формулою 4.1, яку об'єкт проходить за один крок симуляції:

$$S = V \cdot \Delta t \quad (4.1)$$

де

S — пройдена відстань;

V — швидкість об'єкта;

Δt — часовий крок симуляції.

Після визначення пройденої відстані виконується обчислення нових координат на поверхні сфери радіусом R . Для цього використовуються формули прямої геодезичної задачі.

Нова широта визначається за формулою 4.2:

$$\varphi_2 = \arcsin(\sin \varphi_1 \cdot \cos(S/R) + \cos \varphi_1 \cdot \sin(S/R) \cdot \cos \theta) \quad (4.2)$$

де

φ_1 — початкова широта;

φ_2 — нова широта;

R — радіус Землі;

θ — напрямок руху.

Нова довгота обчислюється за формулою 4.3:

$$\lambda_2 = \lambda_1 + \arctan((\sin \theta \cdot \sin(S/R) \cdot \cos \varphi_1) / (\cos(S/R) - \sin \varphi_1 \cdot \sin \varphi_2)) \quad (4.3)$$

де

λ_2 — нова довгота;

λ_1 — початкова довгота.

Алгоритм і розрахунки можна побачити на рисунку 4.2.

```
@Scheduled(fixedRateString = "${app.schedules.move-freq}")  & MykytaDanech
@Transactional
public void moveObject() {
    waterObjectRepository.findAll().stream()
        .filter(WaterObject::isActive)
        .forEach( WaterObject obj -> {

            var speed = obj.getSpeed();
            var direction = obj.getDirection();

            var bearingRad = Math.toRadians(direction);

            var distance = speed * TIME_STEP;

            var latRad = Math.toRadians(obj.getLatitude().doubleValue());
            var lonRad = Math.toRadians(obj.getLongitude().doubleValue());

            var newLat = Math.asin(
                Math.sin(latRad) * Math.cos(distance / EARTH_RADIUS) +
                Math.cos(latRad) * Math.sin(distance / EARTH_RADIUS) * Math.cos(bearingRad)
            );

            var newLon = lonRad + Math.atan2(
                Math.sin(bearingRad) * Math.sin(distance / EARTH_RADIUS) * Math.cos(latRad),
                Math.cos(distance / EARTH_RADIUS) - Math.sin(latRad) * Math.sin(newLat)
            );

            obj.setLatitude(BigDecimal.valueOf(Math.toDegrees(newLat)));
            obj.setLongitude(BigDecimal.valueOf(Math.toDegrees(newLon)));
        });
}
```

Рисунок 4.2 – алгоритм руху водних об’єктів

Після виконання розрахунків отримані значення переводяться з радіан у градуси та зберігаються у відповідних полях об’єкта. Оновлені координати автоматично записуються до бази даних завдяки використанню механізму ORM та транзакційного контексту Spring Data JPA.

4.1.3 Реалізація генерації та передачі сигналів

Після оновлення координат водних об’єктів система виконує формування сигналів для подальшої локалізації. Даний процес реалізований у класі

SenderService, який періодично запускається за допомогою механізму Spring Scheduling.

Для кожного водного об'єкта визначаються найближчі станції спостереження. Вибір саме найближчих станцій дозволяє зменшити кількість хибних спрацювань та підвищити точність подальшої локалізації. Пошук виконується шляхом обчислення відстані між об'єктом та кожною станцією з подальшим сортуванням отриманих значень.

Для визначення відстані використано формула гаверсинусів 4.4, 4.5, 4.6:

$$a = \sin^2(\Delta\phi/2) + \cos(\phi_1)\cos(\phi_2)\sin^2(\Delta\lambda/2) \quad (4.4)$$

$$c = 2 \cdot \arctan((\sqrt{a})/(1-\sqrt{a})) \quad (4.5)$$

$$d = R \cdot c \quad (4.6)$$

де

ϕ_1, ϕ_2 — широти двох точок;

λ_1, λ_2 — довготи двох точок;

d — відстань між точками.

R — радіус Землі;

Після визначення найближчих станцій обчислюється азимут від станції до водного об'єкта. Азимут є основною характеристикою сигналу та використовується основним сервісом для подальшого визначення координат об'єкта методом тріангуляції. Також для кожного сигналу обчислюється його сила.

Після формування сигнал зберігається у базі даних сервісу симуляції та поміщається у внутрішню чергу повідомлень. Передача сигналів до основного сервісу виконується асинхронно окремим потоком, що дозволяє уникнути блокування процесів генерації сигналів та забезпечує стабільну роботу системи навіть при збільшенні кількості об'єктів.

4.2 Реалізація сервісу локалізації

Основний сервіс призначений для прийому сигналів від сервісу симуляції, їх подальшої обробки та визначення координат водних об'єктів. На відміну від сервісу симуляції, який генерує дані, даний сервіс виконує роль центру збору та аналізу інформації. Його основною задачею є перетворення набору незалежних сигналів від різних станцій у координати потенційно виявлених об'єктів.

Архітектура сервісу побудована відповідно до багатошарової моделі застосунків Spring Boot та включає контролери, сервіси, репозиторії та допоміжні класи для математичних обчислень.

4.2.1 Прийом та збереження сигналів

Для отримання сигналів від сервісу симуляції реалізовано REST API. Кожен сигнал містить інформацію про станцію спостереження, азимут на об'єкт, силу сигналу та час його генерації.

Після надходження HTTP-запиту сигнал проходить базову перевірку коректності даних та передається до сервісного шару для подальшої обробки.

Збереження сигналів виконується через Spring Data JPA. Для представлення сигналів використовується окрема сутність Signal, яка містить всі необхідні параметри для подальшої локалізації.

Основною перевагою такого підходу є можливість накопичення історії сигналів та їх подальшого аналізу. Крім того, сигнали можуть використовуватись для відлагодження алгоритмів локалізації та оцінки точності роботи системи.

На рисунку 4.3 наведено фрагмент реалізації контролера прийому сигналів.

```

@RestController @MykytaDanech
@RequiredArgsConstructor
@RequestMapping("/api/signals")
public class SignalController {

    private final SignalService signalService;

    @GetMapping @MykytaDanech
    public List<Signal> getAll() { return signalService.getAll(); }

    @PostMapping @MykytaDanech
    public ResponseEntity<String> save(@RequestBody SignalCreateDTO signalDTO) {
        signalDTO.setTimestamp(LocalDateTime.now());
        signalService.save(signalDTO);
        return ResponseEntity.noContent().build();
    }

    @DeleteMapping("/{id}") @MykytaDanech
    public ResponseEntity<String> delete(@PathVariable UUID id) {
        signalService.delete(id);
        return ResponseEntity.noContent().build();
    }
}

```

Рисунок 4.3 – клас з API прийому сигналів SignalController

Представлений контролер є точкою входу для всіх сигналів, що надходять від сервісу симуляції. Його основною задачею є прийом даних через REST API та передача їх до сервісного шару для подальшого збереження й обробки. Завдяки розділенню відповідальностей між контролером та сервісами забезпечується спрощення супроводу програмного коду, підвищення його модульності та можливість подальшого розширення функціональності без зміни механізму взаємодії між сервісами.

4.2.2 Реалізація алгоритму локалізації

Пошук координат виконується окремим планувальником задач, який запускається через задані проміжки часу. Під час кожного запуску система отримує набір актуальних сигналів за останній часовий інтервал та виконує пошук можливих комбінацій сигналів від різних станцій.

Основною задачею алгоритму є відбір лише тих сигналів, які потенційно можуть належати одному об'єкту. Для цього використовується декілька перевірок:

- сигнали повинні надходити від різних станцій;
- сигнали повинні бути отримані приблизно в один і той самий момент часу;
- різниця між силами сигналів не повинна перевищувати встановлений поріг;
- кут між напрямками не повинен бути надто малим.

Після проходження перевірок система виконує обчислення точки перетину азимутів. Отримані координати додатково перевіряються на допустиму відстань від станцій.

У випадку успішного проходження всіх перевірок створюється новий запис про локалізований об'єкт.

Для реалізації даного механізму використовується клас `LocatorService`, який є центральним компонентом сервісу локалізації.

На рисунку 4.4 наведено фрагмент реалізації алгоритму локалізації.

```
for (int i = 0; i < signals.size(); i++) {  
    for (int j = i + 1; j < signals.size(); j++) {  
        var s1 = signals.get(i);  
        var s2 = signals.get(j);  
  
        if (s1.getStation().getId().equals(s2.getStation().getId())) continue;  
        if (s1.getStrength() < MIN_STRENGTH || s2.getStrength() < MIN_STRENGTH) continue;  
  
        var weight = s1.getStrength() * s2.getStrength();  
        if (weight < MIN_STRENGTH_PRODUCT) continue;  
  
        var angleDiff = Math.abs(s1.getAzimuth() - s2.getAzimuth());  
        if (angleDiff > 180) angleDiff = 360 - angleDiff;  
        if (angleDiff < MIN_ANGLE_DIFF) continue;  
  
        Station st1 = s1.getStation();  
        Station st2 = s2.getStation();
```

Рисунок 4.4 – реалізація алгоритму локалізації у класі `LocatorService`

```

var p = intersectBearings(
    st1.getLatitude().doubleValue(), st1.getLongitude().doubleValue(), s1.getAzimuth(),
    st2.getLatitude().doubleValue(), st2.getLongitude().doubleValue(), s2.getAzimuth()
);
if (p == null) continue;

double lat = p[0], lon = p[1];

var d1 = GeoUtils.distance(st1.getLatitude().doubleValue(), st1.getLongitude().doubleValue(), lat, lon);
var d2 = GeoUtils.distance(st2.getLatitude().doubleValue(), st2.getLongitude().doubleValue(), lat, lon);
if (d1 > MAX_OBJECT_DISTANCE || d2 > MAX_OBJECT_DISTANCE) continue;

var expectedD1 = -5000.0 * Math.log(s1.getStrength());
var expectedD2 = -5000.0 * Math.log(s2.getStrength());
if (Math.abs(d1 - expectedD1) > expectedD1 * 0.6) continue;
if (Math.abs(d2 - expectedD2) > expectedD2 * 0.6) continue;

LocalDateTime detectedAt = s1.getReceivedAt().isAfter(s2.getReceivedAt())
    ? s1.getReceivedAt() : s2.getReceivedAt();

candidates.add(new WeightedPoint(lat, lon, weight, detectedAt));
}
}

```

Рисунок 4.5 – продовження алгоритму локалізації сигналів.

Математичний підхід що стоїть за даною реалізацією описаний далі у підрозділі 4.3.

4.2.3 Передача даних клієнтам у режимі реального часу

Для візуалізації роботи системи реалізовано механізм передачі даних через WebSocket-з'єднання. Такий підхід дозволяє миттєво відображати нові сигнали без необхідності постійного опитування сервера.

У системі використовується протокол STOMP поверх WebSocket. Кожна станція має окремий канал підписки, через який клієнт отримує нові сигнали.

Після надходження нового сигналу інформація автоматично передається всім підключеним клієнтам, що дозволяє спостерігати за роботою системи практично без затримок.

Для реалізації механізму передачі використовуються:

- WebSocketConfig;

- SignalStreamService;
- SimpMessagingTemplate;
- STOMP-протокол.

У розділі 5 детальніше пояснення графічного інтерфейсу та приклад роботи користувача з потоковими даними отримуваними через Web Socket.

4.3 Математична модель локалізації об'єктів

Ключовим елементом розробленої системи є алгоритм визначення координат об'єкта за даними, отриманими від декількох станцій спостереження. На відміну від систем активного позиціонування, де об'єкт може передавати власні координати, у даній роботі місцеположення визначається виключно на основі параметрів отриманих сигналів.

Кожна станція спостереження має фіксовані географічні координати та може визначати напрямок на джерело сигналу у вигляді азимута. Отримана інформація використовується для побудови математичної моделі локалізації за методом тріангуляції.

Нехай існують дві станції спостереження: $S1(lat1,lon1)$, $S2(lat2,lon2)$. Для кожної станції відомий азимут на об'єкт – α_1 та α_2 . Азимут визначає напрямок променя, вздовж якого потенційно знаходиться об'єкт.

Для подальших розрахунків азимуту переводяться у радіани формула 4.7:

$$\alpha = \pi \cdot a / 180 \quad (4.7)$$

де

α — азимут у радіанах;

π — математична константа ($\approx 3,14159$);

a — азимут у градусах.

Після цього для кожної станції формується напрямний вектор за формулами 4.8 та 4.9:

$$dx = \cos(\alpha) \quad (4.8)$$

$$dy = \sin(\alpha) \quad (4.9)$$

де

dx — проекція напрямного вектора на вісь широти;

dy — проекція напрямного вектора на вісь довготи;

α — азимут у радіанах.

Отримані вектори використовуються для побудови двох прямих, які виходять із координат станцій у напрямку джерела сигналу.

Для першої станції рівняння прямої задається як 4.10:

$$\begin{aligned} L1: \{ \\ x &= lat1 + t1 * dx1 \\ y &= lon1 + t1 * dy1 \end{aligned} \quad (4.10)$$

де

$lat1$ — широта першої станції;

$lon1$ — довгота першої станції;

$t1$ — параметр прямої;

$dx1, dy1$ — компоненти напрямного вектора першої станції.

Для другої станції 4.11:

$$\begin{aligned} L2: \{ \\ x &= lat2 + t2 * dx2 \\ y &= lon2 + t2 * dy2 \end{aligned} \quad (4.11)$$

де

$lat2$ — широта другої станції;

$lon2$ — довгота другої станції;

$t2$ — параметр прямої;

$dx2, dy2$ — компоненти напрямного вектора другої станції.

Точка перетину цих прямих розглядається як можливе місцеположення об'єкта.

Для визначення факту перетину використовується визначник формула 4.12:

$$D=dx1*dy2-dy1*dx2 \quad (4.12)$$

де

D — визначник системи рівнянь;

$dx1, dy1$ — компоненти напрямного вектора першої прямої;

$dx2, dy2$ — компоненти напрямного вектора другої прямої.

Якщо з формули 4.12 результат такий що правильний вираз 4.13

$$|D| < \epsilon \quad (4.13)$$

де

$|D|$ — модуль визначника;

ϵ — мале додатне число, що задає допустиму похибку.

то прямі вважаються паралельними або майже паралельними, а така пара сигналів відкидається.

У випадку наявності перетину обчислюється параметр за формулою 4.14:

$$t=((lat2-lat1)*dy2-(lon2-lon1)*dx2)/D \quad (4.14)$$

де

t — параметр положення точки на першій прямій;

$lat1, lon1$ — координати першої станції;

$lat2, lon2$ — координати другої станції;

$dx2, dy2$ — компоненти напрямного вектора другої прямої;

D — визначник системи.

Після чого координати точки локалізації визначаються за формулами 4.15 та 4.16:

$$lat=lat1+t*dx1 \quad (4.15)$$

$$lon=lon1+t*dy1 \quad (4.16)$$

де

lat — розрахована широта точки локалізації;

lon — розрахована довгота точки локалізації;

$lat1, lon1$ — координати першої станції;

t — параметр положення точки перетину;

$dx1$, $dy1$ — компоненти напрямного вектора першої прямої.

Отримані координати проходять додаткову перевірку. Для цього обчислюється відстань від знайденої точки до кожної зі станцій. Якщо відстань перевищує максимально допустимий радіус роботи станції, така локалізація вважається некоректною та відкидається.

Для обчислення відстаней використовується формула гаверсинусів 4.4, 4.5, 4.6.

Оскільки сигнали можуть містити похибки або надходити від різних об'єктів, перед виконанням локалізації система виконує попередню фільтрацію. Для цього перевіряється різниця сил сигналів та мінімальний кут між азимутами. Такий підхід дозволяє зменшити кількість хибних локалізацій та підвищити точність визначення координат.

На рисунку 4.6 наведено фрагмент реалізації математичних обчислень у класі GeoUtils.

```
private static final double EARTH_RADIUS = 6371e3; // usage

public static double distance(double lat1, double lon1, double lat2, double lon2) { // 7 usages @MykytaDanech
    double phi1 = Math.toRadians(lat1);
    double phi2 = Math.toRadians(lat2);

    double dPhi = Math.toRadians(lat2 - lat1);
    double dLambda = Math.toRadians(lon2 - lon1);

    double a = Math.sin(dPhi / 2) * Math.sin(dPhi / 2) +
        Math.cos(phi1) * Math.cos(phi2) *
        Math.sin(dLambda / 2) * Math.sin(dLambda / 2);

    double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));

    return EARTH_RADIUS * c;
}

public static double[] destinationPoint(double lat, double lon, double bearingDeg, double distanceMeters) { // 1 usage
    double R = 6371000;
    double delta = distanceMeters / R;
    double theta = Math.toRadians(bearingDeg);
    double phi1 = Math.toRadians(lat);
    double lam1 = Math.toRadians(lon);

    double phi2 = Math.asin(
        Math.sin(phi1) * Math.cos(delta) +
        Math.cos(phi1) * Math.sin(delta) * Math.cos(theta)
    );
    double lam2 = lam1 + Math.atan2(
        Math.sin(theta) * Math.sin(delta) * Math.cos(phi1),
        Math.cos(delta) - Math.sin(phi1) * Math.sin(phi2)
    );

    return new double[] { Math.toDegrees(phi2), Math.toDegrees(lam2) };
}
```

Рисунок 4.6 – Реалізація математичних методів локалізації

Отримані координати є результатом математичної оцінки положення об'єкта на основі доступних сигналів і можуть містити певну похибку, зумовлену неточністю вимірювання азимутів, втратами сигналів або особливостями геометричного розташування станцій. Незважаючи на це, використання декількох перевірок валідності сигналів та додаткових обмежень на допустимі відстані дозволяє значно зменшити кількість хибних локалізацій. Запропонований підхід демонструє можливість визначення приблизного місцеположення об'єктів навіть за відсутності інформації про їх реальні координати та може бути використаний як основа для побудови більш складних систем пасивного спостереження.

Висновки до розділу 4

У цьому розділі було розглянуто процес програмної реалізації основних компонентів системи локалізації водних об'єктів та адміністрування даних гідроакустичних станцій. Детально описано структуру сервісу симуляції, який відповідає за моделювання руху об'єктів, генерацію сигналів та їх передачу до основного сервісу. Було показано механізм обчислення нових координат об'єктів на поверхні Землі з використанням геодезичних формул, а також процес формування сигналів для найближчих станцій спостереження.

Окрему увагу приділено реалізації сервісу локалізації, який виконує прийом, збереження та аналіз отриманих сигналів. Розглянуто механізм відбору валідних сигналів, принципи їх фільтрації та алгоритм пошуку можливих кандидатів для подальшої локалізації. Також описано реалізацію передачі даних у режимі реального часу за допомогою WebSocket, що забезпечує оперативне отримання інформації клієнтами системи.

У підрозділі, присвяченому математичним основам локалізації, було наведено метод визначення координат об'єктів за допомогою тріангуляції. Розглянуто процес побудови променів за азимутами станцій, знаходження точки їх перетину та виконання додаткових перевірок отриманих результатів.

Запропонований підхід дозволяє визначати приблизне місцеположення водних об'єктів без необхідності передачі їх фактичних координат між сервісами.

Таким чином, у межах даного розділу було сформовано повний опис програмної реалізації системи та математичного апарату, який забезпечує функціонування механізму локалізації. Отримані результати стали основою для проведення подальшого тестування та оцінки ефективності розробленого програмного забезпечення.

5 КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС ТА РЕЗУЛЬТАТИ РОБОТИ СИСТЕМИ

Після реалізації серверної частини та алгоритмів локалізації було створено демонстраційний користувацький інтерфейс, який дозволяє візуалізувати роботу системи та спостерігати за процесом локалізації об'єктів у режимі реального часу.

На поточному етапі проект орієнтований насамперед на дослідження алгоритмів обробки сигналів та локалізації об'єктів, тому окремий frontend-застосунок не розроблявся. Для демонстрації функціональності було реалізовано вбудований вебінтерфейс засобами Spring Boot та JavaScript.

Архітектура системи передбачає можливість подальшого підключення окремого frontend-застосунку без зміни серверної логіки. Всі дані передаються через REST API та WebSocket-підключення, що дозволяє у майбутньому реалізувати повноцінний інтерфейс на основі React, Angular або Vue.js.

Для відображення географічних даних використовується бібліотека Leaflet та картографічний сервіс OpenStreetMap.

5.1 Інтерфейс сервісу симуляції

Сервіс симуляції містить окрему сторінку для відображення водних об'єктів та станцій спостереження.

- На карті відображаються:
- гідроакустичні станції;
- водні об'єкти;
- поточне положення кожного об'єкта;
- оновлення координат у режимі реального часу.

Під час роботи системи координати водних об'єктів автоматично оновлюються відповідно до заданих параметрів руху. Це дозволяє спостерігати за процесом симуляції та перевіряти коректність роботи алгоритмів переміщення.

На рисунку 5.1 наведено головне вікно сервісу симуляції.

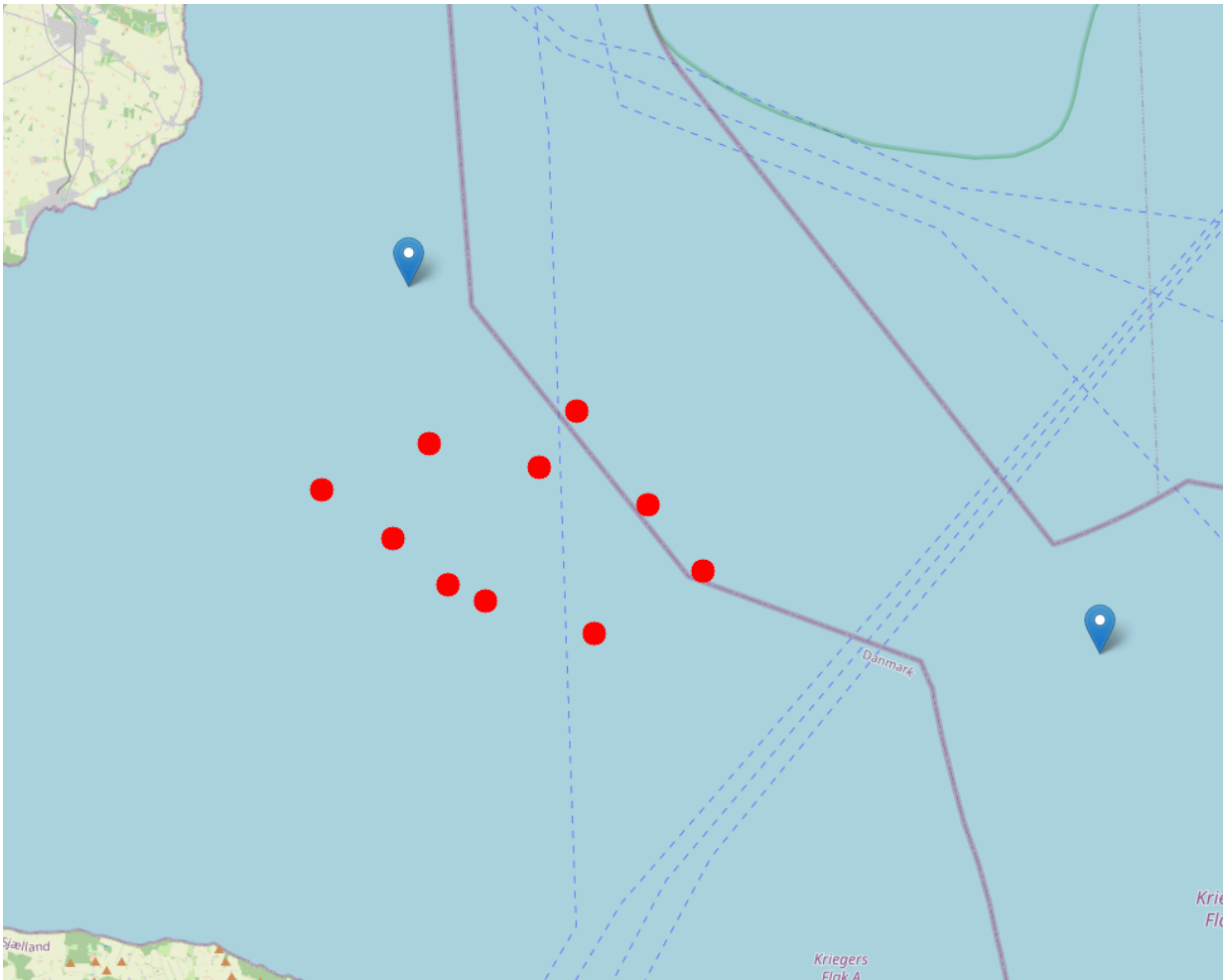


Рисунок 5.1 – Відображення водних об'єктів на карті сервісу симуляції

Таким чином, сервіс симуляції виконує не лише функцію генерації тестових даних, а й дозволяє візуально контролювати процес руху об'єктів та коректність роботи математичної моделі. Отримана інформація використовується як джерело даних для подальшої локалізації об'єктів основним сервісом системи, що забезпечує повний цикл моделювання та обробки сигналів у межах розробленого програмного комплексу.

5.2 Інтерфейс перегляду сигналів станції

Для аналізу сигналів, отриманих окремою станцією спостереження, у системі реалізовано універсальний картографічний інтерфейс, який формується на основі ідентифікатора станції, переданого в URL-запиті.

Після відкриття сторінки система завантажує інформацію про вибрану станцію та історію отриманих нею сигналів. На карті відображається місцеположення станції та азимутальні лінії, побудовані на основі отриманих сигналів. Таким чином користувач може спостерігати напрямки, з яких надходили сигнали, та оцінювати їх просторове розташування.

На карті відображаються координати обраної станції, минулі сигнали отримані станцією, напрямки сигналів у вигляді азимутальних ліній та нові сигнали, що надходять у режимі реального часу та помічаються іншим кольором.

Після відкриття сторінки браузер автоматично встановлює WebSocket-з'єднання із сервером та підписується на канал відповідної станції. При надходженні нового сигналу інформація миттєво відображається на карті без необхідності перезавантаження сторінки.

На рисунку 5.2 наведено приклад відображення сигналів для однієї зі станцій спостереження.

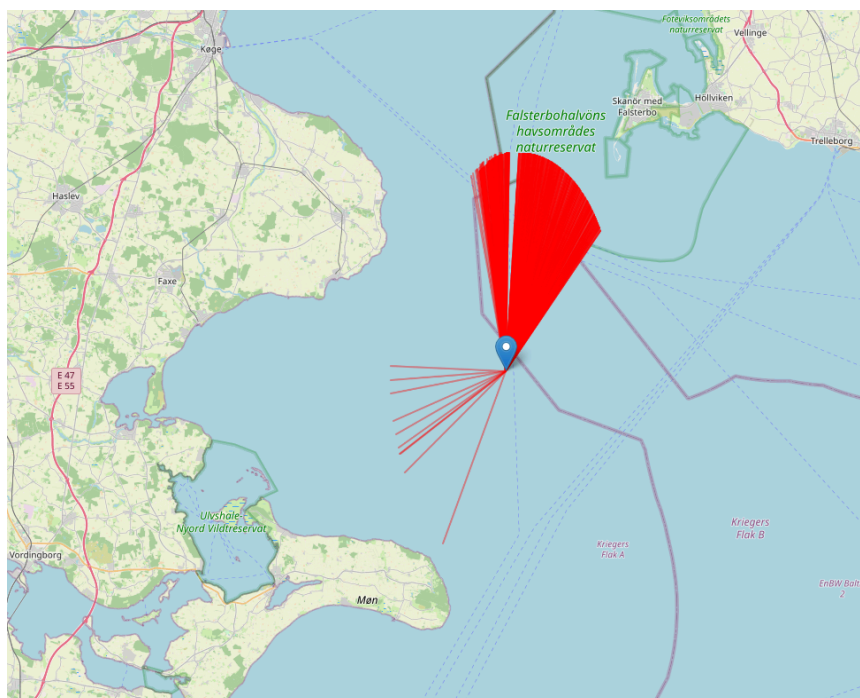


Рисунок 5.2 – Відображення сигналів станції на карті

Представлений інтерфейс дозволяє оцінити роботу окремої станції та простежити за процесом отримання сигналів. Відображення азимутів у режимі реального часу спрощує аналіз роботи алгоритмів локалізації та дає можливість перевіряти коректність передавання даних між сервісами системи.

5.3 Глобальна карта локалізації

Основним елементом інтерфейсу є глобальна карта локалізації. На ній одночасно відображаються усі станції спостереження, результати локалізації об'єктів та актуальні координати знайдених цілей.

Карта автоматично оновлюється через задані проміжки часу та отримує нові дані з основного сервісу.

Відображення результатів локалізації дозволяє оцінити роботу алгоритму триангуляції та перевірити правильність визначення координат об'єктів.

На рисунку 5.3 наведено приклад роботи глобальної карти.

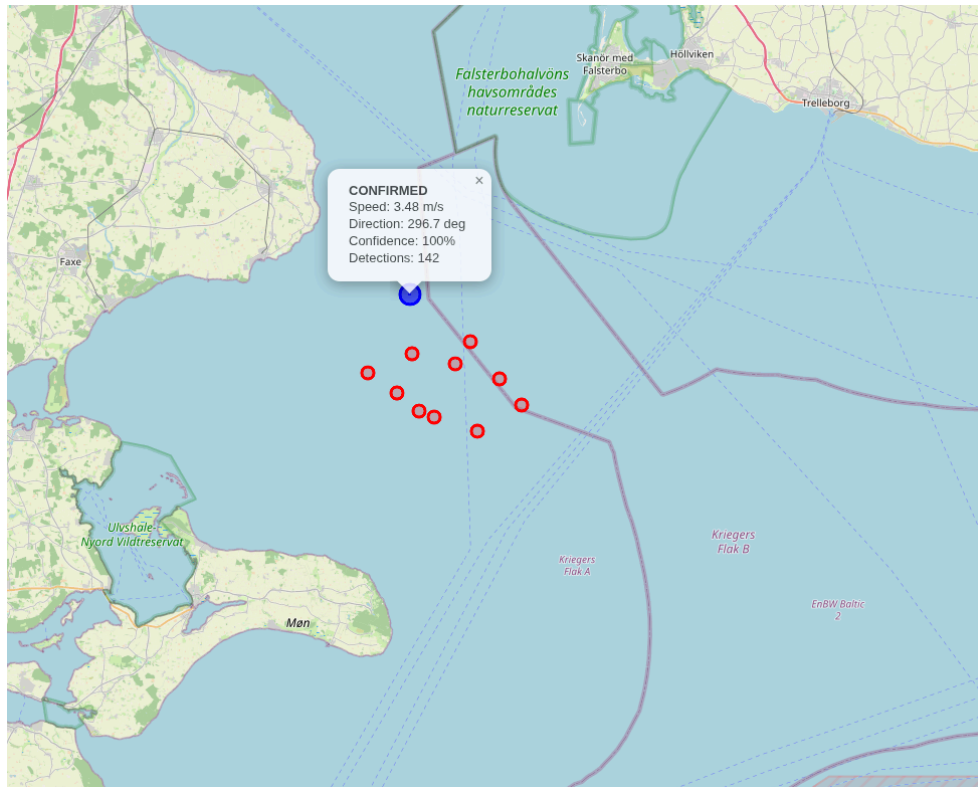


Рисунок 5.3 – Глобальна карта локалізації об'єктів

Глобальна карта є основним засобом контролю результатів роботи алгоритму локалізації. На відміну від інтерфейсу окремої станції, який відображає лише отримані сигнали, дана карта демонструє кінцевий результат обробки цих даних у вигляді знайдених координат об'єктів. Це дозволяє оцінити точність роботи алгоритму тріангуляції та простежити зміну положення локалізованих цілей у часі.

Автоматичне оновлення даних забезпечує відображення актуального стану системи без необхідності перезавантаження сторінки. Завдяки цьому можна спостерігати за усім — від генерації сигналів сервісом симуляції до появи результатів локалізації на карті. Такий підхід значно спрощує тестування системи та аналіз роботи реалізованих алгоритмів у режимі реального часу.

Висновки до розділу 5

У п'ятому розділі було розглянуто реалізований користувацький інтерфейс системи та результати роботи основних компонентів програмного комплексу. Для демонстрації функціональних можливостей було створено вбудований веб інтерфейс, який забезпечує візуалізацію даних, отриманих від сервісу симуляції та сервісу локалізації.

Було описано інтерфейс сервісу симуляції, який дозволяє спостерігати за рухом водних об'єктів у межах заданої області та контролювати коректність роботи математичної моделі переміщення. Також розглянуто механізм відображення сигналів окремої станції спостереження, що дає можливість аналізувати отримані азимути та перевіряти правильність передачі даних між компонентами системи.

Особливу увагу приділено глобальній карті локалізації, яка відображає результати роботи алгоритму визначення координат об'єктів. Саме цей інтерфейс демонструє кінцевий результат функціонування всієї системи — від генерації сигналів до побудови координат локалізованих цілей. Використання WebSocket-з'єднань дозволило реалізувати оновлення даних у режимі реального часу без необхідності перезавантаження сторінок, що значно підвищує зручність спостереження за процесами симуляції та локалізації.

Розроблений інтерфейс має демонстраційний характер та використовується переважно для тестування і дослідження алгоритмів системи. Водночас архітектура програмного комплексу побудована таким чином, що серверна частина повністю відокремлена від засобів відображення даних і взаємодіє через REST API та WebSocket. Це створює можливість подальшого підключення повноцінного frontend-застосунку без необхідності внесення змін до логіки роботи сервісів.

Отримані результати підтверджують працездатність розробленої системи та коректність реалізованих алгоритмів моделювання руху, генерації сигналів і

локалізації водних об'єктів у режимі реального часу. Реалізовані засоби візуалізації дозволяють ефективно аналізувати роботу програмного комплексу та можуть використовуватись як основа для подальшого розвитку системи моніторингу і дослідження методів пасивної локалізації об'єктів.

6 КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС ТА РЕЗУЛЬТАТИ РОБОТИ СИСТЕМИ

Після завершення реалізації програмного комплексу було виконано тестування основних компонентів системи. Метою тестування була перевірка коректності роботи алгоритмів переміщення об'єктів, генерації сигналів, передачі даних між сервісами та локалізації координат. Окрема увага приділялась перевірці стабільності роботи системи в умовах безперервного надходження сигналів та постійного оновлення даних.

Тестування проводиться з використанням фреймворків JUnit 5, Spring Boot Test та Mockito. Для перевірки окремих компонентів застосовувались модульні тести, а для оцінки взаємодії між сервісами — інтеграційне тестування.

6.1 Методика тестування

Тестування системи виконувалося на декількох рівнях:

- модульне тестування окремих сервісів;
- тестування математичних алгоритмів;
- тестування REST API;
- інтеграційне тестування взаємодії сервісів;
- тестування роботи системи в режимі реального часу.

Модульні тести дозволили перевірити коректність роботи окремих компонентів без запуску всієї системи. Для ізоляції тестованих класів використовувались mock-об'єкти та емуляція зовнішніх залежностей.

Інтеграційне тестування виконувалось шляхом запуску сервісів у тестовому середовищі та перевірки передачі даних між ними через HTTP-запити.

6.2 Тестування сервісу симуляції

У межах сервісу симуляції було протестовано основні механізми генерації та обробки даних.

Під час тестування перевірялися:

- створення водних об'єктів;
- коректність збереження координат;
- алгоритм переміщення об'єктів;
- визначення найближчих станцій;
- генерація сигналів;
- робота черги повідомлень;
- асинхронна передача сигналів.

Особлива увага приділялась перевірці математичної моделі переміщення. Для тестових координат були обчислені очікувані значення положення об'єктів після виконання кроку симуляції та порівнянні з результатами роботи сервісу.

На рисунку 6.1 наведено приклад модульного тесту сервісу переміщення об'єктів.

```
1 package com.mykyda.hydrosasim.app.service;
2
3 > import ...
4
5
6 @ExtendWith(MockitoExtension.class) new *
7 class PositionServiceTest {
8
9     @Mock 1 usage
10     private WaterObjectRepository waterObjectRepository;
11
12     @InjectMocks 1 usage
13     private PositionService positionService;
14
15     @Test new *
16     void testMoveObject() {
17         WaterObject obj = WaterObject.builder()
18             .latitude(BigDecimal.valueOf(55.0))
19             .longitude(BigDecimal.valueOf(12.0))
20             .speed(10.0)
21             .direction(90.0)
22             .active(true)
23             .build();
24
25         when(waterObjectRepository.findAll()).thenReturn(List.of(obj));
26
27         positionService.moveObject();
28
29         assertEquals(BigDecimal.valueOf(12.0), obj.getLongitude());
30     }
31 }
```

Рисунок 6.1 – Приклад тестування PositionService

Результати тестування підтвердили правильність роботи алгоритму переміщення та відсутність критичних відхилень від очікуваних значень координат.

6.3 Тестування сервісу локалізації

Основною задачею даного етапу було підтвердження правильності роботи алгоритму локалізації об'єктів.

Під час тестування перевірялись:

- прийом сигналів через REST API;
- збереження сигналів у базі даних;
- пошук актуальних сигналів;
- перевірка валідності комбінацій сигналів;
- обчислення точки перетину азимутів;
- створення локалізованих об'єктів;
- збереження історії локалізації.

Для перевірки алгоритму тріангуляції використовувалися тестові набори сигналів із наперед відомими координатами об'єктів. Отримані результати локалізації порівнювалися з еталонними значеннями.

На рисунку 6.2 наведено приклад тестування алгоритму локалізації.

```

@Test new *
void testLocateSignalAccuracy() {
    Station s1 = Station.builder()
        .id(1L)
        .latitude(BigDecimal.valueOf(50.0))
        .longitude(BigDecimal.valueOf(30.0))
        .build();

    Station s2 = Station.builder()
        .id(2L)
        .latitude(BigDecimal.valueOf(50.0))
        .longitude(BigDecimal.valueOf(30.1))
        .build();

    double strength = 0.36;
    LocalDateTime now = LocalDateTime.now();

    Signal sig1 = Signal.builder()
        .station(s1)
        .azimuth(45.0)
        .strength(strength)
        .receivedAt(now)
        .processed(false)
        .build();

    Signal sig2 = Signal.builder()
        .station(s2)
        .azimuth(315.0)
        .strength(strength)
        .receivedAt(now)
        .processed(false)
        .build();

    when(signalService.getUnprocessed()).thenReturn(List.of(sig1, sig2));
    when(trackedObjectService.findNearestByPrediction(anyDouble(), anyDouble(), anyDouble())).thenReturn(value: null);
    locatorService.locateSignal();
    verify(signalService).markProcessed(anyList());
    ArgumentCaptor<Double> latCaptor = ArgumentCaptor.forClass(Double.class);
    ArgumentCaptor<Double> lonCaptor = ArgumentCaptor.forClass(Double.class);
    verify(trackedObjectService).createNew(latCaptor.capture(), lonCaptor.capture(), anyInt(), any());
    assertEquals("expected: 50.032, latCaptor.getValue(), delta: 0.001");
    assertEquals("expected: 30.05, lonCaptor.getValue(), delta: 0.001");
}

```

Рисунок 6.2 – Приклад тестування LocatorService

У результаті проведених перевірок було встановлено, що алгоритм коректно визначає координати об'єктів за наявності достатньої кількості валідних сигналів від різних станцій спостереження.

6.4 Інтеграційне тестування взаємодії сервісів

Окремим етапом стало тестування взаємодії між сервісом симуляції та сервісом локалізації.

Під час перевірки виконувалась повна послідовність операцій:

1. створення водного об'єкта;
2. переміщення об'єкта;

3. генерація сигналів;
4. передача сигналів до основного сервісу;
5. збереження сигналів;
6. виконання локалізації;
7. створення локалізованого об'єкта.

Тестування показало коректну роботу механізмів обміну даними між сервісами та відсутність втрати інформації під час передачі сигналів.

6.5 Аналіз результатів тестування

Проведене тестування підтвердило працездатність усіх основних компонентів системи.

За результатами перевірки встановлено що алгоритм переміщення об'єктів працює відповідно до математичної моделі, генерація сигналів виконується коректно, передача даних між сервісами відбувається без втрати повідомлень, алгоритм локалізації правильно визначає координати об'єктів, WebSocket-механізм забезпечує передачу даних у режимі реального часу та система стабільно працює при безперервній генерації сигналів.

У процесі тестування критичних помилок, які призводили б до порушення роботи системи, виявлено не було. Отримані результати підтверджують коректність реалізованих алгоритмів та можливість подальшого розвитку програмного комплексу.

Висновки до розділу 6

У останньому розділі було проведено тестування розробленої системи локалізації водних об'єктів. Для перевірки працездатності програмного комплексу використовувалися модульні та інтеграційні тести, що охоплювали як окремі компоненти системи, так і повний цикл взаємодії між сервісами.

Під час тестування було підтверджено коректність роботи алгоритмів переміщення водних об'єктів, генерації сигналів, передачі даних між сервісами та локалізації координат на основі отриманих азимутів. Також перевірено стабільність роботи механізмів збереження даних і передачі повідомлень у режимі реального часу.

Результати тестування показали, що розроблений програмний комплекс успішно виконує поставлені задачі та забезпечує коректну роботу всіх основних функціональних компонентів системи. Отримані результати підтверджують можливість використання реалізованих підходів для подальших досліджень і розвитку систем локалізації об'єктів на основі даних пасивного спостереження.

Додаток покритий тестами на 100 відсотків. Усі сервіси протестовані та показали коректний результат. У сумі для додатку було створено 52 тести, кожен з яких успішно виконаний.

ВИСНОВКИ

У даній дипломній було розроблено серверний додаток адміністрування даних гідроакустичних станцій для локалізації водних об'єктів на основі аналізу сигналів, отриманих від мережі станцій. Основною задачею системи була реалізація механізму визначення приблизних координат об'єктів виключно за азимутами та силою сигналів без прямого доступу до інформації про реальне місцеположення об'єкта.

Під час роботи було реалізовано два окремих мікросервіси: сервіс симуляції руху об'єктів та основний сервіс обробки сигналів. Сервіс симуляції відповідає за створення та переміщення водних об'єктів, генерацію сигналів та передачу їх до основного сервісу. Основний сервіс виконує прийом сигналів, аналіз азимутів, пошук точок перетину та визначення приблизних координат об'єктів.

У процесі розробки було реалізовано систему генерації та передачі сигналів, механізм асинхронної обробки повідомлень, алгоритми географічних обчислень, realtime-відображення даних через WebSocket, інтерактивні карти для візуалізації роботи системи, механізм локалізації об'єктів за сигналами декількох станцій та контейнеризоване розгортання сервісів за допомогою Docker.

Особливу увагу було приділено моделюванню роботи системи в умовах реального часу. Для цього використовувалися планувальники Spring Scheduler, WebSocket-з'єднання та механізми потокової передачі даних. Реалізований підхід дозволив забезпечити автоматичне оновлення інформації на карті без перезавантаження сторінки та наблизити поведінку системи до реальних сценаріїв моніторингу.

Під час розробки було отримано досвід роботи з мікросервісною архітектурою, технологіями Spring Boot, PostgreSQL, Docker, WebSocket, Railway та бібліотеками для картографічної візуалізації. Також було досліджено підходи до організації взаємодії між сервісами, асинхронної обробки подій та побудови realtime-систем.

Розроблена система може використовуватись як основа для подальших досліджень у сфері моніторингу рухомих об'єктів, аналізу сигналів та побудови систем спостереження. Застосунок може бути розширений підтримкою комплексних алгоритмів локалізації, фільтрації шумів, прогнозування траєкторії руху та повноцінного frontend-інтерфейсу для операторів системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Scheduling Tasks // Spring Guides. URL: <https://spring.io/guides/gs/scheduling-tasks> (дата звернення: 04.06.2026).
2. WebSocket API // MDN Web Docs. URL: <https://developer.mozilla.org/en-US/docs/Web/API/WebSocket> (дата звернення: 04.06.2026).
3. Triangulation and Trilateration // The Nature of Geographic Information. Pennsylvania State University. URL: https://courses.ems.psu.edu/natureofgeoinfo/c5_p11.html (дата звернення: 04.06.2026).
4. Urick R. J. *Principles of Underwater Sound*. 3rd ed. New York : McGraw-Hill, 1983. 423 p.
5. Medwin H., Clay C. S. *Fundamentals of Acoustical Oceanography*. Boston : Academic Press, 1998. 712 p.
6. Skolnik M. I. *Introduction to Radar Systems*. 3rd ed. New York : McGraw-Hill, 2001. 747 p.

ДОДАТОК А Лістинг розробленої системи

Код сервісу локалізації

```
package com.mykyda.hydrosa.app.service;

import com.mykyda.hydrosa.app.data.entity.Signal;
import com.mykyda.hydrosa.app.data.entity.Station;
import com.mykyda.hydrosa.app.data.entity.TrackStatus;
import com.mykyda.hydrosa.app.data.entity.TrackedObject;
import com.mykyda.hydrosa.app.utils.GeoUtils;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.math.BigDecimal;
import java.time.Duration;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

@Slf4j
@Service
@RequiredArgsConstructor
public class LocatorService {
```

```

private static final double MAX_OBJECT_DISTANCE = 15_000;
private static final double MIN_ANGLE_DIFF = 20.0;
private static final double DEDUP_ANGLE = 10.0;
private static final double MIN_STRENGTH = 0.1;
private static final double MIN_STRENGTH_PRODUCT = 0.05;
private static final double CLUSTER_RADIUS = 800;
private static final int MIN_CLUSTER_SIZE = 1;
private static final double ASSOCIATION_RADIUS = 2_000;
private static final double LOST_TIMEOUT_SEC = 15;
private static final double EMA_ALPHA = 0.15;

private final SignalService signalService;
private final TrackedObjectService trackedObjectService;
private final ObjectPositionHistoryService historyService;

@Transactional
@Scheduled(fixedRate = 1000)
public void locateSignal() {
    List<Signal> raw = signalService.getUnprocessed();
    if (raw.isEmpty()) return;
    signalService.markProcessed(raw);
    List<Signal> signals = deduplicateByStationAndAzimuth(raw);
    log.info("raw={} dedup={}", raw.size(), signals.size());
    log.info("dedup signals: {}",
        signals.stream()
            .map(s -> "st" + s.getStation().getId() + "@" + String.format("%.1f",
s.getAzimuth()) + " str=" + String.format("%.2f", s.getStrength()))
            .toList());

```

```

List<WeightedPoint> candidates = new ArrayList<>();
for (int i = 0; i < signals.size(); i++) {
    for (int j = i + 1; j < signals.size(); j++) {
        var s1 = signals.get(i);
        var s2 = signals.get(j);
        if (s1.getStation().getId().equals(s2.getStation().getId())) continue;
            if (s1.getStrength() < MIN_STRENGTH || s2.getStrength() <
MIN_STRENGTH) continue;
        var weight = s1.getStrength() * s2.getStrength();
        if (weight < MIN_STRENGTH_PRODUCT) continue;
        var angleDiff = Math.abs(s1.getAzimuth() - s2.getAzimuth());
        if (angleDiff > 180) angleDiff = 360 - angleDiff;
        if (angleDiff < MIN_ANGLE_DIFF) continue;
        Station st1 = s1.getStation();
        Station st2 = s2.getStation();
        var p = intersectBearings(
            st1.getLatitude().doubleValue(), st1.getLongitude().doubleValue(),
s1.getAzimuth(),
            st2.getLatitude().doubleValue(), st2.getLongitude().doubleValue(),
s2.getAzimuth()
        );
        if (p == null) continue;
        double lat = p[0], lon = p[1];
            var d1 = GeoUtils.distance(st1.getLatitude().doubleValue(),
st1.getLongitude().doubleValue(), lat, lon);
            var d2 = GeoUtils.distance(st2.getLatitude().doubleValue(),
st2.getLongitude().doubleValue(), lat, lon);

```

```

        if (d1 > MAX_OBJECT_DISTANCE || d2 > MAX_OBJECT_DISTANCE)
continue;

        var expectedD1 = -5000.0 * Math.log(s1.getStrength());
        var expectedD2 = -5000.0 * Math.log(s2.getStrength());
        if (Math.abs(d1 - expectedD1) > expectedD1 * 0.6) continue;
        if (Math.abs(d2 - expectedD2) > expectedD2 * 0.6) continue;
        LocalDateTime detectedAt = s1.getReceivedAt().isAfter(s2.getReceivedAt())
            ? s1.getReceivedAt() : s2.getReceivedAt();

        candidates.add(new WeightedPoint(lat, lon, weight, detectedAt));
    }
}
log.info("candidates={}", candidates.size());
if (candidates.isEmpty()) return;
List<List<WeightedPoint>> clusters = cluster(candidates);
                                log.info("clusters={}    sizes={}",    clusters.size(),
clusters.stream().map(List::size).toList());
trackedObjectService.markLostIfStale(LOST_TIMEOUT_SEC);
for (List<WeightedPoint> cluster : clusters) {
    if (cluster.size() < MIN_CLUSTER_SIZE) continue;
    var centroid = weightedCentroid(cluster);
    double lat = centroid[0], lon = centroid[1];
    LocalDateTime clusterTime = cluster.stream()
        .map(p -> p.detectedAt)
        .max(LocalDateTime::compareTo)
        .orElse(LocalDateTime.now());
    TrackedObject match = trackedObjectService.findNearestByPrediction(lat, lon,
ASSOCIATION_RADIUS);
    if (match != null) {

```

```

        updateTrackedObject(match, lat, lon, clusterTime);
        trackedObjectService.save(match);
        log.info("updated {} -> {},{} cluster={}", match.getId(), lat, lon,
cluster.size());
    } else {
        trackedObjectService.createNew(lat, lon, cluster.size(), clusterTime);
        log.info("created new -> {},{} cluster={}", lat, lon, cluster.size());
    }
}
}
}

```

```

private void updateTrackedObject(TrackedObject obj, double newLat, double
newLon, LocalDateTime detectedAt) {
    var dt = Duration.between(obj.getLastSeen(), detectedAt).toMillis() / 1000.0;
    if (dt > 0 && dt < 60) {
        var fromLat = obj.getLatitude().doubleValue();
        var fromLon = obj.getLongitude().doubleValue();
        var dist = GeoUtils.distance(fromLat, fromLon, newLat, newLon);
        if (dist > 0.3) {
            var speed = dist / dt;
            var bearing = GeoUtils.bearing(fromLat, fromLon, newLat, newLon);
            if (obj.getEstimatedSpeed() == null) {
                obj.setEstimatedSpeed(speed);
                obj.setEstimatedDirection(bearing);
            } else {
                var speedRatio = speed / obj.getEstimatedSpeed();
                if (speedRatio > 0.3) {
                    obj.setEstimatedSpeed(EMA_ALPHA * speed + (1 - EMA_ALPHA) *
obj.getEstimatedSpeed());

```

```

        obj.setEstimatedDirection(EMA_ALPHA * bearing + (1 -
EMA_ALPHA) * obj.getEstimatedDirection());
    }
}
}
}
obj.setLatitude(BigDecimal.valueOf(newLat));
obj.setLongitude(BigDecimal.valueOf(newLon));
obj.setLastSeen(detectedAt);
obj.setDetectionCount(obj.getDetectionCount() + 1);
obj.setConfidence(Math.min(1.0, obj.getDetectionCount() / 10.0));
obj.setStatus(TrackStatus.CONFIRMED);
historyService.record(obj.getId(), newLat, newLon, detectedAt);
}

```

```

private static class WeightedPoint {
    final double lat, lon, weight;
    final LocalDateTime detectedAt;
    WeightedPoint(double lat, double lon, double weight, LocalDateTime detectedAt)
{
    this.lat = lat;
    this.lon = lon;
    this.weight = weight;
    this.detectedAt = detectedAt;
}
}

```

```

private List<Signal> deduplicateByStationAndAzimuth(List<Signal> raw) {
    Map<Long, List<Signal>> byStation = raw.stream()

```

```

        .collect(Collectors.groupingBy(s -> s.getStation().getId()));
List<Signal> result = new ArrayList<>();
for (List<Signal> stationSignals : byStation.values()) {
    stationSignals.sort(Comparator.comparing(Signal::getReceivedAt).reversed());
    List<Signal> kept = new ArrayList<>();
    for (Signal s : stationSignals) {
        if (s.getStrength() < MIN_STRENGTH) continue;
        var duplicate = kept.stream().anyMatch(k -> {
            var diff = Math.abs(k.getAzimuth() - s.getAzimuth());
            if (diff > 180) diff = 360 - diff;
            return diff < DEDUP_ANGLE;
        });
        if (!duplicate) kept.add(s);
    }
    result.addAll(kept);
}
return result;
}

```

```

private double[] intersectBearings(
    double lat1, double lon1, double az1,
    double lat2, double lon2, double az2) {
    var phi1 = Math.toRadians(lat1);
    var phi2 = Math.toRadians(lat2);
    var cos1 = Math.cos(Math.toRadians(az1));
    var sin1 = Math.sin(Math.toRadians(az1));
    var cos2 = Math.cos(Math.toRadians(az2));
    var sin2 = Math.sin(Math.toRadians(az2));

```

```

var cosLat = Math.cos((phi1 + phi2) / 2);
var dNorth = (lat2 - lat1) * 111_320.0;
var dEast = (lon2 - lon1) * 111_320.0 * cosLat;
var det = sin1 * (-cos2) - cos1 * (-sin2);
if (Math.abs(det) < 1e-10) return null;
var t1 = (dEast * (-cos2) - dNorth * (-sin2)) / det;
var t2 = (sin1 * dNorth - cos1 * dEast) / det;
if (t1 < 0 || t2 < 0) return null;
var latI = lat1 + (t1 * cos1) / 111_320.0;
var lonI = lon1 + (t1 * sin1) / (111_320.0 * cosLat);
return new double[]{latI, lonI};
}

private List<List<WeightedPoint>> cluster(List<WeightedPoint> points) {
    List<List<WeightedPoint>> clusters = new ArrayList<>();
    for (WeightedPoint p : points) {
        var bestDist = Double.MAX_VALUE;
        List<WeightedPoint> best = null;
        for (List<WeightedPoint> c : clusters) {
            var centroid = weightedCentroid(c);
            var d = GeoUtils.distance(centroid[0], centroid[1], p.lat, p.lon);
            if (d < CLUSTER_RADIUS && d < bestDist) {
                bestDist = d;
                best = c;
            }
        }
        if (best != null) best.add(p);
        else {
            List<WeightedPoint> nc = new ArrayList<>();

```

```

        nc.add(p);
        clusters.add(nc);
    }
}
return clusters;
}

private double[] weightedCentroid(List<WeightedPoint> cluster) {
    var totalWeight = cluster.stream().mapToDouble(p -> p.weight).sum();
    if (totalWeight == 0) {
        return new double[]{
            cluster.stream().mapToDouble(p -> p.lat).average().orElse(0),
            cluster.stream().mapToDouble(p -> p.lon).average().orElse(0)
        };
    }
    return new double[]{
        cluster.stream().mapToDouble(p -> p.lat * p.weight).sum() / totalWeight,
        cluster.stream().mapToDouble(p -> p.lon * p.weight).sum() / totalWeight
    };
}
}

```

ДОДАТОК В Презентація

Вставляєте рисунками без підписів всі слайди своєї презентації, можна по два на сторінку.