

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Інтелектуальна система візуалізації текстової
інформації про повітряні цілі під час повітряної тривоги»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

*Кваліфікаційна робота містить результати власних досліджень. Використання
ідей, результатів і текстів інших авторів мають посилання
на відповідне джерело*

(підпис) Євгеній КРАВЧЕНКО

Виконав: здобувач вищої освіти групи ПД-44

Євгеній КРАВЧЕНКО

Керівник: Ірина ЗАМРІЙ

*д-р техн. наук,
проф.*

Рецензент: _____

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення

_____ Ірина ЗАМРІЙ

« ____ » _____ 2026 р.

**ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Кравченку Євгенію Юрійовичу

1. Тема кваліфікаційної роботи: «Інтелектуальна система візуалізації текстової інформації про повітряні цілі під час повітряної тривоги»

керівник кваліфікаційної роботи д-р техн. наук, професор Ірина ЗАМРІЙ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «29» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи обробки природної мови та парсингу неструктурованого тексту, специфіка збору даних з месенджера Telegram, технічна документація технологій платформи .NET 9 (C#), протоколу SignalR, бібліотек візуалізації MapLibre GL JS та deck.gl, концепції побудови систем реального часу та хмарної інфраструктури Microsoft Azure.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд та аналіз існуючих методів моніторингу і систем візуалізації повітряних загроз із відкритих джерел.

2. Обґрунтування вибору технологічних засобів, фреймворків та стеку розробки системи.

3. Проектування архітектури системи, алгоритмів конвеєра обробки повідомлень, геонормалізації та трекінгу цілей.

4. Програмна реалізація, тестування системи та демонстрація результатів роботи клієнтського застосунку і панелі адміністратора.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма послідовності обробки повідомлення.
5. Діаграма зв'язків основних моделей даних.
6. Екранні форми.
7. Апробація результатів дослідження.

6. Дата видачі завдання «21» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	21.02 - 28.02.2026	
2	Аналіз та дослідження існуючих аналогів	01.03 - 07.03.2026	
3	Огляд алгоритмів парсингу тексту та обґрунтування технологічного стеку	08.03 - 15.03.2026	
4	Проектування архітектури системи та конвеєра обробки повідомлень	16.03 - 29.03.2026	
5	Програмна реалізація серверної (бекенд) та клієнтської (фронтенд) частин системи	30.03 - 12.04.2026	
6	Тестування застосунку на складних повідомленнях та виправлення помилок	13.04 - 19.04.2026	
7	Оформлення роботи: вступ, розділи, висновки, реферат	20.04 - 03.05.2026	
8	Розробка демонстраційних матеріалів	04.05 - 10.05.2026	
9	Попередній захист роботи	11.05 - 22.05.2026	
10	Подання роботи	23.05 - 30.05.2026	

Здобувач вищої освіти

_____ (підпис)

Євгеній КРАВЧЕНКО

Керівник
кваліфікаційної роботи

_____ (підпис)

Ірина ЗАМРІЙ

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 68 стор., 2 табл., 27 рис., 22 джерел.

Мета роботи – пришвидшення обробки текстової інформації та інформування користувачів шляхом розробки системи інтелектуальної трансформації неструктурованих текстових даних у динамічні картографічні об'єкти щодо переміщення повітряних загроз у реальному часі.

Об'єкт дослідження – процес автоматизованої обробки та візуалізації неструктурованих даних про переміщення повітряних загроз у реальному часі.

Предмет дослідження – методи та алгоритми функціонування інтелектуальної системи із застосуванням гібридного парсингу та потокової передачі даних.

Короткий зміст роботи: Проаналізовано специфіку збору даних з Telegram-каналів та існуючі аналоги («Neptun», «Dimar», «VARTA AI»), виявлено їх критичні недоліки: затримки та галюцинації штучного інтелекту при обробці координат. Обґрунтовано стек технологій: .NET 9, SignalR, MapLibre GL JS, deck.gl, Azure та Docker. Спроектовано систему «КОНТУР» для перетворення неструктурованого тексту в об'єкти на мапі в реальному часі. Розроблено конвеєр обробки повідомлень, модель трекінгу та модуль нормалізації геоданих. Проведено наскрізне тестування на складному багатовекторному повідомленні.

Сфера використання – цивільний захист та подальший розвиток OSINT-систем моніторингу повітряного простору.

КЛЮЧОВІ СЛОВА: МОНІТОРИНГ ПОВІТРЯНИХ ЗАГРОЗ, ВІЗУАЛІЗАЦІЯ В РЕАЛЬНОМУ ЧАСІ, TELEGRAM MINI APP, СИНТАКСИЧНИЙ АНАЛІЗ, SIGNALR, ШТУЧНИЙ ІНТЕЛЕКТ, .NET 9, C#, MICROSOFT AZURE, DOCKER, ІНТЕЛЕКТУАЛЬНА СИСТЕМА.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	9
ВСТУП	10
1 АНАЛІЗ МЕТОДІВ МОНИТОРИНГУ ТА ВІЗУАЛІЗАЦІЇ ПОВІТРЯНИХ ЗАГРОЗ	12
1.1 Специфіка збору даних про повітряні загрози з відкритих джерел	12
1.2 Порівняльний аналіз існуючих програмних рішень	13
1.2.1 Аналіз системи «Neptun»	13
1.2.2 Аналіз системи «Dimar»	15
1.2.3 Аналіз системи «VARTA AI»	17
1.2 Критичні недоліки аналогів та обґрунтування розробки.....	18
1.3 Визначення вимог до системи «КОНТУР»	20
1.3.1 Функціональні вимоги.....	20
1.4.2 Нефункціональні вимоги.....	21
2 ОБґРУНТУВАННЯ ТЕХНІЧНИХ ЗАСОБІВ ТА СТЕКУ РОЗРОБКИ	22
2.1 Середовище розробки та мова програмування	22
2.1.1 Visual Studio 2022.....	22
2.1.2 C# (.NET 9)	22
2.2 Фреймворки та комунікаційна логіка	23
2.2.1 SignalR (Real-time зв'язок).....	23
2.2.2 REST API та безпека (JWT)	23
2.3 Спеціалізовані компоненти та візуалізація	24
2.3.1 Hunspell та логіка GeoNormalizer	24
2.3.2 MapLibre GL JS та deck.gl	24
2.4 Механізми автентифікації та безпеки доступу	25
2.5 Контейнеризація та розгортання системи	26
2.6 Допоміжні засоби розробки та контролю версій.....	27
2.7 Інструменти інтелектуальної обробки (AI)	28
2.8 Організація зберігання та обробки даних.....	28
3 ПРОЄКТУВАННЯ ТА АРХІТЕКТУРА СИСТЕМИ «КОНТУР»	29
3.1 Доменний рівень	29
3.2 Рівень застосунку	32

3.3 Алгоритмічне забезпечення конвеєра обробки повідомлень	35
3.3.1 Первинна обробка повідомлень	36
3.3.2 Семантична токенизація тексту.....	37
3.3.3 Фільтрація інформаційного шуму.....	38
3.3.4 Класифікація складності структури (Complexity Classification) .	41
3.3.5 Декомпозиція складних подій (AI Splitter та Event Splitter).....	42
3.3.6 Геопросторовий аналіз та нормалізація	44
3.3.7 Генерація сигнатури повідомлення та пошук шаблонів.....	46
3.3.8 Інтелектуальний синтез та валідація правил	47
3.3.9 Синтаксичний розбір та формування об'єкта загрози	49
3.3.10 Семантичне поєднання цілей.....	50
3.3.11 Ідентифікація та супроводження цілей (Tracking Service)	52
3.3.12 Репозиторій активних загроз та відкладене оновлення	53
3.3.13 Клієнтська візуалізація та інтерполяція руху	55
3.4 Інфраструктурний рівень	56
3.4.1 Передача даних	56
3.4.2 Збереження даних	57
3.4.3 Контейнеризація Docker.....	58
3.4.4 Платформа розгортання	58
3.4.5 Інфраструктурна безпека	58
3.4.6 Підсистема адміністрування та моніторингу.....	59
4 ДЕМОНСТРАЦІЯ РЕЗУЛЬТАТІВ ТА ТЕСТУВАННЯ.....	61
4.1 Демонстрація клієнтського веб-застосунку	61
4.2 Демонстрація адміністраторського веб-сайту	67
4.3 Тестування системи	73
4.4 Порівняння з існуючими системами	76
ВИСНОВКИ.....	77
ПЕРЕЛІК ПОСИЛАНЬ	79
ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ	81
ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ.....	90

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface

DTO – Data Transfer Object

JSON – JavaScript Object Notation

JWT – JSON Web Token

LLM – Large Language Model (Великі мовні моделі)

OSINT – Open-Source Intelligence (Розвідка на основі відкритих джерел)

REST – Representational State Transfer

UI – User Interface (Інтерфейс користувача)

UX – User Experience (Досвід користувача)

VPS – Virtual Private Server (Віртуальний виділений сервер)

БпЛА – Безпілотний літальний апарат

КАБ – Керована авіаційна бомба

ОЗУ – Оперативний запам'ятовуючий пристрій

СУБД – Система управління базами даних

ІІ – Штучний інтелект

ВСТУП

Актуальність теми: Забезпечення оперативного інформування цивільного населення про повітряні загрози вимагає мінімізації часу на обробку вхідних даних. Наразі основним джерелом такої інформації виступають текстові моніторингові канали в месенджері Telegram. Користувач змушений вручну аналізувати неструктуровані повідомлення для просторової орієнтації. Таке когнітивне навантаження критично збільшує час прийняття рішень. Відповідно, виникає необхідність автоматизації процесу. У роботі запропоновано програмну систему «КОНТУР». Вона автоматично трансформує неструктурований текстовий потік у геопросторові об'єкти на інтерактивній мапі. Картографічний інтерфейс розроблено за принципами мінімалізму для пришвидшення ідентифікації типу та вектора руху цілі. Застосування сучасних мережевих протоколів гарантує мінімальну затримку передачі. Система працює в режимі реального часу.

Об'єктом дослідження є процес автоматизованої обробки та візуалізації неструктурованої текстової інформації про просторові переміщення повітряних загроз у реальному часі.

Предметом дослідження є методи та алгоритми функціонування інтелектуальної системи із застосуванням гібридного парсингу та потокової передачі даних.

Мета роботи пришвидшення обробки текстової інформації та інформування користувачів шляхом розробки системи інтелектуальної трансформації неструктурованих текстових даних у динамічні картографічні об'єкти щодо переміщення повітряних загроз у реальному часі.

Для досягнення поставленої мети виконано аналіз існуючих рішень. Це дозволило ідентифікувати критичні вразливості аналогів, зокрема генеративні помилки («галюцинації») нейромереж під час обробки координат. На основі отриманих результатів обґрунтовано вибір технологічного стека. Серверну логіку реалізовано на платформі .NET 9 з використанням технології SignalR [1]. Це

забезпечує стабільність високонавантаженого бекенду. Модуль валідації правил розбору інтегровано з Google AI Studio. Хмарну інфраструктуру розгорнуто в середовищі Microsoft Azure [10], а мережевий екран Cloudflare налаштовано для нейтралізації бот-трафіку [15]. Особливу увагу в роботі приділено розробці математичної моделі трекінгу на базі стандартизованих об'єктів. Алгоритм враховує фізичні параметри швидкості та дистанції, що виключає аномальне зміщення («телепортацію») маркерів. Просторовий резолвер адаптовано для коректної обробки транзитних регіональних зон та цілей над морськими акваторіями.

Наукова новизна роботи полягає у розробці гібридної моделі парсингу, яка поєднує детерміновану токенизацію на основі сигнатур із залученням ШІ лише як «вчителя» для генерації нових правил. Це повністю виключає появу некоректних координат та галюцинацій, які притаманні системам, що використовують штучний інтелект для отримання даних. Також новизна полягає у застосуванні протоколу SignalR для забезпечення справжнього Real-time зв'язку без циклічних опитувань сервера чого не можуть запропонувати існуючі системи.

Практична значущість полягає у створенні функціонального інструменту цивільного захисту, який забезпечує оновлення даних протягом 0.1–3 секунд та стабільно працює в умовах високого навантаження. Реалізована система мінімізує час на інтерпретацію загроз та може бути використана як база для подальшого розвитку систем моніторингу повітряного простору.

1 АНАЛІЗ МЕТОДІВ МОНІТОРИНГУ ТА ВІЗУАЛІЗАЦІЇ ПОВІТРЯНИХ ЗАГРОЗ

1.1 Специфіка збору даних про повітряні загрози з відкритих джерел

Сьогодні відкриті текстові канали в месенджері Telegram слугують основним джерелом оперативних даних про переміщення повітряних цілей. Таких цілей, як безпілотні літальні апарати (БПЛА), керовані авіаційні бомби (КАБ) та ракетне озброєння. Вибір цієї платформи обґрунтовується мінімальною мережевою затримкою розповсюдження повідомлень і прямим доступом до зведень від підрозділів візуально-акустичного спостереження. Відповідно, система отримує вхідні масиви даних майже в реальному часі.

Процес агрегації зазначеного потоку інформації має низку технічних особливостей. Зокрема, виділено три ключові фактори:

- неструктурованість інформації, оскільки вхідні повідомлення формуються природною мовою з використанням специфічного сленгу, нерегламентованих аббревіатур та варіативних топонімів, через що модуль синтаксичного аналізу стикається з проблемою розпізнавання сутностей (ідентичні події часто описуються різними конструкціями [5]), а зіставлення текстових шаблонів потребує попередньої нормалізації даних;

- висока динаміка оновлень, де підсистема збору даних працює в режимі безперервного моніторингу, і навіть незначна затримка обробки критично знижує актуальність координат для швидкісних цілей, що вимагає суворої оптимізації швидкодії алгоритму;

- просторова невизначеність, адже текстовий потік зазвичай містить лише відносні орієнтири (вектори, назви регіонів або перетини адміністративних меж), трансформація яких у математично точні географічні координати вимагає

застосування спеціалізованих логічних обробників, а конвертер координат має враховувати похибку описових даних.

Опрацювання первинних масивів інформації вручну створює надмірне когнітивне навантаження. Кінцевий користувач змушений самотійно агрегувати десятки несинхронізованих повідомлень для побудови просторової траєкторії руху загрози. Це суттєво уповільнює процес аналізу.

З огляду на це, просте дублювання тексту є технічно недоцільним. У роботі обґрунтовано розробку автоматизованої програмної архітектури. Система самотійно фільтрує інформаційний шум, валідує параметри та перетворює їх у стандартизовані об'єкти передачі даних (DTO). Такий підхід гарантує коректне відображення просторової логіки на картографічному інтерфейсі.

1.2 Порівняльний аналіз існуючих програмних рішень

На сьогоднішній день існує декілька веб-орієнтованих систем, що намагаються вирішити задачу візуалізації інформації щодо повітряних цілей. Проте кожна з них має свої архітектурні та візуальні особливості.

1.2.1 Аналіз системи «Neptun»

Більш ефективним засобом є спеціалізоване програмне забезпечення, що базується на візуалізації цілей на інтерактивних картах. Одним із прикладів такої системи є мапа «Neptun» [20], яка представляє собою веб-застосунок для відстеження руху повітряних об'єктів. Проте архітектура даної системи має низку суттєвих недоліків:

- ненадійність обробки даних через пряму залежність від «сирого» виводу великих мовних моделей (LLM), що часто призводить до «галюцинацій» координат та появи некоректних міток на мапі;
- відсутність жорсткої валідації даних, що робить систему вразливою до помилок у вхідних текстах моніторингових джерел;

– використання застарілого протоколу синхронізації HTTP Polling, через що цілі «телепортуються» по мапі кожні кілька секунд, не дотримуючись плавного курсу. Приклад інтерфейсу системи «Neptun» наведено на рисунку 1.1.

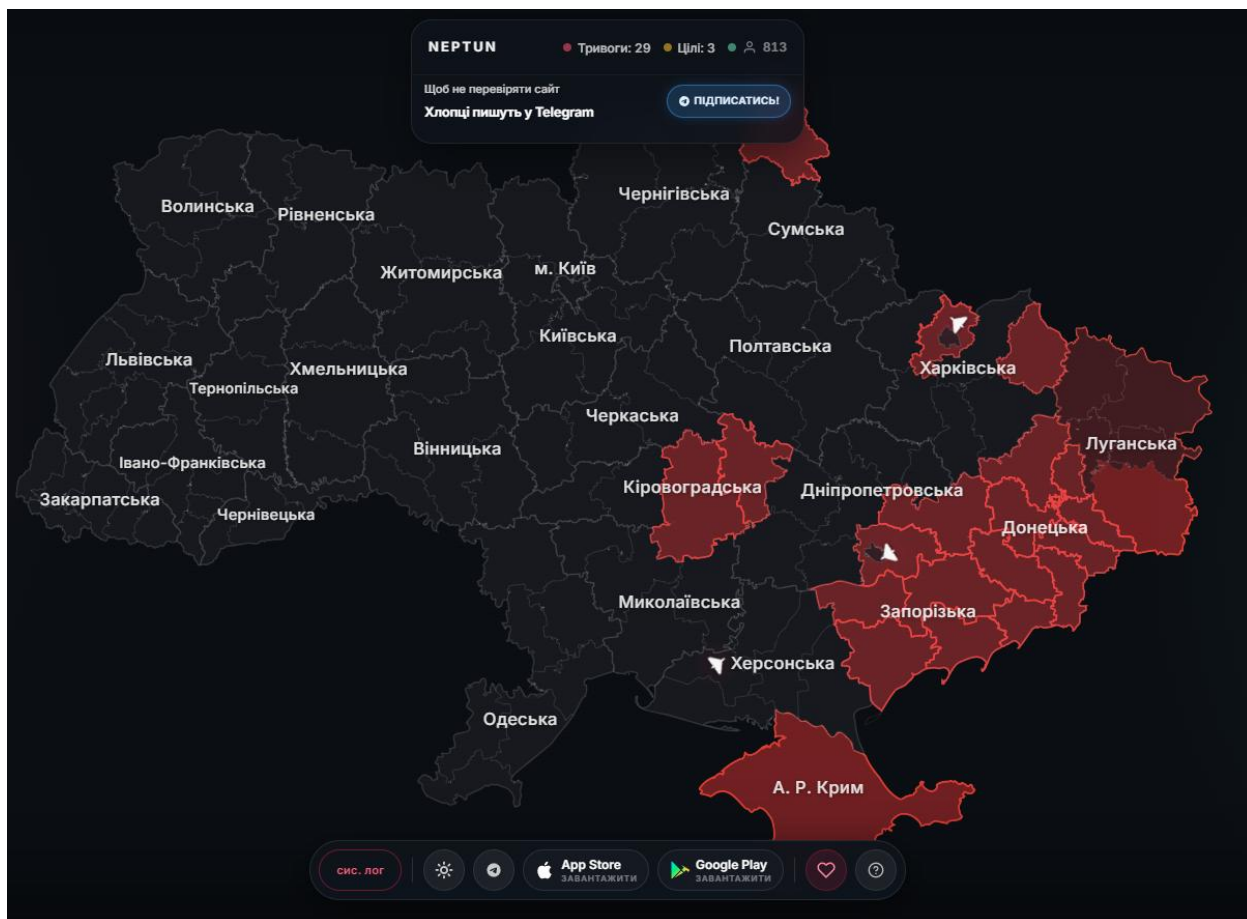


Рис. 1.1 Приклад інтерфейсу системи «Neptun»

Переваги використання системи «Neptun» для візуалізації інформації щодо повітряних цілей:

- доступність через звичайний веб-браузер без необхідності встановлення додаткового ПЗ;
- наявність супутникової підкладки для кращого орієнтування на місцевості при великих масштабах;
- відображення загальної кількості активних тривог та цілей у верхній панелі.

До недоліків використання системи «Neptun» можна віднести наступне:

- неможливість приховати інформаційну «шторку», яка перекриває значну частину робочої області карти та відволікає увагу;
- нечитабельність назв міст при масштабуванні через накладання супутникового ландшафту та міток цілей;
- відсутність механізму автоматичного видалення «мертвих» міток, що знаходяться поза зонами активних тривог;
- низька точність прогнозування траєкторій через імітацію руху випадковою зміною вектора міток.

1.2.2 Аналіз системи «Dimar»

Іншим поширеним інструментом для моніторингу інформації щодо повітряних цілей є веб-орієнтована система «Dimar» [21]. На відміну від аналогів, цей ресурс фокусується на глибинному парсингу текстових повідомлень із залученням багатоетапної обробки штучним інтелектом. Проте архітектурна побудова системи та обрані методи візуалізації мають суттєві недоліки, що заважають оперативному сприйняттю інформації.

Ключовою особливістю «Dimar» є послідовна ШІ-обробка: система використовує окремі запити до LLM для визначення типу цілі, її позиції та вектору руху. Це створює величезні затримки на бекенді, роблячи рішення архітектурно неефективним і дорогим.

Також система характеризується високою інертністю: інтервал оновлення даних (пулінг) складає 15–25 секунд, що є критично великим для динамічних об'єктів, які за цей час можуть змінити позицію на 5–10 км. Через таку затримку на мапі спостерігається ефект телепортації — мітки просто змінюють координати без дотримання плавного курсу.

Приклад зовнішнього вигляду мапи «Dimar» наведено на рисунку 1.2.

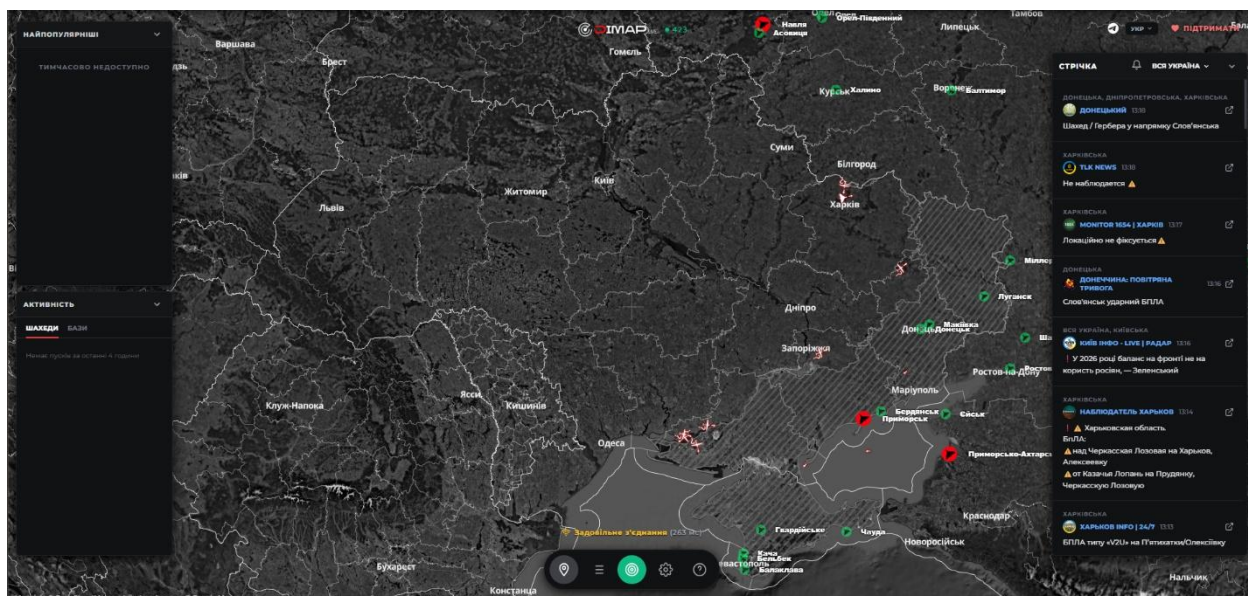


Рис. 1.2 Приклад інтерфейсу системи «DImar»

Переваги використання інформаційної системи «DImar»:

- наявність детальної бічної панелі зі стрічкою новин, що дозволяє паралельно з картою читати першоджерела інформації;
- можливість фільтрації активності за типами загроз (наприклад, окремо «Шаледи» чи «Бази»);
- наявність реактивних запобіжників, що намагаються відсіювати некоректні дані.

До недоліків системи «DImar» можна віднести наступне:

- неефективність обробки даних через багатоетапний парсинг, що призводить до появи «фантомних» цілей, які з'являються на мапі та видаляються лише після візуального відображення, що створює хибні тривоги;
- система намагається імітувати живу карту через випадкову зміну вектора міток, що не відповідає реальному руху об'єктів;
- постійно активна супутникова підкладка у поєднанні зі штрихуванням зон тривоги та численними іконками створює візуальний шум;
- інформаційне перевантаження через масивні бічні панелі, що перекривають до 40% робочої області карти, відволікаючи увагу від пріоритетних об'єктів у критичних ситуаціях

1.2.3 Аналіз системи «VARTA AI»

Третім об'єктом аналізу є сервіс «VARTA AI» [22]. Система позиціонується як інтелектуальний дашборд, проте фактично вона є агрегатором новин із мінімальною статичною візуалізацією на мапі.

Головна особливість інтерфейсу - панель «ШІ-аналітик», яка дублює повідомлення з моніторингових каналів. Проте тотальна залежність від ШІ-парсингу без жорстких алгоритмічних фільтрів робить систему «сліпою»: через галюцинації нейромережі на мапі часто виникають фантомні мітки з некоректними координатами. При цьому рух цілей та вектори їхнього напрямку повністю відсутні - об'єкт просто фіксується в точці, що позбавляє користувача можливості оцінити динаміку загрози. Приклад зовнішнього вигляду інтерфейсу «VARTA AI» наведено на рисунку 1.3.

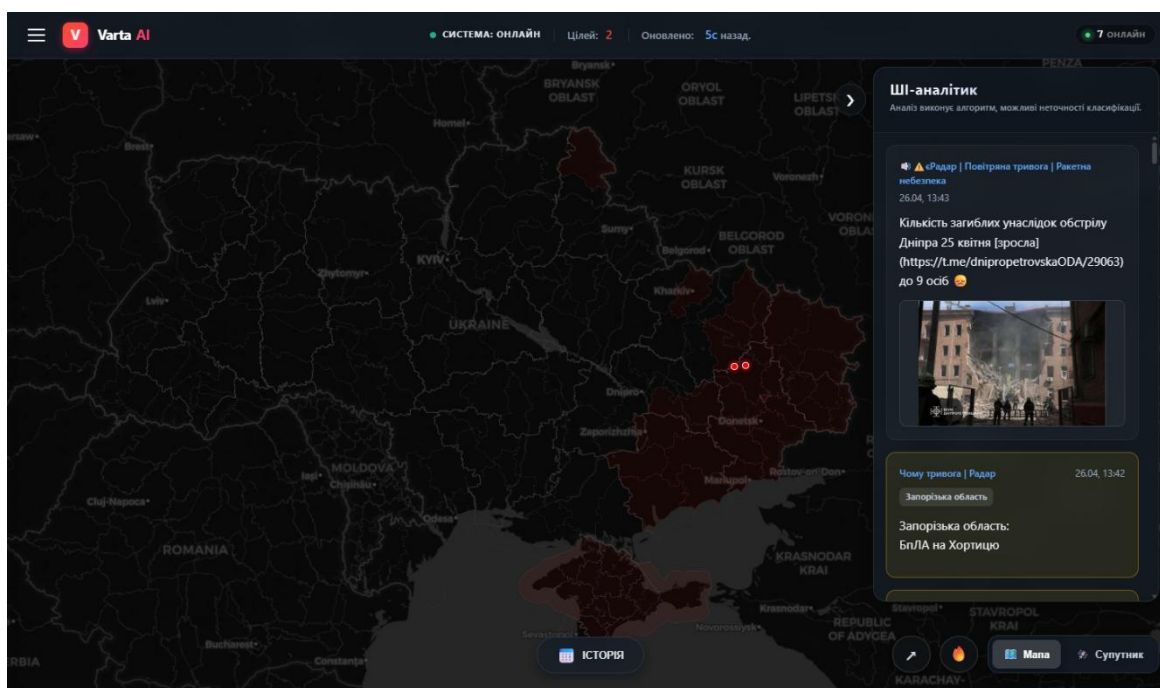


Рис. 1.3 Приклад інтерфейсу системи «VARTA AI»

Переваги використання системи «VARTA AI» для моніторингу інформації щодо повітряних цілей:

- наявність функції «Теплова мапа», що дозволяє візуалізувати частоту прольотів для подальшого статистичного аналізу;

- темна висококонтрастна підкладка карти, що забезпечує зручність роботи в нічний час;
- консолідація новин різних моніторингових каналів в єдиному інтерфейсі.

До недоліків системи «VARTA AI» можна віднести наступне:

- анонімність загроз: відсутність класифікації цілей та їхнього вектора руху робить систему сліпою щодо реального розвитку подій;
- масивна бічна панель («ШІ-аналітик») створює значний візуальний шум, змушуючи користувача знову повертатися до читання тексту;
- при натисканні на мітку користувач не отримує жодної інформації про час виявлення, висоту чи швидкість об'єкта;
- використання HTTP Polling без алгоритмів інтерполяції руху робить моніторинг дискретним та малоефективним під час активних атак.

Зведені результати аналізу характеристик розглянутих систем моніторингу інформації щодо повітряних цілей наведено у таблиці 1.1. Порівняння проводилося за критичними для цивільного населення параметрами, що дозволяє обґрунтувати доцільність розробки системи.

1.2 Критичні недоліки аналогів та обґрунтування розробки

Узагальнюючи аналіз існуючих систем, можна виділити три фундаментальні проблеми, які роблять розробку системи «КОНТУР» актуальною:

- висока залежність аналогів від штучного інтелекту, через що використання їхнього «сирого» виводу для парсингу координат неминуче супроводжується галюцинаціями нейромереж та призводить до появи фантомних міток у випадкових місцях;
- неможливість забезпечення якісного зв'язку в реальному часі через застосування застарілого методу циклічних опитувань, наслідком чого є виникнення затримок та розривів у візуалізованих траєкторіях руху цілей;

– систематичне порушення принципів когнітивного мінімалізму внаслідок перевантаження інтерфейсу рекламою, масивними панелями та супутниковими шарами, що критично заважає користувачу швидко ідентифікувати небезпеку.

Таким чином, виникає потреба у створенні системи, яка б поєднувала гібридний парсинг та протоколи миттєвої передачі даних. Зведені результати аналізу характеристик розглянутих систем моніторингу інформації щодо повітряних цілей наведено у таблиці 1.1. Порівняння проводилося за критичними для цивільного населення параметрами, що дозволяє обґрунтувати доцільність розробки системи

Таблиця 1.1

Зведені результати аналізу характеристик систем

Показник	«Neptun»	«Dimap»	«VARTA AI»
Метод парсингу даних	Пряме використання ІІІ	Поетапна обробка ІІІ	ІІІ-парсинг без фільтрів
Протокол синхронізації	HTTP Polling (циклічний)	HTTP Polling (циклічний)	HTTP Polling (циклічний)
Візуалізація руху	«Телепортація» міток	«Телепортація» міток	Статичне відображення
Зручність (UI/UX)	Середня	Низька	Середня
Класифікація цілей	Часткова	Повна	Відсутня
Платформа	Веб-сайт / Моб. дод.	Веб-сайт	Веб-сайт

1.3 Визначення вимог до системи «КОНТУР»

Детальний аналіз існуючих сервісів моніторингу повітряного простору виявив ряд критичних архітектурних та візуальних недоліків, що перешкоджають оперативному інформуванню населення. На основі проведеного порівняльного дослідження, а також враховуючи специфіку обробки неструктурованих даних у реальному часі, до розроблюваної системи «КОНТУР» висувається комплексний перелік вимог.

1.3.1 Функціональні вимоги

Для забезпечення ефективного функціонування системи як інструменту цивільного захисту, вона повинна реалізовувати наступний функціонал:

- безперервний моніторинг текстових джерел через Telegram Application Programming Interface (API) для збору та первинного аналізу інформації з миттєвим відсіюванням неінформативних повідомлень;
- використання гібридного інтелектуального парсингу з розпізнаванням параметрів цілей (тип, локація, вектор) за допомогою сигнатур токенів, що гарантує детермінованість результатів, із залученням ШІ-агентів лише для генерації нових правил обробки;
- автоматичне перетворення текстових назв населених пунктів у географічні координати з урахуванням особливостей української морфології та топонімічної неоднозначності;
- просторове розв'язання конфліктів на основі аналізу історії попередніх цілей та контексту активних повітряних тривог;
- забезпечення плавного переміщення маркерів на стороні клієнта за допомогою алгоритмів лінійної інтерполяції, для виключення ефекту телепортації;
- автоматичний розрахунок прогнозованого часу прибуття та видалення застарілих міток поза зонами активної небезпеки.

1.4.2 Нефункціональні вимоги

Технічні характеристики системи повинні відповідати високим стандартам надійності та швидкодії:

- трансляція оновлених даних користувачам через протокол SignalR повинна відбуватися протягом 0.1–3 секунд після опрацювання повідомлення сервером;
- стабільне функціонування серверної частини в умовах високого навантаження на ресурсообмеженому обладнанні (обсяг оперативного запам'ятовуючого пристрою (ОЗУ) до 1 ГБ та 2 ядра процесора);
- реалізація інтерфейсу за принципом когнітивного мінімалізму з кольоровою диференціацією загроз для миттєвого зчитування характеру небезпеки;
- наявність механізмів обмеження запитів (Rate Limiting) та захисту від DDoS-атак для збереження працездатності системи під час масових звернень користувачів;
- реалізація у форматі Telegram Mini App для забезпечення кросплатформеності та швидкого доступу без необхідності встановлення окремого ПЗ.

2 ОБҐРУНТУВАННЯ ТЕХНІЧНИХ ЗАСОБІВ ТА СТЕКУ РОЗРОБКИ

Вибір інструментарію для системи «КОНТУР» базувався на необхідності обробляти непередбачувані потоки даних у реальному часі. Оскільки моніторинг повітряних загроз не пробачає затримок, головним критерієм була швидкість реакції сервера та здатність клієнтської частини відображати динамічні об'єкти без втрати продуктивності.

2.1 Середовище розробки та мова програмування

2.1.1 Visual Studio 2022

Для написання та відладки основного коду обрано Visual Studio 2022. Це професійне середовище, яке має найбільш глибоку інтеграцію з платформою .NET. На відміну від легших редакторів, воно надає потужні інструменти для діагностики асинхронних потоків, що критично важливо при роботі з сокетом. Основні переваги використання Visual Studio 2022 у проекті:

- можливість у реальному часі бачити навантаження на CPU та пам'ять, що дозволило оптимізувати бекенд під ліміт у 1 Гб ОЗУ;
- має вбудований моніторинг активних підключень та повідомлень у SignalR хабах;
- інтеграція з Azure та GitHub, що забезпечує безшовне розгортання проекту на сервер та контроль версій без виходу з IDE.

2.1.2 C# (.NET 9)

Ядром системи стала мова C# на платформі .NET 9 [3]. Вибір зумовлений тим, що .NET 9 демонструє одні з найкращих показників швидкодії серед серверних рішень.

Ключові фактори вибору мови:

- асинхронність (Async/Await) дозволяє серверу обробляти сотні одночасних запитів від Telegram API та SignalR-клієнтів, не блокуючи ресурси системи;
- суворі типізація гарантує, що дані про координати цілей не будуть спотворені через помилки в типах даних під час складних математичних розрахунків;
- C# ефективно працює з рядками, це забезпечує високу швидкість обробки інформації, що важливо для первинної токенизації повідомлень перед їх парсингом.

2.2 Фреймворки та комунікаційна логіка

2.2.1 SignalR (Real-time зв'язок)

Для забезпечення миттєвого відображення цілей замість класичних HTTP-запитів використано бібліотеку SignalR [1]. Це «хребет» системи, який дозволяє серверу самостійно надсилати дані клієнтам, як тільки в Telegram з'являється нове повідомлення. Переваги SignalR для «КОНТУРУ»:

- клієнту не потрібно кожну секунду запитувати сервер про наявність нових цілей завдяки відмові від Polling-у, що економить трафік користувача та ресурси сервера;
- завдяки автоматичному вибору протоколу зв'язку система сама обирає WebSockets або альтернативи, забезпечуючи стабільний зв'язок навіть при слабкому мобільному інтернеті.

2.2.2 REST API та безпека (JWT)

Комунікація для адміністративних цілей (керування правилами, перегляд логів) реалізована через Representational State Transfer (REST API). Для захисту доступу обрано стандарт JSON Web Token (JWT) [11].

Чому це важливо:

- сервер не зберігає сесії користувачів, що робить систему легкою та стійкою до перезавантажень;
- кожен запит на зміну логіки парсингу перевіряється на наявність підписаного токена, що відсікає будь-які сторонні втручання.

2.3 Спеціалізовані компоненти та візуалізація

2.3.1 Hunspell та логіка GeoNormalizer

Для нормалізації географічних назв у тексті використано бібліотеку Hunspell [13]. Вона вирішує проблему морфології української мови (відмінювання назв міст). Роль компонента:

- приведення слів до називного відмінка, що дозволяє системі розуміти, що «Києва», «Києві» та «Київ» — це одна й та сама локація;
- Hunspell працює на порядки швидше, ніж будь-яка нейромережа, що забезпечує стабільну швидкість обробки повідомлення незалежно від навантаження на сторонні сервіси.

2.3.2 MapLibre GL JS та deck.gl

Візуальна частина системи побудована на зв'язці JavaScript бібліотек для малювання мапи та великої кількості цілей. Вибір припав на бібліотеки MapLibre GL JS [2] та deck.gl [8], адже вони були створені для високонавантажених інтерфейсів.

- MapLibre GL JS — відповідає за рендер векторної мапи України, бібліотека забезпечує плавний рух мапи, приближення та швидке завантаження мапи як на телефоні так і на комп'ютері;
- deck.gl — фреймворк, який працює на базі WebGL, що дозволяє використовувати потужність відеокарти для малювання сотень активних маркерів повітряних цілей.

2.4 Механізми автентифікації та безпеки доступу

Для захисту адміністративної панелі та критичних кінцевих точок API, у роботі реалізовано механізм авторизації на базі JWT [11]. Сервер самостійно валідує криптографічний підпис токена та визначає права користувача на зміну конфігурацій чи правил парсингу Графічне представлення прав доступу та функціональних можливостей для різних ролей користувачів у межах реалізованої моделі безпеки наведено на діаграмі прецедентів (див. рис. 2.1). Архітектура застосунку побудована за stateless-моделлю. Вона не вимагає виділення ресурсів для зберігання сесій на стороні бекенду.

Такий підхід суттєво економить оперативну пам'ять, що є пріоритетним завданням під час роботи на обмежених апаратних потужностях.

Інтеграція механізму JWT додатково регламентує середовище доступу. Запуск клієнтської частини системи дозволено виключно всередині месенджера Telegram [16]. Модуль авторизації перевіряє цілісність вхідних даних (наприклад, параметрів initData), підтверджуючи легітимність запиту від середовища Telegram Mini App. У разі успішної верифікації криптографічного хешу сервер генерує токен доступу. Це унеможливорює несанкціоноване відкриття картографічного інтерфейсу через сторонні браузері.

Додатковий контур інфраструктурної безпеки забезпечує сервіс Cloudflare [15]. Платформа функціонує як зворотний проксі-сервер (Reverse Proxy). Вона автоматично відхиляє аномальні запити та нейтралізує DDoS-атаки на мережевому рівні.

Зокрема, Cloudflare бере на себе кешування статичних ресурсів на власних крайових вузлах (CDN). Маршрутизація трафіку через розподілену мережу радикально знижує пряме навантаження на основний сервер та прискорює доставку контенту.

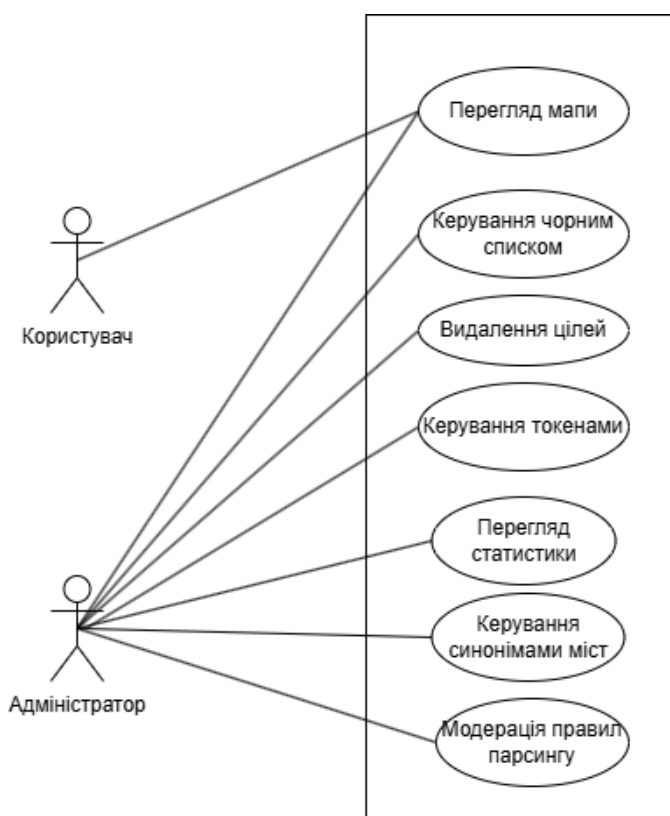


Рис. 2.1 Діаграма прецедентів

2.5 Контейнеризація та розгортання системи

З метою забезпечення апаратної незалежності застосовано технологію Docker [9]. Програмний код, залежності та конфігураційні файли повністю інкапсульовано в ізольований образ. Віртуалізація на рівні операційної системи нівелює ризики конфліктів середовища. Відповідно, алгоритм розгортання системи на нових серверах зведено до мінімуму операцій. Проєкт працює ідентично на будь-якому хостингу.

Серверну інфраструктуру розгорнуто на віртуальному виділеному сервері (VPS) хмарної платформи Microsoft Azure [10]. Вибір цього провайдера продиктований суворими вимогами до відмовостійкості. Адміністрування файлової системи та розміщення оновлень виконується через захищений протокол SFTP із використанням клієнта FileZilla [18]. Це створює зашифрований канал для безпечної передачі артефактів системи. Загальна архітектура розміщення

компонентів системи, що враховує контейнеризацію сервісів та багаторівневу мережеву інфраструктуру, представлена на діаграмі розгортання (див. рис. 2.2).

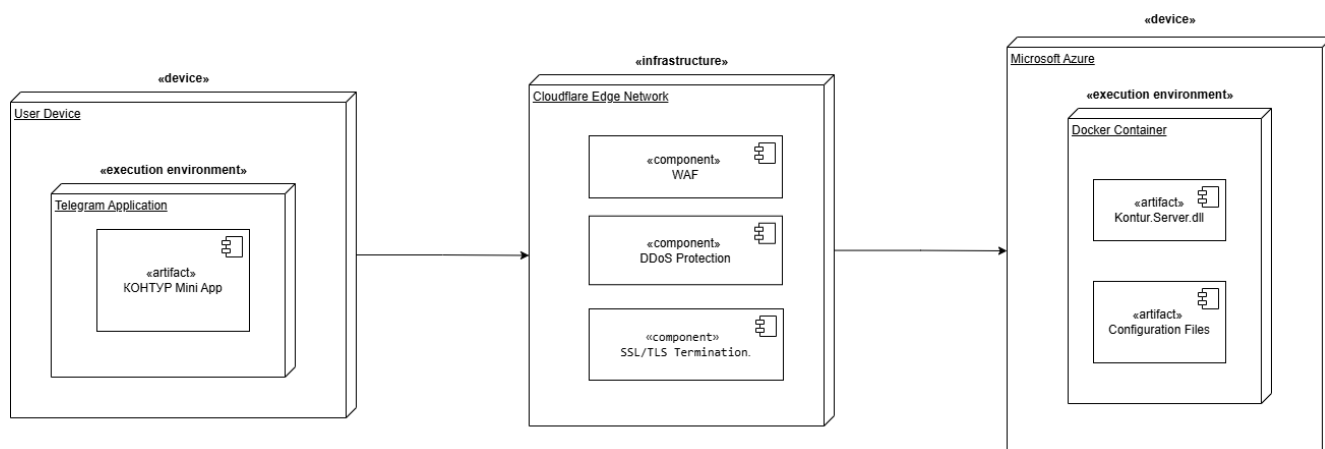


Рис. 2.2 Діаграма розгортання

2.6 Допоміжні засоби розробки та контролю версій

Процес підготовки географічних даних та формування бази координат населених пунктів України реалізовано за допомогою скриптів на мові Python у середовищі Thonny. Це легке IDE ідеально підходить для написання допоміжних інструментів, які парсять відкриті картографічні дані та перетворюють їх на готові структури формату JavaScript Object Notation (JSON) для сервісу *GeoNormalizer*.

Зберігання коду та контроль за кожною зміною в проєкті здійснюється через хмарний сервіс GitHub. Використання систем контролю версій дозволяє відстежувати історію розробки та бути впевненим у збереженні коду у разі збоїв локального обладнання.

2.7 Інструменти інтелектуальної обробки (AI)

Оскільки моніторингові канали постійно змінюють стиль подачі даних, система повинна динамічно адаптуватися до нових формулювань інформації. Для цього задіяно Google AI Studio з використанням API моделі Gemini 3.1 Flash Lite [14]. Важливо зауважити, що штучний інтелект не використовується для парсингу повідомлень, що було б занадто повільно та дорого. Натомість штучний інтелект залучається як «вчитель» лише тоді, коли система зустрічає невідому сигнатуру повідомлення. Модель допомагає згенерувати нове правило парсингу та валідувати його (за умови ввімкненого режиму «III Валідація»), після чого система додає це правило до бази даних і надалі обробляє аналогічні тексти миттєво без звернення до нейромережі.

2.8 Організація зберігання та обробки даних

Однією з ключових архітектурних особливостей системи «КОНТУР» є відмова від використання традиційних систем управління базами даних (СУБД), таких як PostgreSQL чи MongoDB. Це рішення було прийняте на основі аналізу технічних обмежень та специфіки роботи застосунку:

- розгортання окремого сервісу БД на сервері з обмеженою оперативною пам'яттю (1 ГБ) створило б значне додаткове навантаження на систему, тому відмова від СУБД дозволяє виділити максимум доступних ресурсів для роботи SignalR-хабів та логіки парсингу;
- читання даних безпосередньо з оперативної пам'яті відбувається на порядки швидше, ніж через мережеві запити або дискові операції стандартних БД;
- оскільки база координат населених пунктів та правила парсингу є відносно статичними та мають невеликий обсяг (декілька мегабайтів), їх зберігання у форматах JSON та CSV є оптимальним з точки зору зручності та швидкості зчитування.

3 ПРОЄКТУВАННЯ ТА АРХІТЕКТУРА СИСТЕМИ «КОНТУР»

Для проектування системи було обрано модульну архітектуру, завдяки чому при потребі можна без проблем замінити будь який модуль на інший. Загалом архітектура системи поділена на три умовні рівні:

- доменний рівень – рівень на якому оголошено всі сутності які використовуються в проєкті;
- рівень застосунку – основний двигун системи, саме тут реалізовано всю логіку обробки повідомлень;
- інфраструктурний рівень – рівень на якому дані з сервера передаються всім користувачам.

3.1 Доменний рівень

На цьому рівні відсутні будь які залежності від сторонніх бібліотек, адже на цьому рівні лише оголошуються всі об'єкти які використовуються в процесі обробки повідомлень та передачі даних від сервера клієнтам.

З огляду на значну кількість об'єктів у структурі проєкту, у даному розділі акцентовано увагу на ключових сутностях доменного рівня, які складають ядро інформаційної моделі системи та відповідають за збереження критично важливих даних. Одним із ключових об'єктів є *ThreatEvent*.

ThreatEvent - це базова доменна модель, що представляє структурований результат парсингу текстового повідомлення. На відміну від фінальної мітки на карті, цей об'єкт зберігає «сирі» семантичні дані, які ще не пройшли етап остаточного геопросторового розв'язання координат.

Об'єкт містить такі поля:

- тип об'єкту - БпЛА, КАБ, ракета чи балістична ракета;
- поточна позиція – географічна точка без застосування логіки визначення позиції, наприклад «Київ»;

- зміщення відносно поточної позиції – застосовується для уточнення зміщення цілі відносно поточної позиції для точнішого відображення цілі на мапі та коректнішого пошуку можливих попередників цілі, наприклад «схід»;
- цільова позиція – географічна точка в напрямку якої летить ціль, це може бути як місто, регіон так і напрямок типу «північ»;
- поточний регіон – вказує на регіон в якому знаходиться «поточна позиція» цілі, може визначатись як підчас парсингу повідомлення так і після нього;
- цільовий регіон – вказує на регіон в якому знаходиться «цільова позиція» цілі, поле визначається після парсингу;
- унікальний ідентифікатор – поле з типом Guid, призначене для визначення унікального ідентифікатора для кожної цілі;
- батьківський унікальний ідентифікатор – визначається якщо ціль є нащадком вже існуючої цілі, використовується для злиття з батьківським об'єктом, та збереження «ланцюга» з історії цілей;
- часова мітка – визначається після парсингу повідомлення, позначає час «коли була створена ціль», використовується для розрахунків поточної позиції, часу життя;
- посилання – створюється після парсингу, зберігає пряме посилання на повідомлення яке було оброблено парсером і створено об'єкт;
- кількість – поле визначається підчас парсингу повідомлення, зберігає орієнтовну кількість цілей. Поле не є обов'язковим, якщо підчас парсингу повідомлення в поле не було вказано кількість цілей, то кількість визначається як одна ціль.

Завдяки такій кількості полів система має достатньо даних для формування фінального об'єкту цілі, котрий і передається вже всім користувачам. Фінальний об'єкт цілі визначено як *ThreatEventDto*, цей об'єкт містить меншу кількість полів в зв'язку з тим що він призначений лише для передачі конкретних позицій цілі, а не абстрактних.

Об'єкт *ThreatEventDto* містить в собі наступні поля:

- унікальний ідентифікатор – поле з типом *Guid*, що слугує для ідентифікації цілі на стороні клієнта;
- тип об'єкту – вказує на категорію повітряної цілі (БпЛА, ракета, балістика тощо), це значення використовується для колірною кодування та вибору графічного символу мітки, що дозволяє користувачу миттєво зчитати характер небезпеки;
- часова мітка – позначає момент первинної появи цілі в системі. Використовується для аналізу тривалості перебування загрози в повітряному просторі та ведення історії подій;
- час останнього оновлення – фіксує точний час останньої зміни даних про об'єкт, це поле є критично важливим для алгоритмів клієнтської інтерполяції, що забезпечують плавний рух мітки між оновленнями координат;
- поточний час сервера – передає актуальний час на сервері в момент трансляції об'єкта клієнту;
- кількість – відображає чисельність цілей у групі, визначену під час парсингу;
- опорна геопозиція – об'єкт типу *GeoEntry*, що містить валідовані координати (широту та довготу) «якоря», де безпосередньо відображається маркер на інтерактивній мапі;
- цільова геопозиція – об'єкт типу *GeoEntry*, що визначає координати точки, в напрямку якої рухається ціль;
- швидкість – статична величина котра визначається на основі типу цілі під час формування фінального об'єкту цілі;
- статус загрози – вказує на поточний стан об'єкта (активний, статичний або неактивний).

Зауважу що поля «Опорна геопозиція» та «Цільова геопозиція» завжди мають дані в собі, адже на етапі визначення позиції цілі використовується значна

кількість алгоритмів і перевірок, завдяки чому ні одна ціль не залишається без цих даних.

Наступним важливим об'єктом є *GeoEntry*, об'єкт котрий зберігає в собі дані населених пунктів, регіонів та важливих місць. Завдяки цьому об'єкту система має швидкий доступ до детальних даних про кожен населений пункт. Сам об'єкт містить в собі такі поля:

- канонічна назва – основне стандартизоване найменування географічного об'єкта (наприклад, «Київ», «Одеса»), це поле використовується як ключ для ідентифікації локації та відображення назви на інтерфейсі користувача;
- категорія об'єкта – визначає тип географічної сутності: місто, область, район або інше місце (море, об'єкт критичної інфраструктури, пусковий майданчик тощо);
- географічні координати – широта та довгота центру населеного пункту або регіону;
- масив лінгвістичних стемів – набір основ слів, що використовуються алгоритмом для ідентифікації населеного пункту в неструктурованому тексті;
- регіональна приналежність – вказує на адміністративну область, до якої належить даний населений пункт;
- чисельність населення – статистичний показник, що використовується як ваговий коефіцієнт під час просторового аналізу.

Візуалізація зв'язків цих трьох основних моделей даних продемонстровано на рисунку 3.1.

3.2 Рівень застосунку

На рівні застосунку реалізовано основну бізнес-логіку системи «КОНТУР». Він спроектований як багатостадійний конвеєр, де кожне повідомлення проходить через низку фільтрів та аналізаторів (див. рис. 3.2). Саме на цьому рівні система перетворює неструктуроване текстове повідомлення на об'єкт даних, котрий згодом відображається у всіх користувачів.

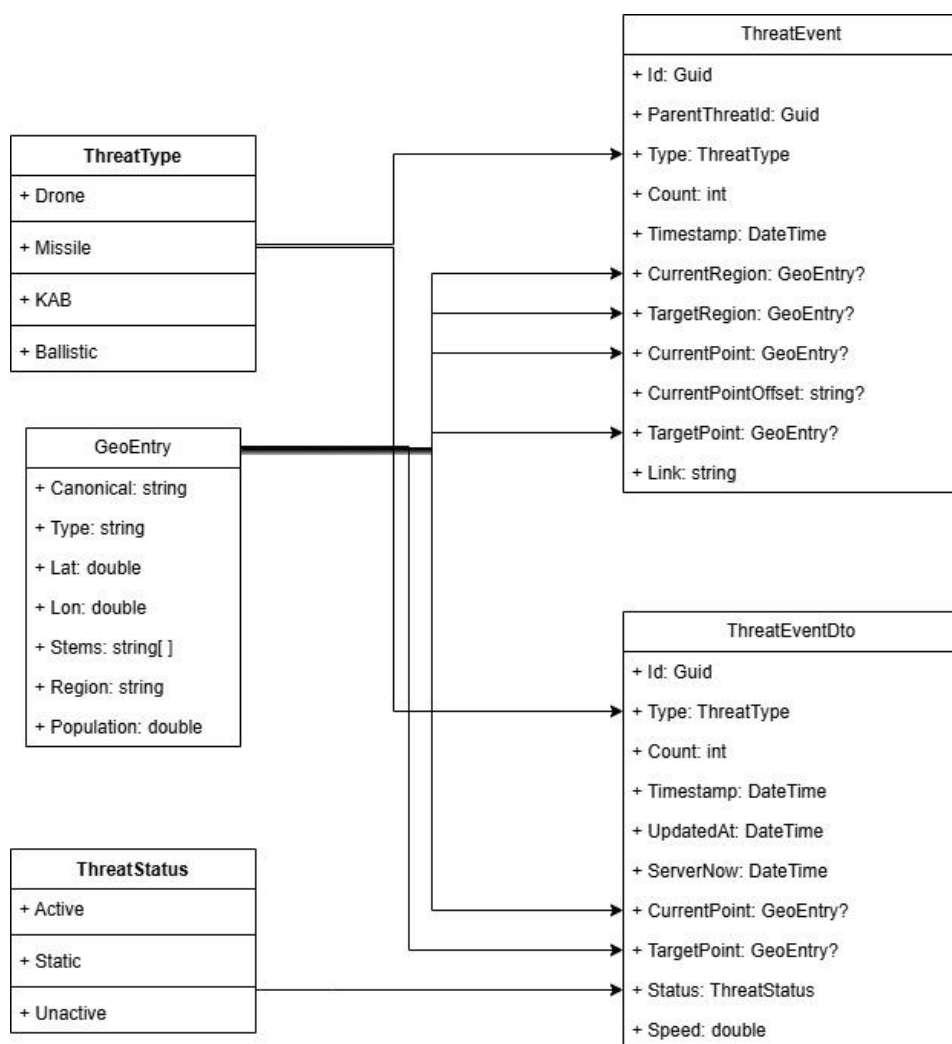


Рис. 3.1 Діаграма зв'язків основних моделей даних

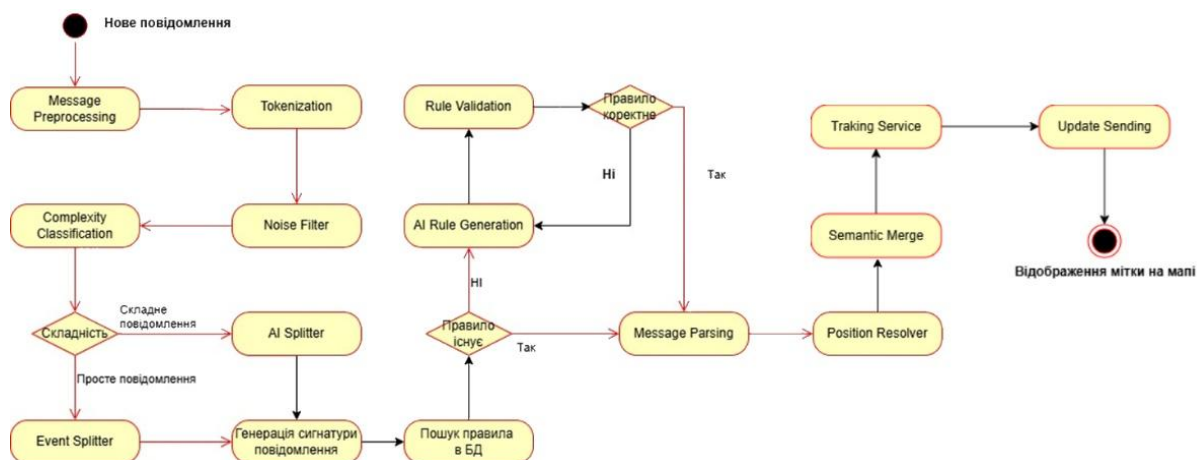


Рис. 3.2 Діаграма послідовності обробки повідомлення

Повідомлення на рівні застосунку проходить довгий шлях обробки:

- message preprocessing — попередня обробка сирого тексту повідомлення;
- tokenization — розбиття тексту на окремі семантичні одиниці (токени);
- noise filter — відсіювання неінформативного «шуму» та повідомлень, що не містять даних про загрози;
- complexity classification — визначення рівня складності структури повідомлення;
- розділення подій (AI Splitter / Event Splitter) — декомпозиція складних повідомлень на окремі одиниці за допомогою ШІ або зумовлених алгоритмів;
- генерація сигнатури повідомлення — створення унікального ідентифікатора структури повідомлення на основі токенів;
- пошук правила в БД — перевірка наявності готового шаблону обробки для отриманої сигнатури;
- ai rule generation та rule validation — цикл самонавчання: генерація та перевірка нового правила у разі відсутності аналогів у базі;
- message parsing — остаточне розбирання повідомлення на структурований об'єкт даних про ціль;
- position resolver — визначення географічних координат цілі на основі контексту тривоги та історії об'єктів;
- семантичне об'єднання — поєднання цілей з однаковими даними але їх різною повнотою та конкретикою;
- tracking service — перевірка об'єкта на приналежність до існуючих цілей (ідентифікація оновлення треку);
- update sending — трансляція оновлених даних через протокол реального часу;
- відображення мітки на карті — фінальна візуалізація та інтерполяція руху цілі на стороні користувача.

У роботі застосовано багаторівневу декомпозицію логіки обробки вхідного потоку. Відповідно, система «КОНТУР» функціонує за гібридною моделлю аналізу даних. Безпосередній синтаксичний розбір неструктурованих повідомлень у реальному часі виконується детермінованими алгоритмами без прямого залучення штучного інтелекту. Це гарантує стабільність вихідних структур.

ШІ-модуль інтегровано в загальний конвеєр виключно як механізм самонавчання системи. Сервер ініціює запит до нейромережі лише для синтезу та перевірки нових правил парсингу на коректність, коли фіксує появу невідомих раніше сигнатур токенизованого повідомлення. Такий розподіл обов'язків ізолює критичний шлях обробки даних від непрогнозованих мережових затримок.

На відміну від існуючих архітектурних рішень, де LLM-моделі обробляють кожен вхідний запит, запропонований підхід оптимізує роботу системи за трьома векторами:

- швидкодія обчислень, оскільки модуль зіставляє вхідні дані з базою правил безпосередньо в оперативній пам'яті, зводячи затримку трансляції до мілісекунд;
- апаратна та фінансова ресурсоефективність, яка досягається шляхом делегування штучному інтелекту виключно функції оновлення правил, що радикально зменшує кількість платних викликів зовнішніх API та пропорційно знижує навантаження на процесор і використання оперативної пам'яті сервера;
- критична надійність, завдяки якій жорстка алгоритмізація розбору потоку повністю унеможливорює виникнення генеративних «галюцинацій», дозволяючи картографічному модулю відображати лише математично перевірену інформацію.

3.3 Алгоритмічне забезпечення конвеєра обробки повідомлень

На основі розробленої архітектурної схеми, процес трансформації неструктурованого тексту у візуальний об'єкт поділено на наступні етапи:

3.3.1 Первинна обробка повідомлень

На цьому етапі система виконує попередні дії з повідомленням, однією з найпопулярніших дій з повідомленням є видалення завчасно визначених слів. При отриманні повідомлення система бере унікальний ідентифікатор Telegram-каналу, та звіряє його з завчасно визначеними ідентифікаторами каналів для первинної обробки повідомлень, якщо є відповідний блок дій для попередньої обробки з таким ідентифікатором що співпадає з ідентифікатором каналу отриманого повідомлення, система бере отримане повідомлення та виконує дії вказані в файлі налаштувань первинної обробки.

Первинна обробка передбачає такі дії [6]:

- видалення визначених слів/рядків;
- заміна унікальних емої символів на конкретний текст (наприклад замість емої БпЛА система підставить текст «БпЛА»);
- додавання типу цілі до повідомлення без вказаного явного типу цілі.

Потреба у первинній обробці виникла на тлі того, що значна кількість моніторингових Telegram-каналів використовують унікальні емої замість тексту для позначення типу цілі, через що система фільтрації інформаційного шуму не пропускає повідомлення в обробку, адже вважає його не інформативним. Також деякі Telegram-канали не вказують тип цілі якщо це БпЛА, адже переважна більшість повітряних цілей – це БпЛА. Завдяки цьому етапу, система не відкидає інформативні повідомлення з завчасно визначених Telegram-каналів і успішно парсить дані з нього.

Також етап попередньої обробки включає в себе видалення завчасно визначених слів або рядків. Цей етап застосовується до декількох Telegram-каналів. Використання цієї обробки потрібен в зв'язку з тим, що Telegram-канали до повідомлень додають закінчення з посиланням або назвою власного каналу, назва якого може містити назви населених пунктів (наприклад «Конотопський Вартовий»), через що система хибно вважає, що це або поточна позиція, або цільова позиція об'єкта.

3.3.2 Семантична токенизація тексту

Наступним етапом після первинної обробки повідомлення є етап семантичної токенизації [4]. На цьому етапі застосовується семантичний токенизатор, який виконує роль інтелектуального лексичного аналізатора. На відміну від звичайних токенизаторів, які просто ділять текст на слова, цей компонент:

- розпізнає складні багатослівні фрази;
- використовує Fuzzy Search (нечіткий пошук) для виправлення помилок у написанні;
- зберігає контекст для специфічних типів даних (наприклад, назви загроз або кількості);
- виконує семантичну перевірку логічних зв'язків.

3.3.2.1 Алгоритм роботи семантичного токенизатора

Процес токенизації розділений на три критичні фази, що дозволяє уникнути конфліктів між короткими словами та довгими фразами.

Програма спочатку шукає токени, що складаються з декількох слів (наприклад, «Південний захід»). Для кожної фрази створюється регулярний вираз. Якщо фраза складається лише з букв, додається перевірка меж слова ((?<!\w)), щоб уникнути помилкового спрацювання всередині інших слів. Якщо токен позначений як "Value Token" (наприклад, THREAT), система зберігає оригінальне значення: [THREAT=крилата ракета]. Це важливо для збереження конкретики при подальшому аналізі. Після обробки фраз система аналізує окремі слова, що залишилися. Якщо слово вже є частиною знайденої раніше фрази, воно ігнорується (IsOverlapping).

Алгоритм Левенштейна, це центральна частина стійкості системи. Якщо слово написане з помилкою (наприклад, "раккета"), метод FindTokenForWord обчислює відстань редагування [12]:

- для слів довше 8 символів дозволено 2 помилки;
- для слів довше 5 символів - 1 помилка;

– якщо слово не знайдено в основному словнику, система перевіряє додаткові конфігурації ключових слів.

Останнім етапом токенизації є побудова фінального рядка та вирішення конфліктів. Це етап збірки результату без руйнування структури тексту (пробілів, розділових знаків).

– токени сортуються за індексом, а при рівних індексах - за довжиною (спочатку довші). це гарантує, що "Південний Захід" не буде розбитий на два окремих токени "Південний" і "Захід";

– система видаляє токени, які фізично накладаються один на одного;

– ефективна збірка тексту, де замість знайдених слів вставляються токени у форматі [TOKEN_TYPE].

3.3.2.2 Семантична валідація

Цей метод реалізує базову контекстну граматику. Він перевіряє логічний оператор [AND].

Логіка: Оператор [AND] залишається в тексті лише тоді, коли він стоїть між двома містами (токен CITY).

Приклад:

– "Київ [AND] Львів" - залишається як є;

– "Ракета [AND] Львів" - [AND] буде видалено, оскільки семантично це не перелік локацій.

3.3.3 Фільтрація інформаційного шуму

Для забезпечення високої точності аналізу повідомлень у реальному часі було розроблено багаторівневу систему фільтрації. Основне завдання компонента, це ідентифікація та відсіювання повідомлень, які не несуть операційної цінності (так званого «інформаційного сміття» або «шуму»), що дозволяє знизити навантаження на обчислювальні ресурси та мінімізувати кількість хибно-позитивних результатів.

3.3.3.1 Архітектура процесу оцінки складності повідомлення

Алгоритм використовує комбінований підхід, що включає статистичний аналіз, лексичну перевірку та семантичну верифікацію. Процес оцінки повідомлення розділений на кілька етапів:

- лексична фільтрація на основі сигнатур;
- оцінка структурної цілісності та вертикального форматування.

На початковому етапі система проводить сигнатурний аналіз тексту за допомогою попередньо визначеного словника стоп-фраз (`_stopPhrases`). До цього списку включені:

- маркери завершення загрози («відбій загрози», «простір чистий»);
- адміністративно-побутові повідомлення («посилання на банку», «донат», «спокійної ночі»);
- ретроспективні звіти та новини («у ніч на», «станом на»).

Для запобігання помилковим спрацьовуванням усередині довгих слів використовується механізм регулярних виразів із перевіркою меж слова (`\b`), що забезпечує пошук лише цілих лексем.

Етап оцінки структурної цілісності спрямований на розпізнавання дописів аналітичного або публіцистичного характеру, які, попри наявність ключових слів, не мають оперативної цінності через свою надмірну розлогість. Метод базується на аналізі архітектури тексту — співвідношенні загального обсягу символів до кількості логічних розривів (переносів рядків).

Принцип розрахунку: Система визначає ступінь сегментації тексту, обчислюючи, скільки в середньому символів припадає на один рядок повідомлення. Це дозволяє виявити «монолітні» блоки тексту, де відсутнє характерне для оперативних зведень розбиття на короткі тези чи пункти.

Критерії класифікації: Повідомлення маркується як нерелевантне («суцільна стаття»), якщо воно відповідає двом одночасним умовам: має значний загальний обсяг (понад двісті знаків) та демонструє аномально високу середню довжину рядка (більше ста знаків на один сегмент).

Вибір таких параметрів зумовлений специфікою моніторингу загроз: критично важлива інформація зазвичай подається у вигляді лаконічних, структурованих списків. Натомість відсутність регулярних переносів рядків у великому текстовому масиві свідчить про описовий характер повідомлення, що є ознакою інформаційного шуму для систем реального часу.

3.3.3.2 Семантична верифікація та «сигнальна щільність»

Після проходження базових фільтрів повідомлення перевіряється наявність критично важливих семантичних одиниць (токенів). Повідомлення вважається інформативним лише за наявності хоча б одного токена типу THREAT (тип загрози), ACTION_START (початок дії), COUNT (кількість) або DIRECTION (вектор руху). Розраховується коефіцієнт відношення кількості виявлених токенів до загальної кількості значущих слів у тексті. При значенні менше 0.4 (\$40\%\$) повідомлення вважається «низькощільним» і відхиляється як занадто розмите або надлишкове.

3.3.3.3 Детекція аномалій та контекстний аналіз

Система реалізує захист від специфічних помилок токенизації та масового зведення даних:

- географічна аномалія: виявлення послідовностей із трьох і більше географічних назв (CITY) поспіль без розділових знаків або логічних зв'язок вважається ознакою помилкового розбору або неструктурованого переліку, що призводить до дискваліфікації повідомлення;
- детекція пакетних зведень (Batch Messages): окремий алгоритм визначає, чи є повідомлення багаторядковим зведенням, і якщо текст містить понад три рядки з високою концентрацією токенів загроз та локацій, він позначається як Batch, що дозволяє застосувати до нього іншу логіку парсингу;

– щільність геоданих: якщо частка географічних токенів у повідомленні перевищує 35% від загальної кількості слів, повідомлення ідентифікується як статистичне зведення або новина, а не оперативна інформація про рух загрози.

3.3.4 Класифікація складності структури (Complexity Classification)

Система класифікації складності реалізує багаторівневу детерміновану логіку для віднесення повідомлення до одного з трьох основних класів складності: Simple (Просте), MultiVector (Мультивекторне) або Batch (Групове/Список). Класифікація базується на аналізі взаємозв'язків між виявленими токенами загроз, напрямків та географічних об'єктів.

3.3.4.1 Етап ідентифікації структури типу «Batch» (Списки)

На першому етапі алгоритм перевіряє повідомлення на приналежність до класу Batch. Це критично важливо, оскільки такі повідомлення містять переліки незалежних подій, що вимагають попереднього розділення (сплітингу). Метод IsBatchListMessage аналізує текст порядково за наступними критеріями:

- кожен рядок класифікується як «заголовок» (містить назву загрози або двокрапку), «повноцінна подія» (загроза + локація) або «елемент списку» (лише локація/напрямок);
- повідомлення класифікується як Batch, якщо виявлено принаймні дві самостійні події в різних рядках, або якщо заголовок із назвою загрози супроводжується переліком географічних об'єктів.

3.3.4.2 Аналіз просторової топології (Multi-Vector Detection)

Якщо повідомлення не є списком, алгоритм переходить до оцінки його внутрішньої семантичної складності. Ключовим завданням тут є виявлення незалежних кінцевих цілей руху загрози.

Для уникнення помилкового трактування транзитних пунктів як фінальних цілей, алгоритм застосовує контекстні фільтри до кожного токена типу CITY або REGION:

- локація ігнорується як незалежна ціль, якщо вона поєднана з попередньою через логічні сполучники;
- використовуючи токени прийменників, алгоритм виключає локації, які є лише точками старту або проміжними етапами маршруту;
- якщо за локацією слідує маркер напрямку, вона вважається проміжним орієнтиром, а не фінальною метою.

Повідомлення отримує складність MultiVector, якщо виконується хоча б одна з умов:

- виявлено більше однієї незалежної фінальної локації (цілі) в одному реченні;
- наявність декількох векторів напрямку, які не з'єднані логічними сполучниками, що вказує на складну траєкторію або розділення об'єктів;
- присутність декількох маркерів активного маневрування в одному контексті.

3.3.4.3 Складність «Simple»

До складності Simple відносяться повідомлення, які пройшли попередні етапи перевірки та містять лінійну логіку: одна загроза, що рухається за одним вектором або до однієї конкретної цілі (або групи пов'язаних цілей).

Така модель класифікації дозволяє системі «Контур» адаптивно обирати метод парсингу: прості події обробляються лінійно, мультивекторні проходять через глибший семантичний розбір для побудови графа руху, а групові повідомлення попередньо сегментуються на атомарні складники. Це мінімізує ризик втрати інформації при складних сценаріях моніторингу інформації щодо повітряних цілей.

3.3.5 Декомпозиція складних подій (AI Splitter та Event Splitter)

Після отримання типу складності повідомлення система має два шляхи обробки повідомлення:

- поділ на окремі події за допомогою коду;
- поділ на окремі події за допомогою ІІІ.

Поділ за допомогою коду, використовується якщо повідомлення має типи «просте» або «список», а для поділу на окремі події за допомогою штучного інтелекту, повідомлення повинно мати тип складності «багатовекторне».

Модуль функціонує як машина станів, що оперує трьома основними категоріями контексту:

- ідентифікатор об'єкта (наприклад, [THREAT=DRONE]);
- неографічні координати або назви регіонів, що стосуються наступних подій;
- текстовий префікс: оригінальний неструктурований текст для збереження читабельності фінального повідомлення.

Алгоритм аналізує кожен рядок повідомлення, класифікуючи його за одним із наступних паттернів:

- явні заголовки визначаються за наявністю маркера [COLON] (двокрапка), текст до роздільника екстрагується як сильний контекст, що буде застосований до всіх наступних рядків у межах цього блоку;
- зведені заголовки загроз, це рядки що містять лише тип загрози без деталізації цілей, а наступний рядок містить географічну прив'язку, поточний рядок стає контекстним донором;
- неявні локаційні заголовки, це окремі назви регіонів (наприклад, «Сумщина»), які формально не є подіями, але визначають територіальну приналежність наступних інформаційних пунктів.

Специфічною особливістю модуля є здатність ідентифікувати продовження маршруту. Якщо рядок починається з маркерів руху ([DIRECTION_MARKER], [TRANSIT_PREP], [AND]) і не містить нових сутностей чи кількості, він не створює нову подію, а агрегується з попередньою. Це дозволяє коректно обробляти конструкції типу:

- «крилаті ракети на Київ» (Подія 1);

- «...через Бровари» (Продовження Події 1, а не нова подія).

На фінальній стадії формування події відбувається «насичення» рядка відсутньою інформацією:

- якщо в рядку відсутній токен [THREAT], система автоматично додає його з поточного контексту;
- якщо рядок містить деталі руху, але не містить прив'язки до регіону/міста, контекстні локаційні токени вставляються на початок або після токенів кількості;
- для реконструкції тексту до вихідного неструктурованого рядка додається префікс із контексту для забезпечення лінгвістичної повноти (наприклад, додавання назви області до назви міста).

Другий варіантом поділу текстового повідомлення на окремі незалежні події зумовлений тим, що передбачити всі формулювання подачі інформації в Telegram-каналах – неможливо. В зв'язку з цим реалізовано систему котра у випадку потрапляння в обробку повідомлення котре має складну структуру і послідовність токенів передає повідомлення штучному інтелекту з запитом на поділ його на окремі атомарні події. Для кращого розуміння що саме потрібно зробити, створено системний промпт, котрий описує задачу та наводить декілька реальних прикладів того яким може бути вхідне повідомлення і яким повинна бути його відповідь.

Завдяки такому підходу до поділу подій система завжди отримує інформацію з текстових повідомлень незалежно від того яку структуру має повідомлення.

3.3.6 Геопросторовий аналіз та нормалізація

Одним з важливих етапів системи є розпізнавання географічних об'єктів у тексті повідомлення та їх точна прив'язка до реальних координат. Цей процес починається з модуля розпізнавання міст, який циклічно проходить по тексту токенозованого повідомлення. За допомогою регулярних виразів система розбиває текст на окремі елементи, автоматично ігноруючи системні токени та слова з чорного списку. Для того, щоб коректно обробляти складені географічні назви,

алгоритм сканує сусідні токени (слова) вперед, намагаючись об'єднати їх у єдину фразу, а також перевіряє наявність уточнюючого регіонального контексту, наприклад, назви області в дужках. Знайдена фраза групується у так званий пул кандидатів, який далі передається на етап нормалізації.

Ключовим етапом обробки геоданих є нормалізація знайдених фраз. Оскільки в українській мові географічні назви часто вживаються у різних відмінках, система застосовує лематизацію на базі словників Hunspell та генерує власні морфологічні форми, щоб привести вхідне слово до базового називного відмінка [13]. Знайдені у базі даних локації-кандидати порівнюються з оригінальним текстом за допомогою алгоритму Левенштейна для обчислення відсотка лінгвістичної подібності [12]. Після цього кожен кандидат проходить сувору контекстну фільтрацію. Система перевіряє поточну ситуацію: чи оголошена повітряна тривога у відповідній області або сусідніх районах, чи належить ця область до переліку постійно небезпечних зон. Якщо кандидат виявляється населеним пунктом, що знаходиться глибоко в тилу без активної загрози, системні алгоритми накладають на нього суттєвий математичний штраф, тим самим відсікаючи хибні спрацювання.

Найскладнішою функцією просторового аналізу є розв'язання географічних колізій, які виникають, коли базі відомо декілька населених пунктів з однаковою назвою. Для вирішення цього завдання система використовує модуль просторового розв'язання. Алгоритм намагається виділити «просторовий якір» — велике місто або пріоритетну область у повідомленні, у яких система впевнена найбільше. Якщо такий якір знайдено, система оцінює всі інші спірні локації за їх відстанню до цього центру: локації, що розташовані ближче до якоря, отримують значний просторовий бонус і виграють у рейтингу.

Додатково на вагу кандидата впливають демографічний бонус (залежить від кількості населення) та тактичний бонус (близькість до кордону або наявність нещодавніх загроз у цьому секторі).

У випадку, якщо система не може виділити єдиний просторовий якір, вона застосовує алгоритм взаємної кластеризації. Модуль виконує перехресну перевірку

координат між усіма спірними варіантами з різних пулів, шукаючи таку комбінацію населених пунктів, які фізично розташовані найближче один до одного.

Після того як просторові конфлікти вирішено і найкращих кандидатів затверджено, система здійснює фінальну заміну. Оригінальні слова в тексті замінюються на стандартизовані системні токени відповідного типу, наприклад [CITY=Назва], [REGION=Назва] або [CARDINAL_DIRECTION=Напрямок], після чого повністю розмічене повідомлення повертається для подальшої обробки парсером.

3.3.7 Генерація сигнатури повідомлення та пошук шаблонів

Одним з важливих етапів системи «Контур» є генерація сигнатури токенованого повідомлення. Завдяки цьому етапу система має короткий зліпок сигнатури повідомлення, і саме по цьому «зліпку» система й виконує пошук відповідного правила для парсингу повідомлення.

Для генерації сигнатури повідомлення, система використовує завчасно визначену «мапу символів токенів», при надходженні нового повідомлення в метод генерації сигнатури, повідомлення спочатку розбивається на окремі токени, таким чином система матиме окремий доступ до кожного токenu, що значно полегшує роботу генератора сигнатури. Наступним етапом після поділу токенованого повідомлення на окремі токени є циклічний перебір всіх токенів і формування рядку сигнатури. Система в циклі проходиться по списку окремих токенів, і для кожного з них за допомогою *Regex* виразів знаходить відповідний символ. Після виконання циклу система має сформований рядок сигнатури повідомлення.

Після формування сигнатури система використовуючи її виконує пошук відповідного правила парсингу серед наявних. Якщо в системі вже є правило парсингу з відповідною сигнатурою, система поверне його для подальшого використання парсером. А у випадку відсутності правила парсингу з відповідною сигнатурою система звернеться до штучного інтелекту для генерації його.

3.3.8 Інтелектуальний синтез та валідація правил

Ключовим етапом самонавчання системи є використання штучного інтелекту як генератора правил для парсингу [7]. Якщо у системі відсутнє правило для нової сигнатури, система звертається до штучного інтелекту для генерації цього правила. Для правильного розуміння штучним інтелектом задачі котра поставлена, створено системний промпт, котрий включає в себе детальний опис вимог, заборон та настанов для генерації нового правила, також в системному промпті вказано декілька прикладів, завдяки чому штучний інтелект краще розуміє, що саме йому потрібно повернути і яким чином опрацювати вхідне повідомлення.

Після генерації створене правило проходить первинну перевірку кодом. Викликається метод перевірки правила, котрий в свою чергу, виконує перевірки:

- наявність токenu в полі Type;
- присутність лише одного токenu в полі Type, а якщо буде декілька токenu в полі, то система відхилить правило;
- наявність хоча б однієї географічної точки (поточна позиція, або цільова), якщо точка відсутня - правило відхиляється.

Завдяки перевірці правила кодом, система автоматично відсікає не правильні або не повноцінні правила парсингу, завдяки чому до наступного етапу перевірки правил доходять лише не правильно оформлені правила.

Якщо правило пройшло перевірку кодом, і система не відхилила його, наступним кроком стає його затвердження. Сам процес затвердження поділяється на три типи:

- ручне затвердження;
- затвердження за допомогою штучного інтелекту;
- автоматичне затвердження без перевірок.

Варіант затвердження змінюється через відповідний метод, виклик якого доступний адміністратору в окремому веб-додатку.

У випадку якщо обрано ручний режим затвердження, система надсилає в адміністраторський веб інтерфейс запит на перевірку правила, де адміністратор

бачить сире повідомлення, токенозоване повідомлення та правило парсингу. Проаналізувавши сире повідомлення адміністратор перевіряє правило парсингу, якщо всі поля розмічено вірно – адміністратор затверджує правило і воно відправляється на збереження в систему, після чого воно записується в json файл і система перезапускає етап «пошук правила за сигнатурою», а завдяки тому що ми тільки що зберегли нове правило з відповідною сигнатурою система успішно його знаходить та надсилає правило і повідомлення в паттерн парсер.

Якщо обрано режим затвердження правил через штучний інтелект, то система надсилає сире повідомлення та правило парсингу штучному інтелекту. Додатково надсилається системний промпт, котрий детально описує роль штучного інтелекта, задає йому інструкції та приклади які правила є вірні а які ні. Основна ідея штучного інтелекту як валідатор, це «чи вірно передано суть сирого повідомлення в правилі парсингу?», ми змушуємо штучний інтелект зрозуміти суть повідомлення, а не порядок слів, і чи вірно передається ця суть повідомлення, адже в українській мові до прикладу «на» може означати як і курс «куди летить» так і «де зараз» ціль. Після перевірки штучний інтелект повертає об'єкт котрий в собі містить такі поля:

- булевий прапорець «правильне чи неправильне» правило;
- опис рішення або причини відхилення правила.

Якщо у повернутому об'єкті булевий прапорець дорівнює false, це означає що правило не є вірним, в такому випадку система надсилає новий запит штучному інтелекту для генерації правила, але до попереднього системного промпту додається новий контекстний блок, у якому вказано всі попередні правила які він згенерував в контексті цього повідомлення, та уточнення що «всі ці правила не є правильними, їх не повторювати». Завдяки такому циклу система має змогу самостійно навчатись, генерувати та перевіряти нові правила парсингу.

У випадку якщо штучні інтелекти не змогли узгодити правило за умовно 3 ітерації циклу, останнє створене правило надсилається на ручну перевірку адміністратору, час життя такої ручної перевірки становить 4 хвилини, якщо за цей

час адміністратор не виконає перевірку правила, і на сервер не надійде будь якої дії - правило автоматично вважається не вірним.

У випадку автоматичного прийняття правил не застосовуються ніякі перевірки окрім перевірки кодом. Якщо система визначила, що правило містить один токен типу цілі в правилі, та хоча б одну географічну точку (поточна позиція або цільова), правило автоматично зберігається та система продовжує роботу з цим правилом. Цей варіант затвердження створено на майбутнє, коли системні промпти будуть налаштовані максимально ефективно, завдяки чому перевірка їх вручну або через штучний інтелект не потрібна буде.

3.3.9 Синтаксичний розбір та формування об'єкта загрози

Завершальною ланкою в обробці повідомлення з Telegram-каналу є використання модуля паттерн парсера для об'єктної десеріалізації повідомлень. Основною його задачею є розібрати вхідне токенозоване повідомлення, і створити з нього об'єкт цілі *ThreatEvent*, для цього модулю передається правило для парсингу, та саме токенозоване повідомлення, після чого виконується ряд функцій для розбору повідомлення на об'єкт даних.

Однією з найскладніших функцій парсера є розв'язання географічних колізій та обробка складних локацій. Також система виконує групування індексів, якщо між двома токенами [CITY=value] стоїть токен [AND], система відмічає, що це не одна ціль а декілька, адже у нас йде перелік міст. Відповідно якщо у нас в полі поточної позиції стоїть перелік з двох і більше міст, система створює стільки об'єктів, скільки міст перелічено через токен [AND].

Якщо в полі цільової позиції, або поточної позиції визначено декілька міст/областей і між ними стоїть токен [SLASH] система бере всі географічні точки котрі перелічені, та вираховує середню точку між ними, при цьому створюється лише один об'єкт. Така логіка зумовлена тим, що в більшості випадків коли адміністратори моніторингових Telegram-каналів не знають куди точно летить ціль, на «місто А» чи на «місто Б» вони пишуть назви обох міст через скісну риску.

Окрім обробки географічних даних, модуль виконує фінальне визначення типу цілі та кількість.

3.3.10 Семантичне поєднання цілей

Після отримання сирого об'єкта цілі, система повинна перевірити на те чи нова ціль не є вже існуючою але з іншим рівнем конкретики даних.

До прикладу, перший Telegram-канал опублікував повідомлення «БпЛА з Сумщини на Чернігівщину», рівень конкретики даних тут майже максимальний, адже в повідомленні не вказано жодного конкретного населеного пункту, а лише області які займають сотні квадратних кілометрів площі, і визначити конкретно де ціль – не можливо. А другий Telegram-канал умовно через хвилину після першого, опублікував повідомлення «БпЛА з Шостки курсом на Новгород-Сіверський», і ось в цьому повідомленні рівень конкретики вже максимальний, адже міста, це конкретні точки на мапі, а не великі площі території.

Отримавши другий об'єкт з конкретними містами, створена система спочатку візьме список всіх активних цілей, для пошуку кандидатів, після чого відфільтрує і залишить лише тих кандидатів котрі мають такий самий тип як і новий об'єкт.

Отримавши список цілей того самого типу система розпочинає процес глибокої фільтрації та оцінювання для визначення найкращого кандидата на злиття. Спочатку перевіряється часовий інтервал: об'єкти можуть бути об'єднані лише якщо різниця в часі їх фіксації не більше 20 хвилин.

Також на цьому етапі система враховує сумісність типів загроз, допускаючи навіть об'єднання злиття типу «Ракета» та «Балістика», якщо повідомлення про них надійшли з різницею менше однієї хвилини, таке злиття зумовлене тим, що не всі Telegram-канали пишуть «балістична», «балістика» або «швидкісна» для оголошення що саме балістична ракета летить на якийсь населений пункт, часто використовується просто «ракета», через що і було реалізовано логіку злиття типу «ракета» та «балістика» якщо дані про ці об'єкти надійшла протягом однієї хвилини, і обидві цілі летять в одну точку.

Для відфільтрованих кандидатів застосовується жорстка просторова перевірка. Якщо для обох об'єктів (існуючого та нового) відомі точки старту та цілі, алгоритм вираховує їхній кут підльоту. У випадку, коли різниця цих векторних кутів перевищує 60 градусів, алгоритм класифікує рух цілей як з суттєво різних боків, тому розцінює їх як різні об'єкти та відхиляє злиття.

Кандидати, які успішно пройшли базові перевірки, переходять до етапу оцінювання, що дозволяє системі обрати найбільш достовірний збіг з кількох можливих. Алгоритм нарахування балів спирається на визначення рівня географічної точності об'єктів, який поділяється на три категорії: абстрактні напрямки (наприклад «Північ»), регіони (наприклад «Сумщина») та високоточні об'єкти (конкретні міста). Максимальну оцінку в 50 балів алгоритм нараховує за ідеальний текстовий збіг назв локацій. Якщо абсолютного збігу немає, система аналізує перетин рівнів точності:

- 40 балів - підтверджений просторовий збіг двох конкретних міст;
- 30 балів - збіг точного міста та відповідного широкого регіону;
- 10 балів - збіг абстрактних векторів руху;
- 1 бал - різниця в часі між публікаціями повідомлень становить не більше 5 хвилин.

Визначивши кандидата з найвищим балом, система ініціює фінальне злиття атрибутів. Під час цього процесу алгоритм детально порівнює рівень географічної точності для початкової та кінцевої точок маршруту обох об'єктів і динамічно залишає ту координату, яка має найвищу деталізацію. Таким чином, абстрактна «Чернігівщина» автоматично замінюється на конкретний «Новгород-Сіверський», при цьому система зберігає найраніший час виявлення об'єкта та максимальне значення кількості цілей у групі.

Якщо ж у процесі пошуку жодна з існуючих цілей не пройшла перевірок або не набрала прохідного балу, система остаточно розцінює вхідні дані як нову, незалежну загрозу та передає її на подальше самостійне відстеження.

3.3.11 Ідентифікація та супроводження цілей (Tracking Service)

Після етапу географічного розв'язання та нормалізації, оброблена інформація передається до ключового компонента системи - модуля трекінгу. Головне завдання модуля полягає у безперервному супроводі цілей, усуненні дублювання даних від різних джерел та побудові фізично достовірних траєкторій руху об'єктів у реальному часі.

Процес обробки нової цілі проходиться два етапи. Перший етап - це семантичне злиття. Оскільки інформація про одну й ту саму загрозу може надходити від різних Telegram-каналів але з різним рівнем деталізації, система спочатку виконує семантичне злиття цілей, після чого система стеження . Якщо алгоритм виявляє що нове повідомлення є лише уточненням даних про вже існуючий об'єкт - система виконує безшовне злиття, підчас якого змінюється лише курс цілі, а сама ціль лишається на тій позиції на якій була до злиття. При цьому існуючий об'єкт оновлює свої координати на більш точні, а етап фізичного прорахунку пропускається, завдяки чому економляться обчислювальні ресурси системи.

Перед тим як перейти до математичного моделювання для об'єктів які не злилися семантично, модуль виконує перевірку відстаней між поточною точкою та цільовою точкою об'єкта.

Наприклад, якщо відстань від поточної позиції до цільової більше ніж 300 кілометрів для регіонів або 150 кілометрів для міст, система відхиляє таку ціль, адже така відстань в повідомленні не реалістична і скоріш за все на етапі пошуку населених пунктів стався збій. Цей фільтр ігнорується лише для специфічних типів загроз, таких як балістичні ракети або КАБи, траєкторія яких майже не змінюється.

Другий етап трекінгу - це пошук найкращого кандидата на основі законів фізики та кінематики. Якщо нове повідомлення не є прямим дублікатом, алгоритм припускає, що це оновлення вже існуючої цілі. Для кожного кандидата з бази активних загроз система розраховує складну систему штрафних балів.

Спочатку перевіряється швидкість: алгоритм вираховує час що минув з моменту останньої фіксації, та фізичну відстань до нової точки. Якщо необхідна

для подолання цього відрізка швидкість перевищує максимально допустиму для цього типу цілей у півтора рази - кандидат одразу відхиляється. Для об'єктів, що знаходяться у стані спокою або втрати локації, система застосовує механізм нівелювання похибки до 10 кілометрів, адже зміна координат може бути викликана зміщенням зони покриття локаторів, а не реальним польотом.

Окрім швидкості, модуль ретельно аналізує зміну напрямку та цілі. Якщо активна загроза раптово змінює фінальну точку маршруту на інше місто, алгоритми накладають на такого кандидата жорсткий штраф. Натомість, якщо нова розрахована точка простору лише підтверджує рух до раніше визначеної цілі, кандидат отримує заохочувальний бонус у вигляді віднімання балів.

Також система розраховує кут розвороту об'єкта: якщо різниця векторів перевищує 120 градусів для дронів або 90 градусів для крилатих ракет, такий маневр вважається фізично неможливим, і кандидат відхиляється.

Останнім кроком оцінювання кандидатів є розрахунок майбутньої позиції: система математично моделює, де мала б знаходитися загроза прямо зараз і додає до штрафу відстань між майбутньою точкою та поточною позицією цілі.

Кандидат, який після всіх перевірок набирає найменшу кількість штрафних балів і вкладається у встановлений поріг допуску, визнається найкращим фізичним збігом. У такому разі його динамічна позиція оновлюється, а час відліку маршруту скидається для подальшого ведення.

Якщо ж жодна з існуючих цілей не пройшла перевірку фізикою та не вклалася у ліміти штрафів, алгоритм класифікує об'єкт як нову загрозу, і передає його без змін на збереження до репозиторія.

3.3.12 Репозиторій активних загроз та відкладене оновлення

Головним вузлом, куди стікаються всі оброблені дані, виступає клас репозиторій цілей. Фактично, об'єкт виступає глобальним сховищем стану системи. Під час масових ракетних атак повідомлення надходять безперервно, тому використання звичайних списків неминуче призвело б до помилок багатопоточності. Для уникнення цього архітектуру зберігання побудовано на базі

потокобезпечної колекції `ConcurrentDictionary`. Це дає змогу безпечно читати та оновлювати дані про загрози без ризику так званого стану гонитви. Замість того, щоб ставити жорсткі таймери на очищення пам'яті, було реалізовано власний фоновий планувальник `SchedulerLoop`.

`SchedulerLoop` працює наступним чином: самостійно вираховує час до найближчої події (умовно, підльоту БпЛА до цільового міста) і переходить у режим очікування на визначений інтервал, але не більше ніж на хвилину. Коли цикл прокидається, починається ревізія пам'яті. Діє жорстке правило — якщо об'єкт перебуває в активному стані понад 5 годин без оновлень, він автоматично видаляється з пам'яті без додаткових умов. Далі застосовуються контекстні перевірки.

Наприклад, якщо БпЛА кружляє над морем вже 20 хвилин, а тривогу на березі так і не дали, система сприймає його як зниклого і видаляє. Такий самий підхід діє для цілей, які опинилися глибоко в тилу без увімкнених сирен.

Окремої уваги заслуговує інтеграція з системою тривоги. Репозиторій інтегровано з подіями скасування повітряної тривоги в районах/областях. Як тільки в якійсь області дають відбій, алгоритм одразу перевіряє координати всіх активних загроз. Ті цілі, які територіально опиняються в зоні відбою, одразу списуються. Для цього було спеціально закладено допуск у 10 кілометрів, щоб випадково не втратити об'єкти, які летять по межах областей чи вздовж русел річок. Так як сервери періодично треба перезавантажувати, наприклад для оновлення, для запобігання втраті контексту відображення після кожного рестарту, поточний стан загроз серіалізується та зберігається у JSON-файли. Під час наступного запуску метод `LoadData` підтягує ці дані.

Проте він не просто переносить їх у пам'ять, а перевіряє кожну ціль заново. Адже цілком логічно, що поки сервер був недоступний, деякі загрози могли вже зникнути або втратити актуальність за законами кінематики.

На завершальному етапі кожної ітерації оновлення бази викликається метод надсилання оновлених даних користувачам. Він забирає вже готовий,

відфільтрований масив загроз і відправляє його всім клієнтам через SignalR. Саме завдяки цьому користувачі бачать рух цілей на екранах практично без затримок.

3.3.13 Клієнтська візуалізація та інтерполяція руху

Для візуалізації повітряних цілей на боці користувача, було використано такі бібліотеки як `deck.gl` та `MapLibre GL Js`. Завдяки використанню цих бібліотек система має змогу відображати величезну кількість цілей одночасно, адже вони були розроблені саме для високонавантажених інтерфейсів.

Також використання цих бібліотек значно покращило продуктивність самої мапи, адже обидві бібліотеки використовують графічний процесор для візуалізації, в той час як стандартний спосіб створення міток через `html` є дуже повільним та дорогим у використанні ресурсів.

Загальний алгоритм візуалізації полягає в тому, що при отриманні оновлення від сервера через SignalR, система передає масив нових даних у спеціальний метод обробки. Спочатку перевіряється унікальний ідентифікатор кожної цілі. Після чого система порівнює отриману ціль із локальним списком, де зберігаються всі поточні об'єкти.

Якщо ціль нова, для неї створюється запис: фіксується початкова точка, розраховується вектор руху (кут та загальну відстань). Також якщо, якщо сервер не передав базову швидкість цілі, алгоритм ставить 150 км/год для БПЛА або 800 км/год для ракет.

Якщо ж ціль вже існує в системі, ми оновлюємо її координати, колір забарвлення (у випадку зміни статусу) чи вектор руху. Щоб при оновленні маркер не "стрибав" різко по карті назад до серверної точки, для рухомих об'єктів ми перезаписуємо їхню початкову точку на поточну розраховану позицію. Це дозволяє продовжити рух максимально природно.

Оскільки сервер надсилає дані пакетами, а не безперервно, для їх плавного відображення було розроблено алгоритм клієнтської інтерполяції. Він працює у нескінченному циклі, який прив'язаний до частоти оновлення кадрів самого браузера.

Алгоритм функціонує за наступним принципом. Для кожної рухомої цілі ми беремо час її останнього оновлення і віднімаємо його від поточного часу.

Також ми обов'язково враховуємо різницю між годинником на сервері та на пристрої клієнта, щоб уникнути розсинхронізації даних. Отримавши чистий час у русі, ми множимо його на швидкість об'єкта. Отримуємо пройдену відстань.

Далі алгоритм вираховує прогрес руху (значення від 0 до 1) відносно загального маршруту цілі. Завдяки цьому прогресу ми обчислюємо точні проміжні координати цілі в конкретний кадр.

На фінальному етапі всі розраховані точки передаються у шар створений `deck.gl`. Цей шар вже самостійно відмальовує іконки на мапі, враховуючи їх розмір залежно від приближення карти, кількість об'єктів у групі та кут повороту.

3.4 Інфраструктурний рівень

3.4.1 Передача даних

Зважаючи на загальне визначення, інфраструктурний рівень описує технології та механізми, завдяки яким система має змогу функціонувати та взаємодіяти із зовнішнім середовищем.

Для забезпечення роботи системи «Контур» використовується значний спектр сучасних технологій, однією з ключових серед яких є бібліотека `SignalR` [1].

Окрім `SignalR`, архітектура системи передбачає використання класичних `REST API` кінцевих точок (`endpoints`). Це зумовлено необхідністю реалізації непомітної для користувача авторизації.

Оскільки цільовою платформою клієнтської частини є `Telegram Mini App`, система отримує об'єкт `initData`, який генерується месенджером [16]. Цей об'єкт використовується для валідації запиту: система перевіряє, чи дійсно він ініційований через офіційного `Telegram`-бота. У разі негативного результату запит відхиляється, що слугує захистом від злоумисників, які намагаються отримати прямий доступ до `API`.

Використання REST API є доцільним саме для процесу авторизації, оскільки він працює за принципом одноразового запиту без збереження стану (stateless), тоді як SignalR відповідає за підтримку довготривалого динамічного з'єднання. Після успішної одноразової авторизації через кінцеву точку REST API система встановлює постійний канал зв'язку для передачі оперативних даних.

3.4.2 Збереження даних

Оскільки система оперує безперервним та динамічним потоком інформації, виникає необхідність її надійного збереження. У випадку непередбачуваного перезавантаження сервера система повинна мати змогу відновити стан оброблених цілей.

Крім того, модульна архітектура «Контур» передбачає наявність специфічних налаштувань для кожного компонента.

Наприклад, сервіс географічної нормалізації використовує власний словник синонімів.

Оскільки єдиним джерелом інформації для парсера виступають моніторингові Telegram-канали, адміністратори яких часто використовують неформальні, скорочені або "народні" назви населених пунктів, система потребує механізму їх точної ідентифікації. Створений словник забезпечує зіставлення таких синонімів із їхніми офіційними географічними назвами.

Для збереження налаштувань та оперативних даних у проєкті свідомо обрано формат JSON-файлів у комбінації зі зберіганням в оперативній пам'яті [19]. Однією з головних нефункціональних вимог до системи є ефективна робота на ресурсообмеженому обладнанні.

Використання традиційних реляційних баз даних (RDBMS) створило б додаткове постійне навантаження на процесор та оперативну пам'ять сервера через необхідність підтримки роботи бази даних.

Відмова від СУБД на користь структурованих JSON-файлів та In-Memory кешування дозволила значно пришвидшити швидкість доступу до даних та

вивільнити системні ресурси для виконання більш пріоритетних операцій з обробки просторових координат.

3.4.3 Контейнеризація Docker

Для забезпечення стабільної роботи системи та її легкого розгортання було обрано технологію контейнеризації Docker [9].

Зважаючи на те, що «Контур» є комплексною системою з різними залежностями, дуже важливо гарантувати ідентичність середовища як на етапі локальної розробки, так і на віддаленому сервері.

Використання Docker дозволило запакувати весь серверний застосунок разом із необхідним середовищем .NET у єдиний ізольований контейнер. Це виключає можливі конфлікти програмного забезпечення та значно спрощує процес розгортання нових версій, дозволяючи системі працювати стабільно та прогнозовано незалежно від базової конфігурації системи хоста.

3.4.4 Платформа розгортання

Як апаратну платформу для розміщення серверної частини було обрано віртуальний приватний сервер (VPS) від хмарного провайдера Microsoft Azure [10]. Оскільки система візуалізації повітряних загроз вимагає безперервного доступу до мережі та високого показника доступності, розміщення на звичайному локальному обладнанні несло б критичні ризики, пов'язані з перебоями електропостачання чи інтернет-з'єднання.

Хмарне рішення від Azure забезпечує системі гарантовану стабільність, масштабовані обчислювальні ресурси та високу пропускну здатність мережевого каналу, що є життєво необхідним для обробки постійного потоку даних та великої кількості одночасних з'єднань клієнтів.

3.4.5 Інфраструктурна безпека

Окремої уваги заслуговує питання інфраструктурної безпеки та мережевої оптимізації, для реалізації яких було залучено сервіс Cloudflare [15].

Оскільки система є публічно доступною та оперує соціально важливою інформацією в реальному часі, вона є потенційно вразливою до кібератак, зокрема розподілених атак на відмову в обслуговуванні (DDoS). Cloudflare налаштовано як зворотний проксі-сервер (reverse proxy), що виступає "щитом" між кінцевими користувачами та сервером Azure. Він автоматично фільтрує підозрілий та шкідливий трафік ще до того, як той досягне обчислювальних потужностей системи.

Додатковою перевагою є використання мережі доставки контенту (CDN), завдяки якій статичні ресурси клієнтського інтерфейсу кешуються на серверах Cloudflare. Це не лише пришвидшує первинне завантаження мапи для користувачів, але й суттєво зменшує загальне навантаження на основний сервер системи.

3.4.6 Підсистема адміністрування та моніторингу

Для ізоляції інтерфейсу керування налаштуваннями системи та моніторингу загального стану системи, реалізовано адміністративний веб-сайт на базі фреймворку Blazor [17]. Веб-сайт розгортається локально на комп'ютері адміністратора.

Для авторизації в системі, і отримання прав на редагування та будь яку взаємодію з файлами налаштувань, реалізовано додатковий варіант авторизації через адміністраторський код доступу.

Адміністраторський веб-сайт, дає змогу виконувати наступні операції:

- перегляд статистики відвідувань мапи користувачами;
- перегляд статистики кількості загроз;
- перегляд історії загроз;
- редагування «чорного списку» слів для географічного нормалізатора;
- редагування списку токенів загрози та додавання синонімів до певного типу загрози;
- редагування токенів географічних напрямків руху;

- редагування словника синонімів міст;
- редагування символів для позначення токенів (використовуються для генерації сигнатури токенізованого повідомлення);
- перегляд та видалення правил парсингу;
- редагування загальних токенів;
- перегляд списку активних цілей з можливістю видалення будь-якої з них;
- ручна перевірка правил парсингу;
- перегляд мапи;
- зміна режиму затвердження правил;
- перегляд кількості активних користувачів.

Взаємодія між адміністраторським веб-сайтом та серверною частиною системи відбувається як через REST API кінцеві точки, так і через SignalR зв'язок. Завдяки чому адміністратор оперативно отримує інформацію, та має змогу своєчасно виконувати відповідні дії.

4 ДЕМОНСТРАЦІЯ РЕЗУЛЬТАТІВ ТА ТЕСТУВАННЯ

4.1 Демонстрація клієнтського веб-застосунку

Для зручності використання застосунку в будь-який час доби, було реалізовано два варіанти теми застосунку – темну та світлу.

Для кращої видимості цілей на мапі, використано акцентні кольори, котрі виділяються на фоні мапи, та активної підсвітки районів в котрих оголошено повітряні тривоги. Нижче продемонстровано ключові екранні форми клієнтського застосунку.

На рисунку 4.1 наведено приклади головного екрана системи у світлій та темній темах оформлення. Наявність двох кольорових тем дозволяє адаптувати інтерфейс для комфортного використання в будь-який час доби.

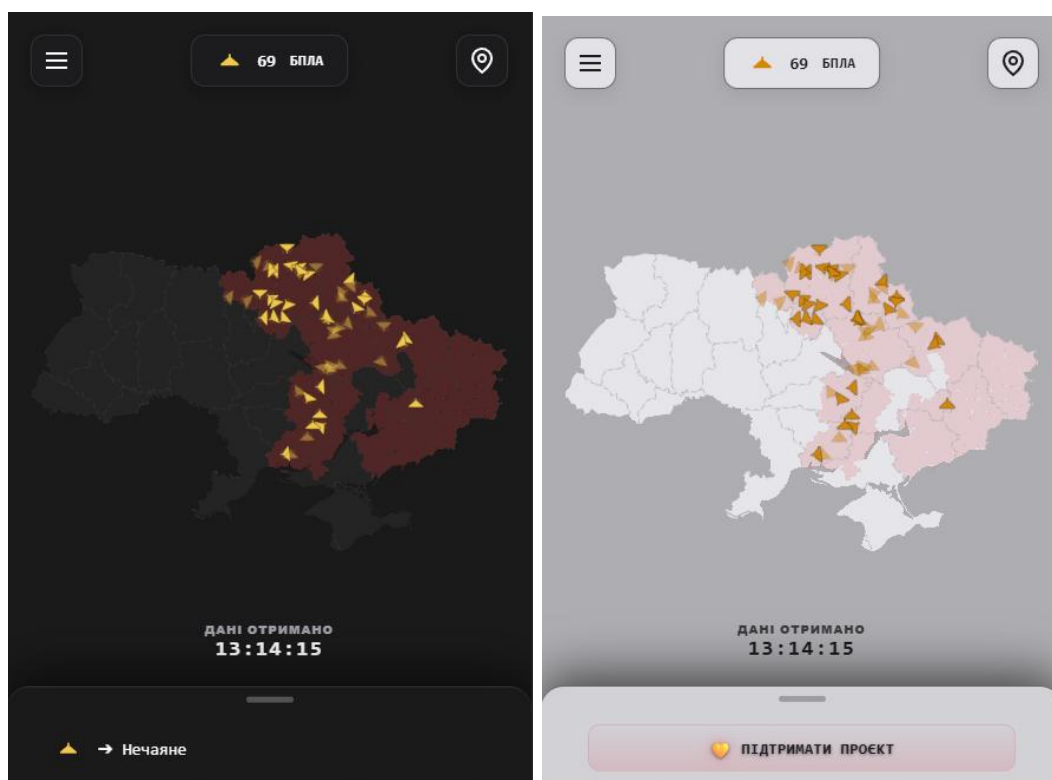


Рис. 4.1 Головний екран, в темній та світлій темі

Основним елементом головного екрана є інтерактивна мапа. Для швидкої ідентифікації типу цілі застосовано метод колірної диференціації: від жовтого кольору для позначення БПЛА до червоного — для балістичних ракет, що асоціюється з найвищим рівнем небезпеки.

Для детального аналізу ситуації користувач має змогу переглядати список актуальних та минулих загроз, розгорнувши панель у нижній частині екрана (див. рис. 4.2). Також користувач має змогу переглянути інформацію про конкретну загрозу натиснувши на стрілку з правого боку панелі цілі.

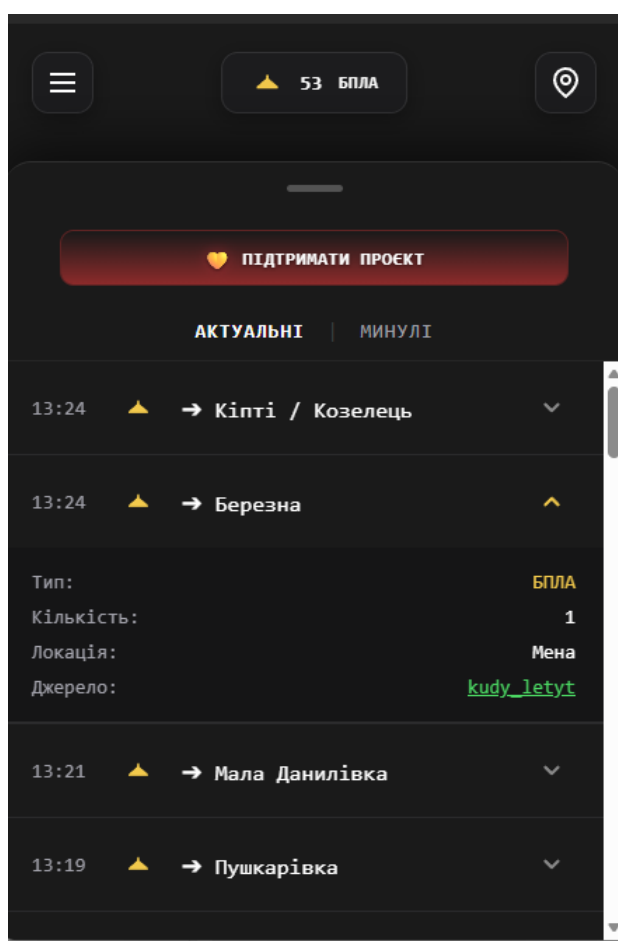


Рис. 4.2 Панель історії цілей

З метою інформування користувача про актуальність даних, над панеллю історії розміщено індикатор часу останнього оновлення інформації від сервера. Для швидкої оцінки загального масштабу небезпеки у верхній частині екрана реалізовано лічильник активних цілей із розбивкою за їх типом. При взаємодії з

цим лічильником на екран виводиться легенда мапи (див. рис. 4.3), у якій наведено вичерпний перелік усіх типів загроз, що обробляються системою, разом із їхніми графічними іконками та кольором.

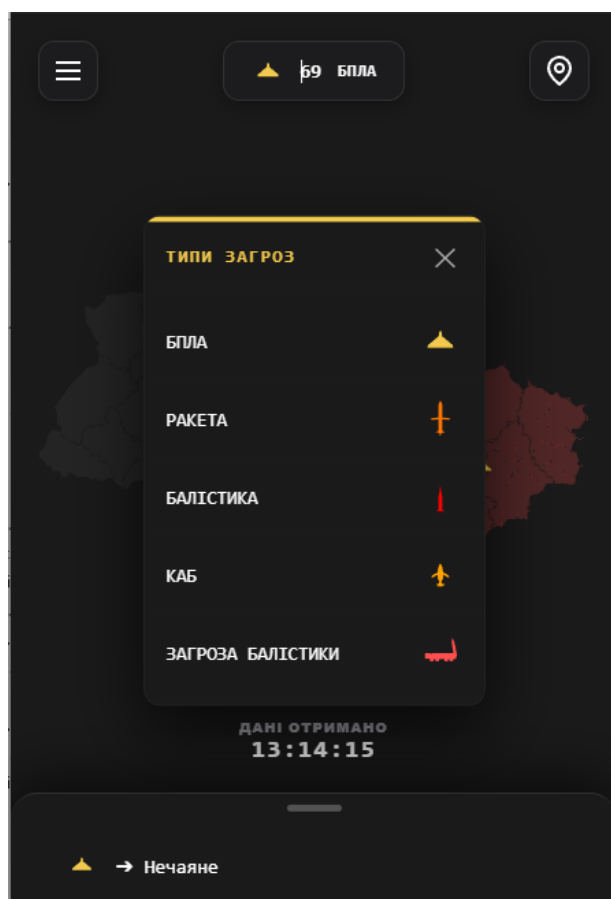


Рис. 4.3 Легенда мапи

У верхньому лівому куті на головному екрані присутня кнопка з трьома горизонтальними лініями, натиснувши на яку користувач перейде у меню проекту (див. рис. 4.4). А у верхньому правому куті екрану присутня клавіша «Мое місцезнаходження», при натиску на неї на мапі відобразиться місце де в поточний момент знаходиться користувач. Потреба в цій кнопці виникла через те, що підчас масованих атак, не всі користувачі перебуваючи під впливом психологічного тиску, здатні швидко зорієнтуватись де вони знаходяться.

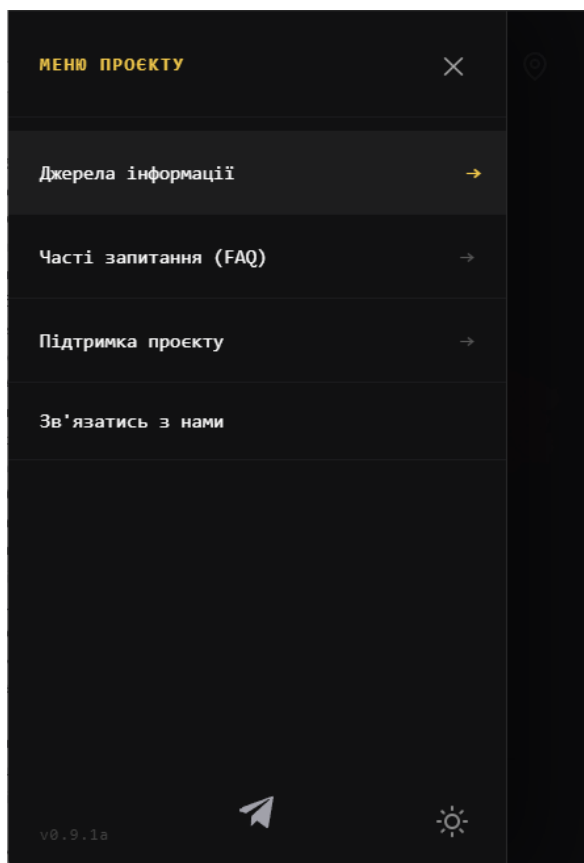


Рис. 4.4 Меню проєкту

В меню проєкту користувач має змогу відкрити сторінку списку джерел (див. рис. 4.5). На сторінці списку джерел, виділено два блоки «Пріоритетні джерела» та «Додаткові джерела».

В списку пріоритетних джерел, розміщені Telegram-канали дані в котрих з'являються регулярно та структуровано. Пріоритетні джерела створюють основний потік інформації в систему, тому вони виділені окремим блоком.

В списку додаткових джерел, розміщено Telegram -канали котрі мають дещо хаотичний формат даних, та потік інформації від них не є постійним. Також в блоці додаткових джерел розміщуються канали котрі поширюють інформацію по конкретному регіону, а не по всій країні.

Також користувач має змогу підтримати будь яке з джерел інформації, для цього є окрема кнопка поруч з назвою Telegram-каналу. Або просто перейти в обраний канал і стати його підписником – для цього користувачу достатньо натиснути на назву Telegram-каналу.

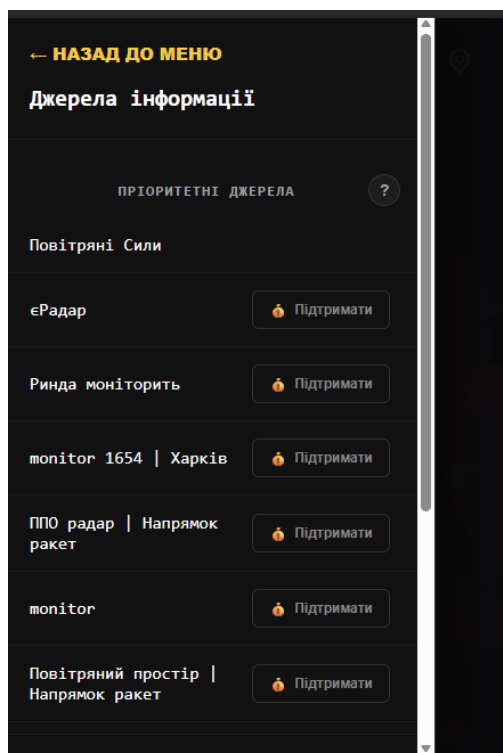


Рис. 4.5 Сторінка списку джерел

Також з меню проекту користувач має змогу перейти на сторінку частих запитань (див. рис. 4.6).

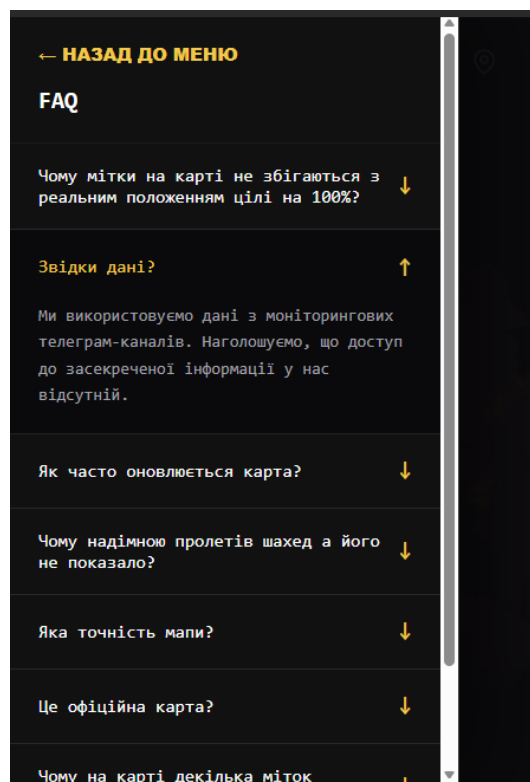


Рис. 4.6 Сторінка частих запитань

Для тих користувачів котрі бажають підтримати розвиток проекту, окремою сторінкою виділено доступні способи підтримки (див. рис. 4.7). Також присутня кнопка котра ініціює перехід до сервісу прийому благодійних внесків.

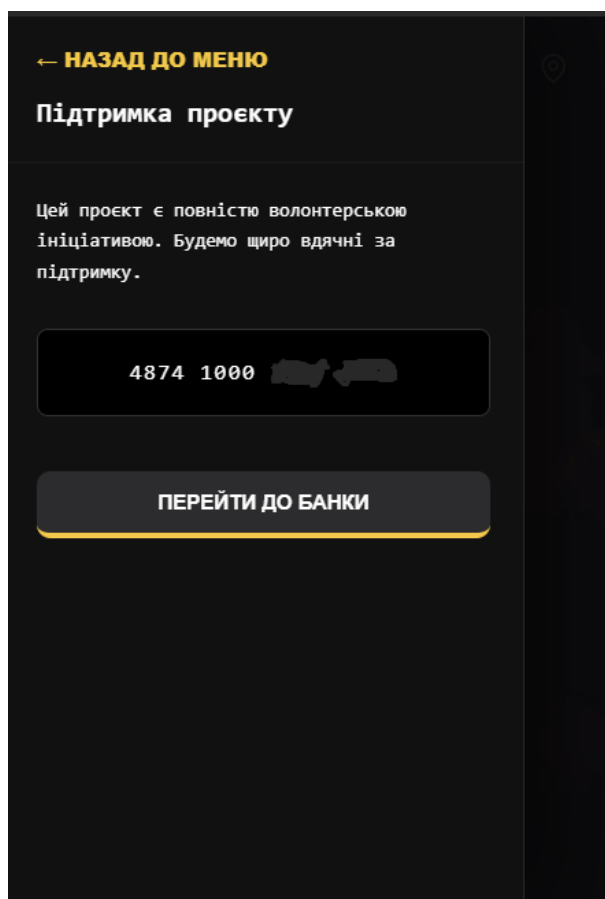


Рис. 4.7 Сторінка підтримки проекту

Також в меню проекту присутня кнопка «Зв'язок з нами», завдяки якій користувач може зв'язатись з адміністратором для вирішення проблем, або отримання відповіді на запитання. З самого низу меню, присутня кнопка з іконкою «паперовий літак», при натиску на яку, користувачу відкриється Telegram-канал з інформацією та новинами проекту. А також знизу меню присутній номер версії застосунку.

Для зміни кольорової теми, знизу меню є відповідна кнопка котра має іконку сонця або місяця (залежить від поточної теми).

4.2 Демонстрація адміністраторського веб-сайту

На головному екрані адміністраторського веб-сайту, відображаються графіки кількості цілей, графік кількості користувачів, кількість наявних правил парсингу, та підсумкова статистика по кількості користувачів за годину, день та найбільша кількість користувачів за весь час (див. рис. 4.8).

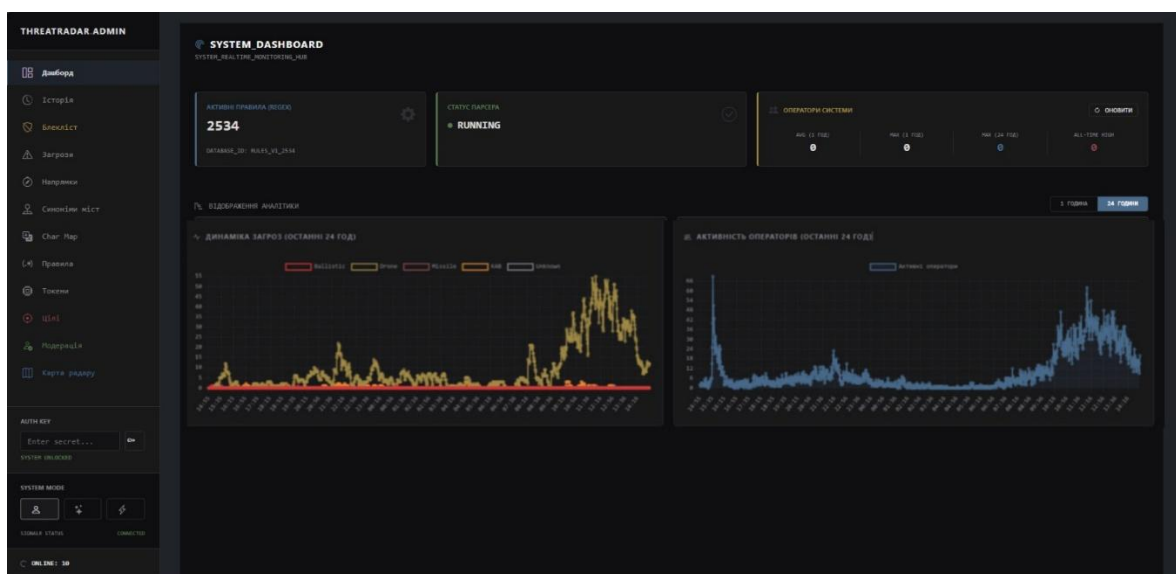


Рис. 4.8 Головний екран адміністраторського веб-сайту

Наступною сторінкою веб-сайту є «Історія цілей», на цій сторінці адміністратор має змогу переглянути повну історію цілей за останній день (див. рис. 4.9). Також є змога відкрити детальну інформацію про кожну ціль (див. рис. 4.10). В картці інформації відображається вся доступна інформація, така як:

- тип цілі;
- кількість;
- час фіксації;
- сире повідомлення, та сигнатура його токенозованого вигляду;
- стартова точка;
- цільова точка.

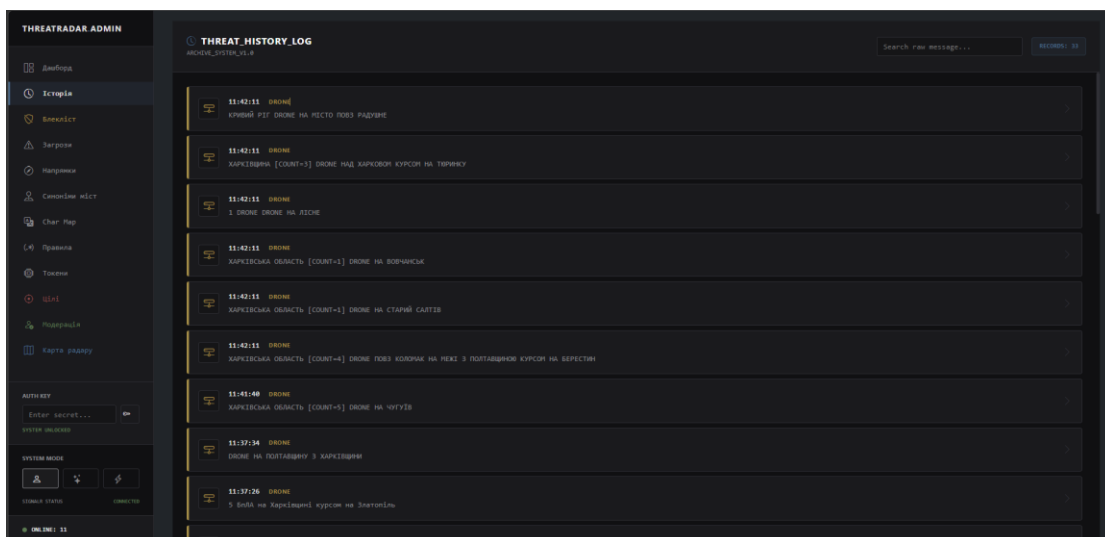


Рис. 4.9 Сторінка історії цілей

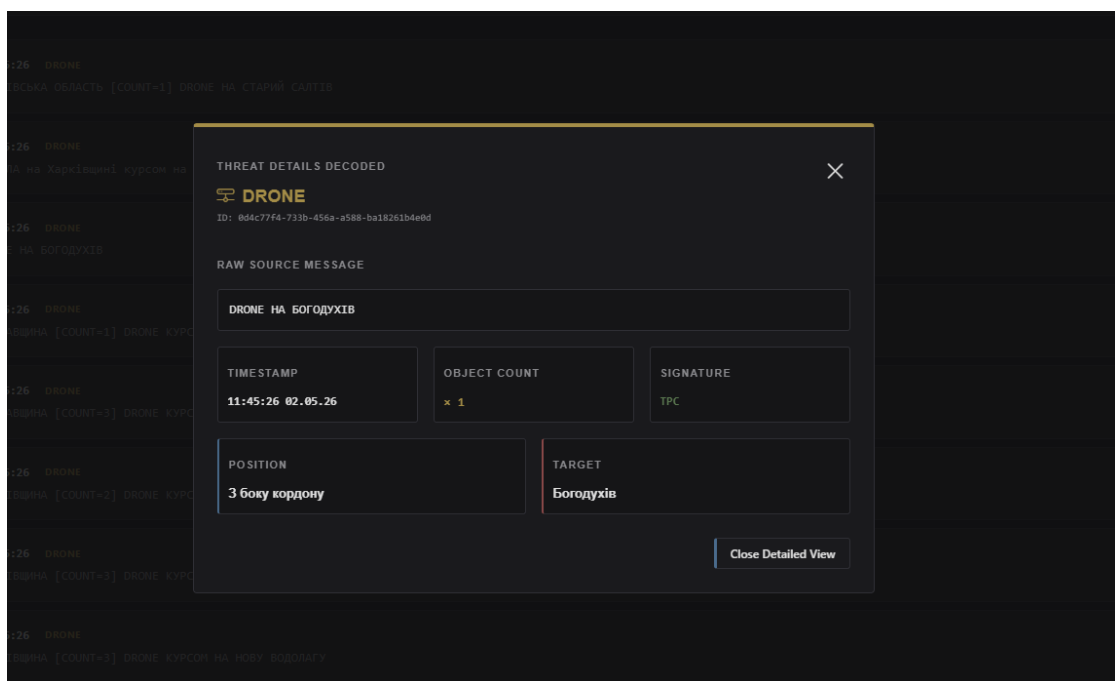


Рис. 4.10 Вікно детальної інформації про ціль

Однією з важливих сторінок для коректної роботи системи, є сторінка «чорний список», адже саме на цій сторінці відбувається взаємодія зі словами, котрі система не повинна навіть пробувати нормалізувати в географічну точку (див. рис. 4.11). Потреба в цій сторінці виникла через велику кількість слів, котрі мають велику схожість з назвами реальних населених пунктів, через що, система могла повертати не ту географічну точку про яку йшлося в повідомленні.

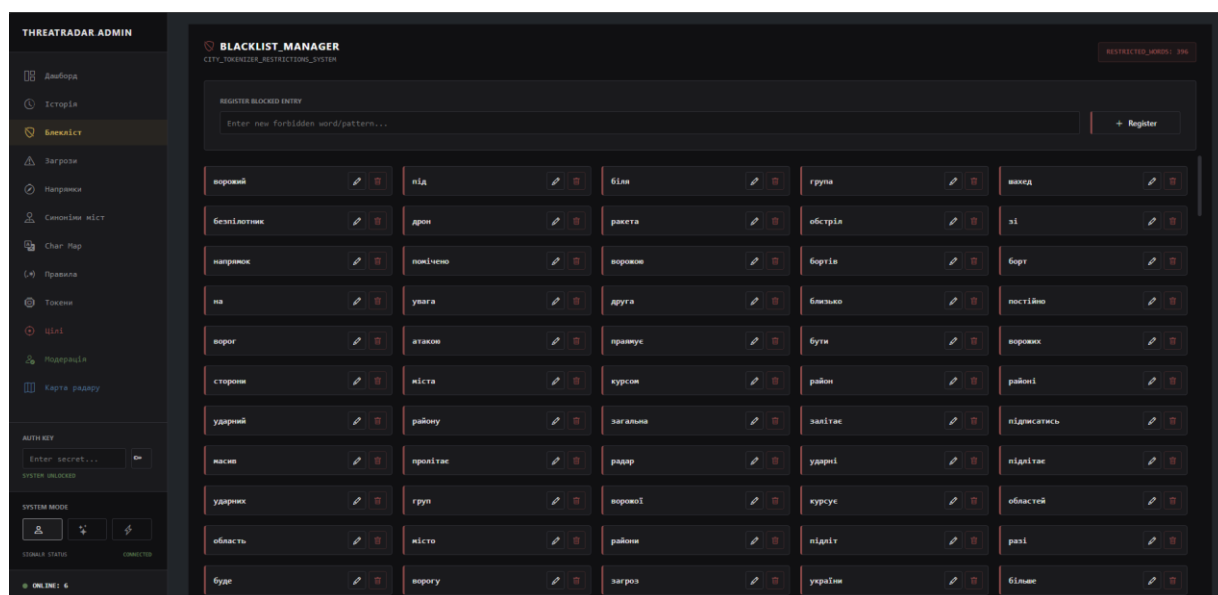


Рис. 4.11 Сторінка чорного списку слів

Адміністраторський веб-сайт також надає змогу створювати, редагувати, видаляти та розширювати списки синонімів для токенів (див. рис. 4.12):

- тип цілі;
- географічний напрямок;
- синоніми міст;
- семантичні токени.

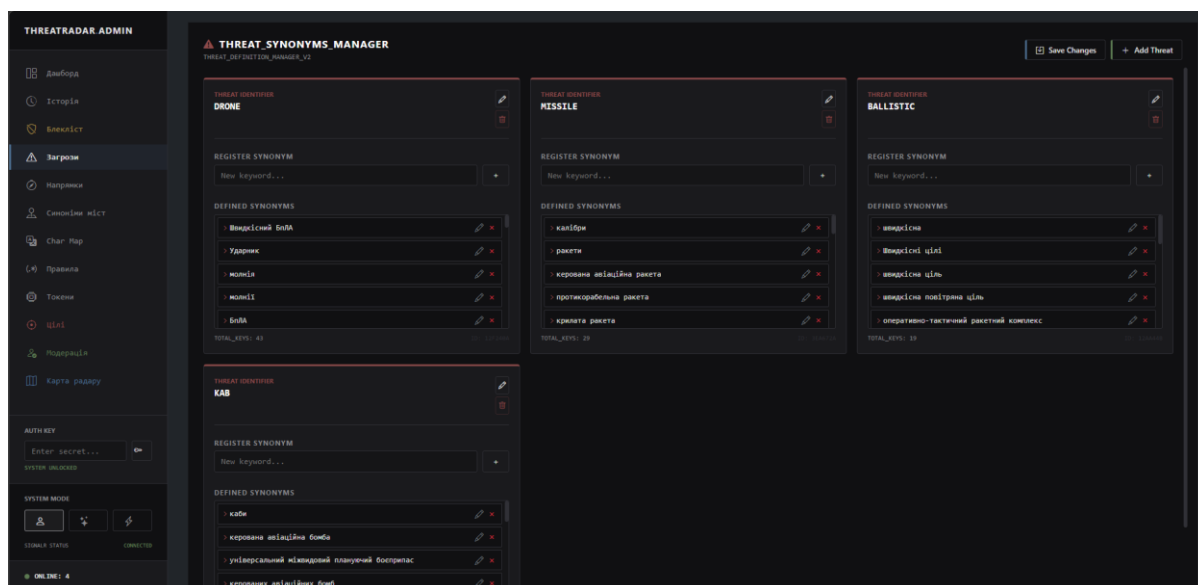


Рис. 4.12 Сторінка редагування токенів типу цілі

Функціонал всіх чотирьох сторінок: тип цілі, географічний напрямок, синоніми міст та семантичні токени, є повністю однаковим. Зовнішня складова також ідентична, єдина відмінність це дані з якими вони працюють та внутрішня логіка обробки змін.

На адміністраторському веб-сайті також присутня можливість переглядати та видаляти правила парсингу (див. рис. 4.13).

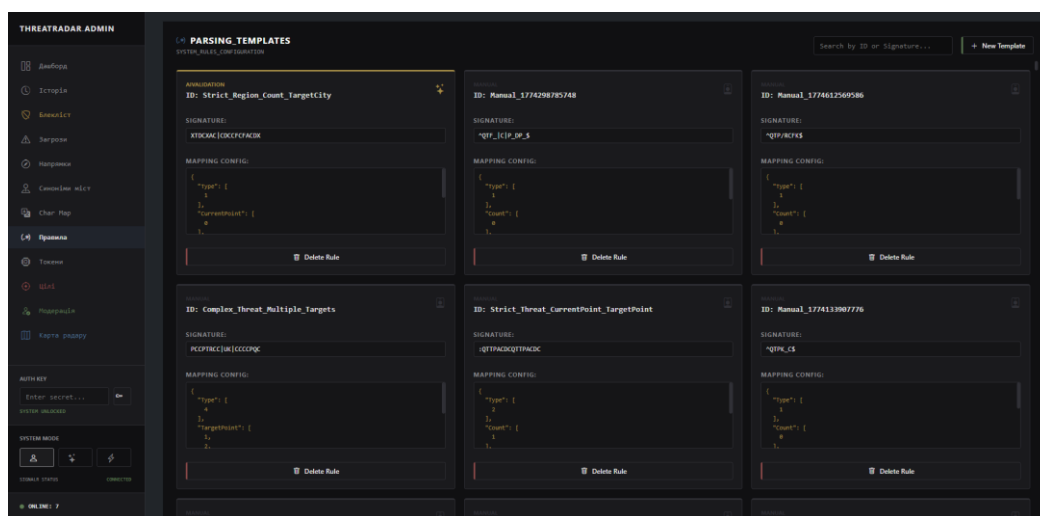


Рис. 4.13 Сторінка правил парсингу

Для перегляду актуального списку всіх повітряних цілей, реалізована сторінка «Цілі» (див. рис. 4.14).

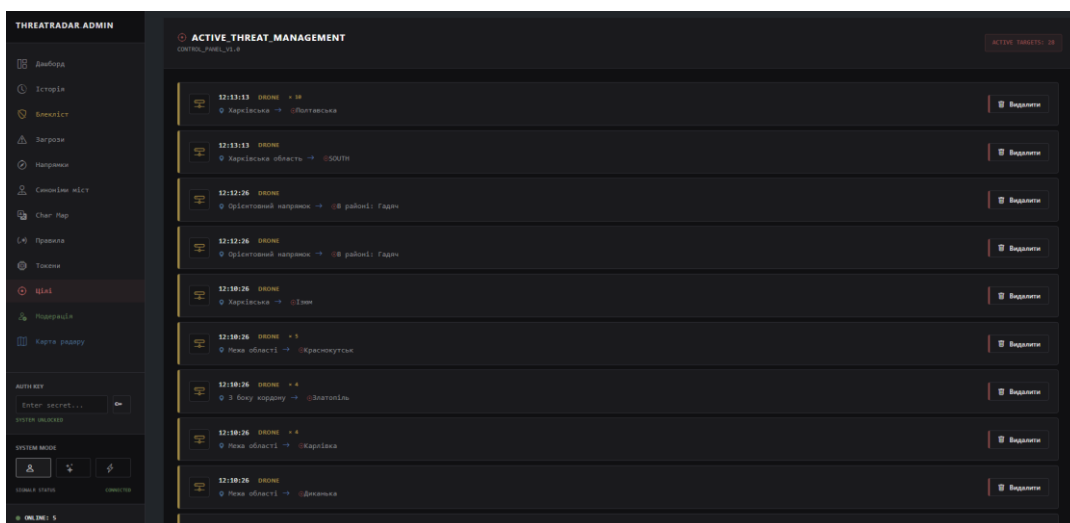


Рис. 4.14 Сторінка актуальних цілей

Найважливішою сторінкою адміністраторського веб-сайту, є сторінка перевірки нових правил парсингу (див. рис. 4.15).

На першому екрані з правого боку присутня історія перевірок правил парсингу, а по центру екрану зображено саме правило котре потрібно перевірити. Вище над правилом парсингу окремими блоками зображено токенизований вигляд повідомлення, та сире повідомлення.

Для визначення дії з правилом, адміністратор має чотири кнопки дії:

- прийняти: використовується для підтвердження того, що правило згенеровано вірно;
- регенерація: використовується для виклику повторної генерації правила через проблеми з розміткою токенів в поля;
- ручне редагування: відкриває вікно редактору правил для створення правила парсингу конкретно під цю сигнатуру повідомлення (див. рис. 4.16);
- скасувати повідомлення: повністю відмінляє обробку повідомлення системою. Використовується якщо система помилково пропустила неінформативне повідомлення, або не змогла вірно визначити географічні точки/населені пункти.

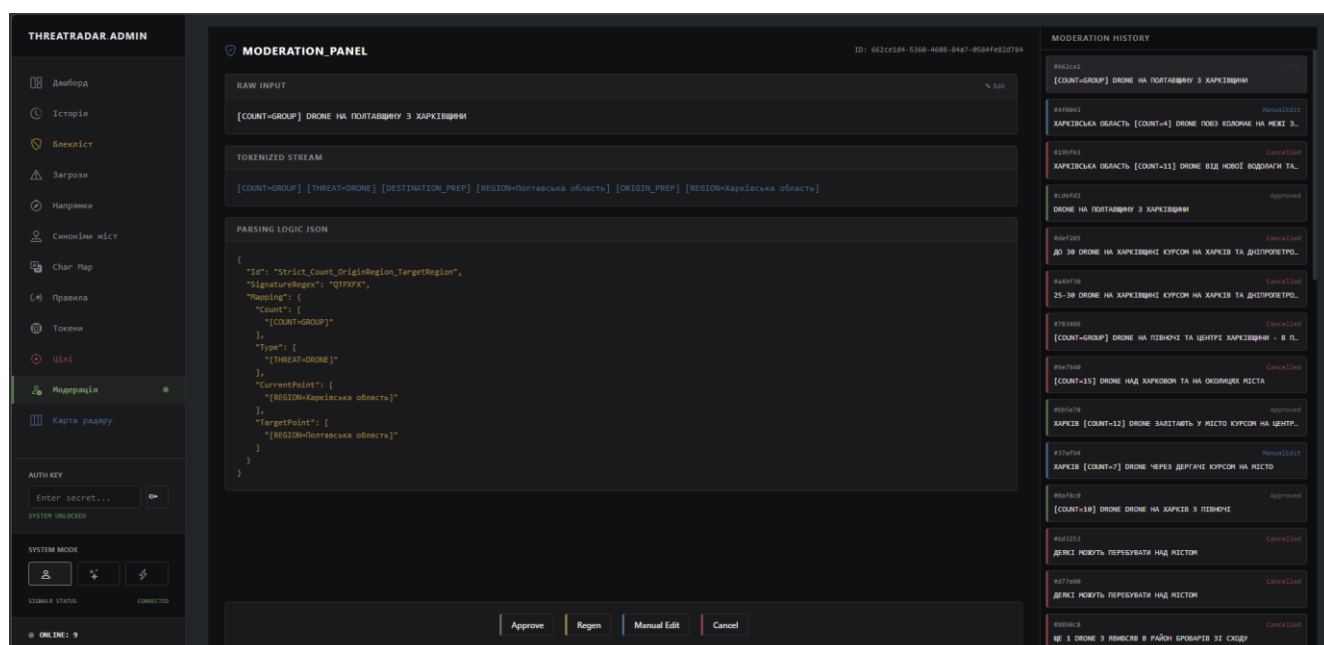


Рис. 4.15 Головний екран перевірки правил

У випадку якщо правило парсингу було згенеровано не вірно декілька разів, і потребує ручного створення, адміністратор має змогу відкрити сторінку створення правил парсингу натиснувши на відповідну кнопку знизу головного екрану перевірки правил (див. рис. 4.16).

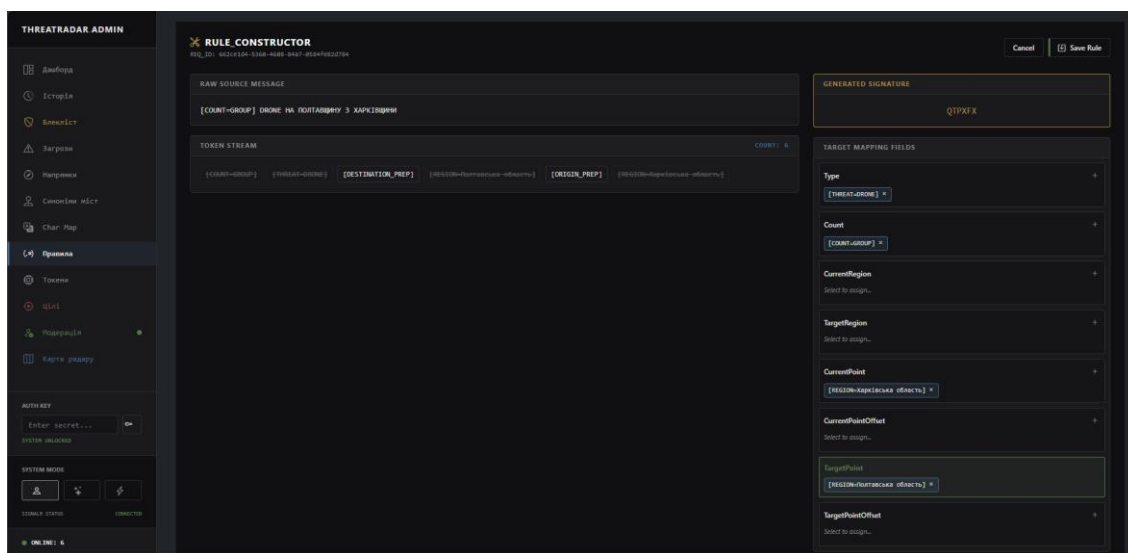


Рис. 4.16 Сторінка створення правил парсингу

Останньою сторінкою адміністраторського веб-сайту є мапа. На цій сторінці в блоці iframe відображається та сама мапа цілей, що і у всіх користувачів (див. рис. 4.17). Також є можливість відкрити мапу в режимі «зображення в зображенні».

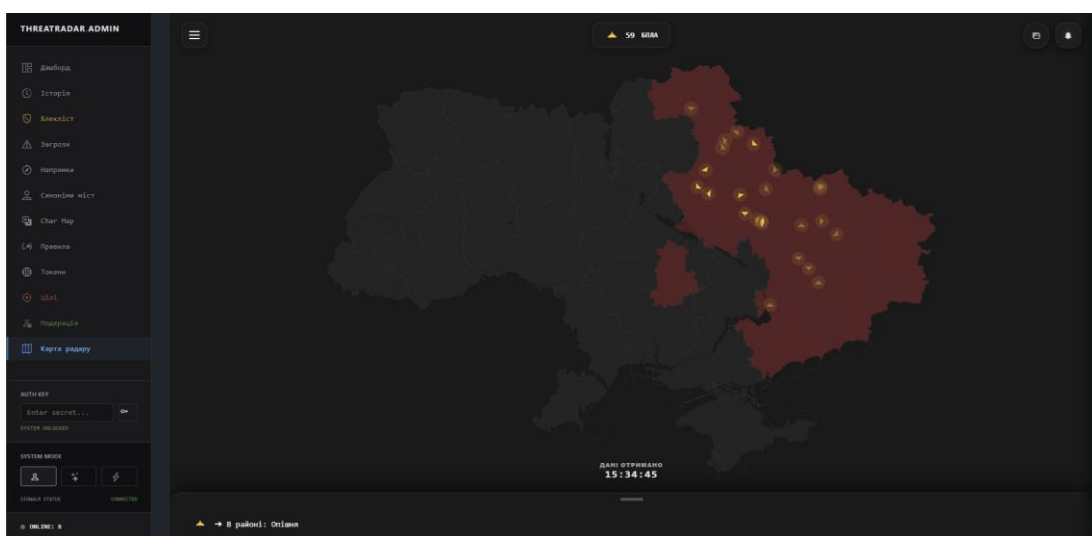


Рис. 4.17 Сторінка мапи повітряних цілей

4.3 Тестування системи

Для перевірки коректності роботи всіх модулів, було проведено наскрізне тестування, з використанням складного повідомлення, з різним формулюванням, типами цілей та напрямками. Метою є демонстрація того, як неструктурований текст зі складним синтаксисом, кількома векторами та географічними колізіями перетворюється на фінальні об'єкти на мапі.

У тестовий Telegram-канал надсилається повідомлення високої складності. Воно містить конкретні географічні точки цілей, цілі з відсутньою стартовою точкою, цілі з декількома векторами атаки міст, запутану структуру повідомлення:

*«БпЛА з півночі та сходу на Лебедин
на Куп'янськ з півночі ракета
КАБ на Семенівку
БпЛА з моря на Одесу
на Харків швидкісна ціль
БпЛА Чернігова курсом на Ніжин,
Іх БпЛА на Суми та ще один на Чернігів з півночі»*

Таке повідомлення не можливо було б розібрати на дані завчасно створеним шаблоном, але створена система, завдяки самонавчанню має змогу працювати з таким повідомленням.

Першим етапом обробки повідомлення є токенізація. Саме на цьому етапі, повідомлення перетворюється з людської мови, в детерміновану форму. Тестове повідомлення, після етапу токенізації зображено на рисунку 4.18.

```
=====
[Pipeline] FINAL CLEARED TOKEN TEXT: [THREAT=DRONE] [ORIGIN_PREP] [CARDINAL_DIRECTION=NORTH] [AND] [CARDINAL_DIRECTION=
EAST] [DESTINATION_PREP] [CITY=Лебедин]
[DESTINATION_PREP] [CITY=Куп'янськ] [ORIGIN_PREP] [CARDINAL_DIRECTION=NORTH] [THREAT=MISSILE]
[THREAT=KAB] [DESTINATION_PREP] [CITY=Семенівка]
[THREAT=DRONE] [ORIGIN_PREP] [CITY=Море] [DESTINATION_PREP] [CITY=Одеса]
[DESTINATION_PREP] [CITY=Харків] [THREAT=BALLISTIC]
[THREAT=DRONE] [CITY=Чернігів] [DIRECTION_MARKER] [CITY=Ніжин] [AND] [COUNT=1] [THREAT=DRONE] [DESTINATION_PREP] [CITY=
Суми] [AND] [ADDITIVE] [DESTINATION_PREP] [CITY=Чернігів] [ORIGIN_PREP] [CARDINAL_DIRECTION=NORTH]
[Pipeline] Received TOKEN message: [THREAT=DRONE] [ORIGIN_PREP] [CARDINAL_DIRECTION=NORTH] [AND] [CARDINAL_DIRECTION=EA
```

Рис. 4.18 Токенізований вигляд тестового повідомлення

Наступним етапом обробки повідомлення, є класифікація його складності, так як це тестове повідомлення, воно свідомо писалось під «Багатовекторну» складність, щоб наочно показати роботу системи, з не типовими повідомленнями.

Для визначення складності повідомлення використовується клас *TokenProcessor* та його метод *ClassifyComplexity*.

Тестове повідомлення отримало рівень складності MultiVector (див. рис. 4.19).

```
[Pipeline] Event: [THREAT=DRONE] [ORIGIN_PREP] [CARDINAL_DIRECTION=NORTH] [AND] [CARDINAL_DIRECTION=EAST] [DESTINATION_PREP] [CITY=Лебедин]
[DESTINATION_PREP] [CITY=Куп'янськ] [ORIGIN_PREP] [CARDINAL_DIRECTION=NORTH] [THREAT=MISSILE]
[THREAT=KAB] [DESTINATION_PREP] [CITY=Семенівка]
[THREAT=DRONE] [ORIGIN_PREP] [CITY=Море] [DESTINATION_PREP] [CITY=Одеса]
[DESTINATION_PREP] [CITY=Харків] [THREAT=BALLISTIC]
[THREAT=DRONE] [CITY=Чернігів] [DIRECTION_MARKER] [CITY=Ніжин] [AND] [COUNT=1] [THREAT=DRONE] [DESTINATION_PREP] [CITY=Суми] [AND] [ADDITIVE] [DESTINATION_PREP] [CITY=Чернігів] [ORIGIN_PREP] [CARDINAL_DIRECTION=NORTH]
[Pipeline] Complexity: MultiVector
```

Рис. 4.19 Рівень складності тестового повідомлення

Відповідно до закладеної логіки, повідомлення з рівнем складності MultiVector надсилаються штучному інтелекту, для поділу його на окремі незалежні події, так як їх структура є заплутаною, та не підлягає поділу на події з використанням реалізованих алгоритмів.

Результатом роботи штучного інтелекту, є набір наступних подій:

- БпЛА зі сходу на Лебедин;
- БпЛА з півночі на Лебедин;
- ракета з півночі на Куп'янськ;
- КАБ на Семенівку;
- БпЛА з моря на Одесу;
- швидкісна ціль на Харків;
- БпЛА з Чернігова курсом на Ніжин;
- 1х БпЛА на Суми;
- 1х БпЛА з півночі на Чернігів.

Завдяки такому поділу, складного повідомлення, на окремі незалежні події, система має змогу опрацювати кожну з них використовуючи власні алгоритми.

В результаті роботи, було створено 9 цілей, кожна з котрих має всі дані котрі вказано було в повідомленні. А ті цілі котрі не мали початкових точок руху, для них було застосовано алгоритми пошуку місця старту, завдяки чому:

- КАБ котрий летів на Семенівку створився з боку кордону, адже КАБи пускають лише з тилу ворожої сторони;
- швидкісна ціль котра летіла на Харків була створена з боку кордону, адже зазвичай пуски балістики по Харкову ведуться зі сторони Белгорода.;
- БпЛА на Суми створився з боку кордону, адже місто Суми знаходиться близько до кордону та пускового майданчика БпЛА, а інших БпЛА поруч не було.

Результатом роботи системи, є цілі, розміщені відповідно до згадок населених пунктів/напрямоків в текстовому повідомленні (див. рис. 4.20).



Рис. 4.20 Результат обробки системою тестового повідомлення

4.4 Порівняння з існуючими системами

Для достовірного порівняння систем, було обрано лише ті параметри, котрі можливо перевірити не маючи прямого доступу до серверної частини систем. В розрахунок не брався такий параметр як «час обробки повідомлення», адже визначити його у всіх трьох існуючих системах – не можливо, в зв'язку з тим, що у всіх трьох систем відсутній рух цілей, а після оновлення значна кількість цілей телепортується, і дізнатись чи це нова ціль, чи стара – не можливо.

Проведений аналіз за доступними параметрами, наведено в таблиці 4.1. Порівняння проводилось за такими параметрами:

- затримка синхронізації даних;
- кількість елементів інтерфейсу;
- кількість типів цілей;
- візуалізація руху;
- середовище виконання.

Таблиця 4.1

Зведені дані порівняння систем моніторингу інформації про повітряні цілі

Критерій оцінювання	«Neptun»	«Dimap»	«VARTA AI»	«КОНТУР»
Затримка синхронізації даних	~ 30 с	15 – 25 с	~5 с	0.1 – 3 с
Кількість елементів інтерфейсу	12	~20	~16	6
Кількість типів цілей	2 типи	2 типи	Без класифікації	4 типи
Візуалізація руху	Телепортація	Телепортація	Відсутній	Плавна інтерполяція

Аналіз результатів, наведених у таблиці 4.1, дозволяє зробити висновок про технічну та експлуатаційну перевагу розробленої системи «КОНТУР» над існуючими аналогами. Розроблена система не лише виправляє недоліки існуючих рішень у частині візуалізації даних, а й пропонує новий підхід до побудови інтерфейсів моніторингових систем, орієнтований на швидкість зчитування інформації.

ВИСНОВКИ

У межах кваліфікаційної роботи реалізовано систему інтелектуальної візуалізації повітряних загроз у реальному часі «Контур». Аналіз існуючих рішень (Neptun, Dimap, VARTA AI) виявив їхні критичні недоліки: затримки синхронізації через HTTP Polling та проблему AI-галюцинацій. Для вирішення цих проблем запропоновано відмовостійку гібридну архітектуру, що суттєво знижує когнітивне навантаження на користувачів.

Серверну логіку реалізовано на платформі .NET 9 (C#) із застосуванням протоколу SignalR для миттєвої двосторонньої передачі даних. З метою оптимізації роботи на ресурсообмеженому обладнанні систему побудовано без реляційних СУБД, використовуючи In-Memory кешування та JSON-файли. Клієнтську частину розгорнуто як Telegram Mini App, де за допомогою бібліотек MapLibre GL JS та deck.gl реалізовано плавне апаратне відображення об'єктів з інтерполяцією руху.

Ключовим науково-практичним рішенням стала розробка багаторівневого конвеєра обробки даних (Pipeline) із гібридним парсингом. Штучний інтелект у системі залучається виключно як генератор та валідатор нових шаблонів, тоді як поточний розбір повідомлень залишається суворо детермінованим. Створена математична DTO-модель трекінгу враховує фізичну логіку (швидкість, вектор), що усуває аномальні стрибки міток на мапі, а сервіс GeoNormalizer успішно розв'язує топонімічні колізії.

Наскрізне тестування (End-to-End) підтвердило точну декомпозицію складних багатовекторних повідомлень. Надійність інфраструктури забезпечується контейнеризацією Docker, розміщенням на хмарному сервері Azure, захистом Cloudflare та JWT-авторизацією.

Отже, всі поставлені завдання виконано в повному обсязі. Створено готовий до експлуатації інструмент цивільного захисту, що пришвидшує оцінку тактичної обстановки під час небезпеки. У перспективі заплановано масштабування проекту до формату повноцінного web-сайту.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Кравченко Є.Ю., Замрій І.В. Розробка мікросервісної платформи для оптимізації hr-процесів в умовах автономного розгортання. II Всеукраїнська науково-технічна конференція «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К.: ДУІКТ, 2025, С.482-484.

2. Кравченко Є.Ю. Захист критичної інфраструктури у web-застосунку для візуалізації повітряних загроз. Матеріали Всеукраїнської науково-практичної конференції «Цифрова трансформація кібербезпеки», 26 квітня 2026 року, Київ, ДУІКТ. Збірник тез. *(подано до друку)*.

ПЕРЕЛІК ПОСИЛАНЬ

1. ASP.NET Core SignalR Overview. Microsoft Learn. 21.04.2026. URL: <https://learn.microsoft.com/en-us/aspnet/core/signalr/introduction> (date of access: 26.04.2026).
2. MapLibre GL JS Documentation. MapLibre. URL: <https://maplibre.org/maplibre-gl-js/docs/> (date of access: 26.04.2026).
3. ASP.NET Core Fundamentals and .NET 9 Architecture Guide. Microsoft Learn. 16.12.2025. URL: <https://learn.microsoft.com/en-us/aspnet/core/fundamentals/> (date of access: 26.04.2026).
4. Natural language processing: state of the art, current trends and challenges / D. Khurana та ін. Springer Nature. 2023. С. 3713–3744.
5. Tunstall L., Von Werra L., Wolf T. Natural Language Processing with Transformers. Sebastopol: O'Reilly Media, Inc., 2022. 408 p.
6. Берко А. Ю., Висоцька В. А., Пасічник В. В. Системи опрацювання текстової інформації: навч. посіб. Львів: Магнолія-2006, 2014. 380 с.
7. Субботін С. О. Побудова інтелектуальних систем: навч. посіб. Запоріжжя: Національний університет «Запорізька політехніка», 2020. 163 с.
8. deck.gl: WebGL-powered framework for visual exploratory data analysis. URL: <https://deck.gl/docs> (date of access: 04.05.2026).
9. Docker Documentation. URL: <https://docs.docker.com/manuals/> (date of access: 04.05.2026).
10. Azure Virtual Machines documentation. Microsoft Azure. URL: <https://learn.microsoft.com/en-us/azure/virtual-machines/> (date of access: 04.05.2026).
11. RFC 7519: JSON Web Token (JWT) / Internet Engineering Task Force (IETF). URL: <https://datatracker.ietf.org/doc/html/rfc7519> (date of access: 04.05.2026).
12. Yujian L., Bo L. A Normalized Levenshtein Distance Metric. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2007. Vol. 29, no. 6. P. 1091–1095. URL: <https://doi.org/10.1109/TPAMI.2007.1078> (date of access: 04.05.2026).

13. Hunspell: Spell checker and morphological analyzer library. URL: <https://hunspell.github.io/> (date of access: 04.05.2026).
14. Google Gemini API Documentation. URL: <https://ai.google.dev/docs> (date of access: 04.05.2026).
15. Cloudflare DDoS Protection and Mitigation. URL: <https://www.cloudflare.com/ddos/> (date of access: 04.05.2026).
16. Telegram Mini Apps Documentation. Telegram Core. URL: <https://core.telegram.org/bots/webapps> (date of access: 04.05.2026).
17. ASP.NET Core Blazor. Microsoft Learn. URL: <https://learn.microsoft.com/en-us/aspnet/core/blazor/> (date of access: 04.05.2026).
18. RFC 9136: SSH File Transfer Protocol (SFTP) / Internet Engineering Task Force (IETF). URL: <https://datatracker.ietf.org/doc/html/rfc9136> (date of access: 04.05.2026).
19. ECMA-404: The JSON Data Interchange Syntax. URL: <https://ecma-international.org/publications-and-standards/standards/ecma-404/> (date of access: 04.05.2026).
20. Сервіс моніторингу повітряних загроз. NEPTUN. URL: <https://neptun.in.ua/> (дата звернення: 26.04.2026).
21. Сервіс моніторингу повітряних загроз. Dimap.Live. URL: <https://dimap.live/> (дата звернення: 26.04.2026).
22. Сервіс моніторингу повітряних загроз. Varta.Ai. URL: <https://vartaai.com.ua/> (дата звернення: 26.04.2026).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ



ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



Інтелектуальна система візуалізації текстової інформації про повітряні цілі під час повітряної тривоги

Виконав студент 4 курсу
групи ПД-44

Кравченко Євгеній Юрійович
Керівник роботи

д-р техн. наук, проф Замрій Ірина Вікторівна

Київ – 2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – пришвидшення обробки текстової інформації та інформування користувачів шляхом розробки системи інтелектуальної трансформації неструктурованих текстових даних у динамічні картографічні об'єкти щодо переміщення повітряних загроз у реальному часі.

Об'єкт дослідження – процес автоматизованої обробки та візуалізації неструктурованих даних про переміщення повітряних загроз у реальному часі.

Предмет дослідження – методи та алгоритми функціонування інтелектуальної системи із застосуванням гібридного парсингу та потокової передачі даних.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Провести огляд та аналіз існуючих методів моніторингу і систем візуалізації повітряних загроз із відкритих джерел.
2. Обґрунтувати вибір технологічних засобів, фреймворків та стеку розробки системи.
3. Спроектувати архітектуру системи, алгоритми конвеєра обробки повідомлень, геонормалізації та трекінгу цілей.
4. Виконати програмну реалізацію, тестування системи та демонстрацію результатів роботи.

3

АНАЛІЗ АНАЛОГІВ

Критерій порівняння	Neptun	Dimap	Varta	Система «КОНТУР»
Обробка даних	Пряме використання ШІ	Поступова обробка ШІ	Пряме використання ШІ	Шаблонна токенизація з самонавчанням
Зручність інтерфейсу (UI/UX)	Середня	Низька	Середня	Висока
Протокол синхронізації даних	HTTP Polling	HTTP Polling	HTTP Polling	SignalR (realtime зв'язок)
Платформа розгортання	Веб-сайт та мобільний застосунок	Веб-сайт	Веб-сайт	Telegram Mini App
Рух цілей	Телепортація випадкова зміна курсу	Телепортація, часта випадкова зміна курсу	Відсутній	Плавний рух цілей, без випадкових змін курсу

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

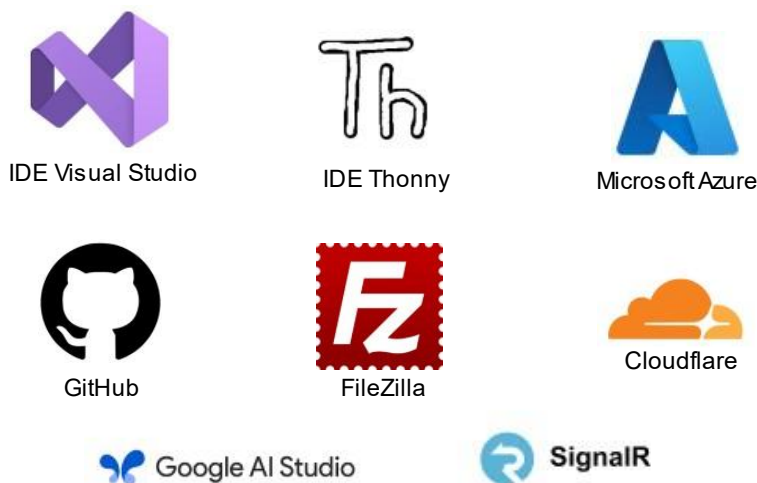
1. Автоматичний збір даних: безперервний моніторинг текстових джерел у реальному часі.
2. Гібридний парсинг повідомлень: розпізнавання параметрів цілей за допомогою шаблонізації та токенизації (тип цілі, локація, вектор).
3. Динамічна картографічна візуалізація: відображення маркерів об'єктів та прогнозованих траскторій на інтерактивній мапі.
4. Подієве керування об'єктами: автоматичний розрахунок часу прибуття (ETA) та видалення застарілих маркерів без втручання оператора.
5. Клієнтська інтерполяція: забезпечення плавності руху міток безпосередньо на стороні клієнта

Нефункціональні вимоги:

1. Ефективність: стабільна робота серверної частини на ресурсообмеженому обладнанні (1гб ОЗУ, CPU – 2 ядра)
2. Швидкість оновлення даних: забезпечення зв'язку через SignalR для оновлення даних на боці користувача протягом 0.1 -3 секунд (в залежності від навантаження на сервер)
3. Відмовостійкість та безпека наявність механізмів Rate Limiting для захисту від перевантажень при масових перепідключеннях .
4. Простота інтерфейсу: мінімізувати кількість елементів на екрані користувача (до 10 елементів на головному екрані)

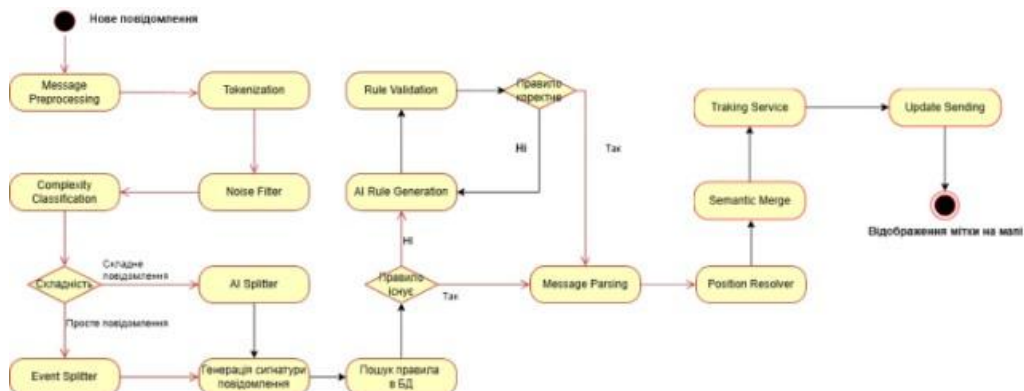
5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



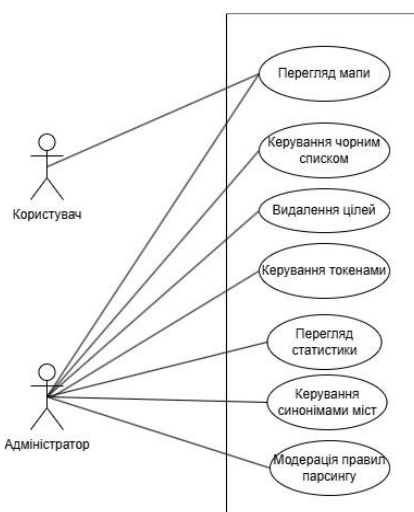
6

ДІАГРАМА ПОСЛІДОВНОСТІ ОБРОБКИ ПОВІДОМЛЕННЯ



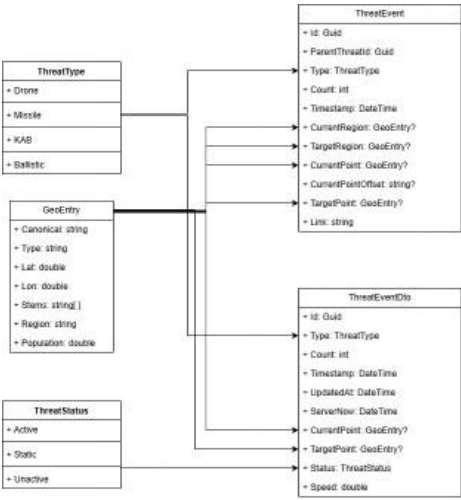
7

ДІАГРАМА ПРЕЦЕДЕНТІВ



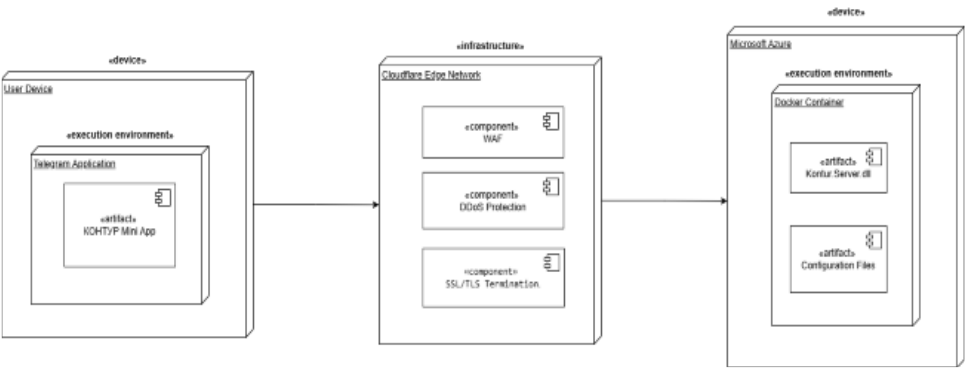
8

ДІАГРАМА ЗВ'ЯЗКІВ ОСНОВНИХ МОДЕЛЕЙ ДАНИХ



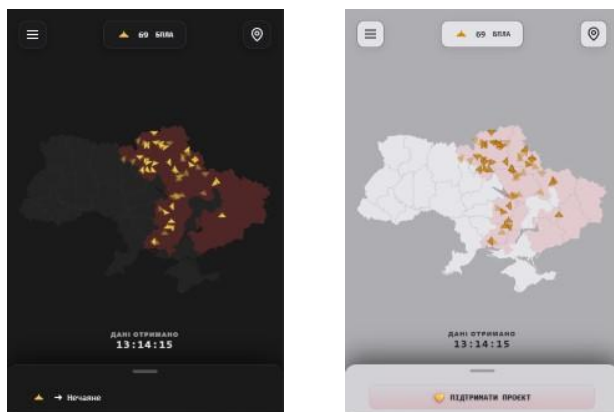
9

ДІАГРАМА РОЗГОРТАННЯ



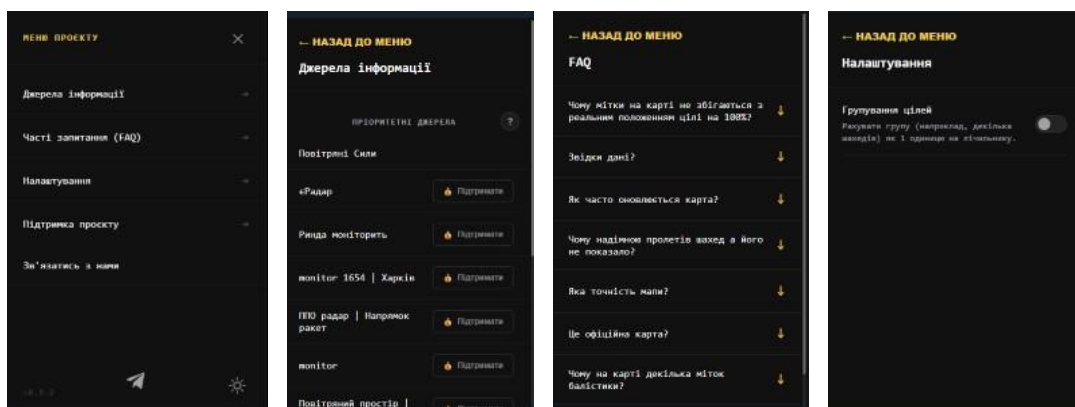
10

ЕКРАННІ ФОРМИ



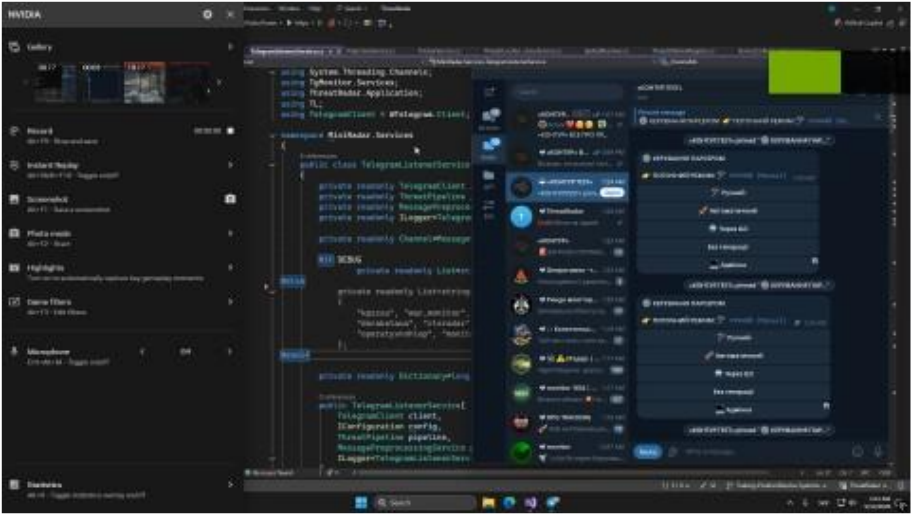
11

ЕКРАННІ ФОРМИ



12

ДЕМОНСТРАЦІЯ РОБОТИ



15

Зведені дані порівняння систем моніторингу інформації про повітряні цілі

Критерій оцінювання	«Neptun»	«Dimar»	«VARTA AI»	«КОНТУР»
Затримка синхронізації даних	~ 30 с	15 – 25 с	~ 5 с	0.1 – 3 с
Кількість елементів інтерфейсу	12	~20	~16	6
Кількість типів цілей	2 типи	2 типи	Без класифікації	4 типи
Візуалізація руху	Телепортація	Телепортація	Відсутній	Плавна інтерполяція
Середовище виконання	Web-браузер	Web-браузер	Web-браузер	Telegram Mini App

16

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Кравченко Є.Ю., Замрій І.В. Розробка мікросервісної платформи для оптимізації ІТ-процесів в умовах автономного розгортання. II Всеукраїнська науково-технічна конференція «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К.: ДУІКТ, 2025, С.482-484.
2. Кравченко Є.Ю. Захист критичної інфраструктури у web-застосунку для візуалізації повітряних загроз. Матеріали Всеукраїнської науково-практичної конференції «Цифрова трансформація кібербезпеки», 26 квітня 2026 року, Київ, ДУІКТ. Збірник тез. *(подано до друку)*.

17

ВИСНОВОК

1. Проведено аналіз існуючих методів обробки неструктурованих текстових даних
2. Проведено аналіз існуючих систем візуалізації повітряних загроз, завдяки чому визначено ключові проблеми.
3. Сформовано вимоги до системи та спроектовано багаторівневий конвеєр (Pipeline) із детермінованим парсингом.
4. Проаналізовано засоби розробки та обґрунтовано вибір .NET SignalR, MapLibreGL JS і відмову від реляційних СУБД на користь In-Методу кешування.
5. Розроблено клієнт-серверний застосунок «Контур» (Telegram Mini App) з плавним відображенням об'єктів як готовий інструмент цивільного захисту.
6. Проведено тестування (End-to-End), яке підтвердило здатність системи коректно опрацьовувати складні вхідні дані та забезпечувати безперервне існування об'єктів у штатному режимі

18

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

```

public class SemanticTokenizer
{
    private readonly ISemanticTokenService
    _semanticTokenService;

    public SemanticTokenizer(ISemanticTokenService
    semanticTokenService)
    {
        _semanticTokenService = semanticTokenService;
    }

    public string Tokenize(string text)
    {
        if (string.IsNullOrEmpty(text)) return text;

        var replacements = new List<(int Index, int Length, string
    Text)>();
        var currentPhraseTokens =
    _semanticTokenService.GetCurrentTokens().PhraseTokens;

        foreach (var token in currentPhraseTokens)
        {
            var patternPhrase = token.Pattern;
            var type = token.TokenType;
            var canonical = token.CanonicalValue;
            var parts = Regex.Split(patternPhrase, @"\s+")
                .Where(p => !string.IsNullOrEmpty(p))
                .Select(Regex.Escape);

            string regexBody = string.Join(@"\s+", parts);
            string regexPattern;

            if (Regex.IsMatch(patternPhrase, @"^[^\w\s]+$"))
            {
                regexPattern = $@"(?:<!\w){regexBody}(?!\\w)";
            }
            else
            {
                regexPattern = regexBody;
            }

            var matches = Regex.Matches(text, regexPattern,
    RegexOptions.IgnoreCase);

            foreach (Match match in matches)
            {
                string tokenString;
                if (IsValueToken(type))
                {
                    tokenString = $" [{type}]={match.Value} ";
                }
                else
                {
                    tokenString = $" [{type}] ";
                }

                replacements.Add((match.Index, match.Length,
    tokenString));
            }

            var wordMatches = Regex.Matches(text, @"\w+[^^\w\s]");
            var currentConfig =
    _semanticTokenService.GetCurrentTokens().Config;

            foreach (Match match in wordMatches)
            {
                if (IsOverlapping(match.Index, match.Length,
    replacements))
                    continue;

                string wordPart = match.Value;
                var tokenMatch = FindTokenForWord(wordPart);

                if (tokenMatch.HasValue)
                {
                    var (type, canonical) = tokenMatch.Value;
                    string tokenString;
                    if (IsValueToken(type))
                    {
                        tokenString = $" [{type}]={wordPart} ";
                    }
                    else
                    {
                        tokenString = $" [{type}] ";
                    }
                    replacements.Add((match.Index, match.Length,
    tokenString));
                    continue;
                }

                var lowerAtom = wordPart.Trim().ToLowerInvariant();
                var configMatch = currentConfig.TokenDefinitions
                    .FirstOrDefault(d => d.Keywords != null &&
                        d.Keywords.Any(k =>
    k.Trim().ToLowerInvariant() == lowerAtom));

                if (configMatch != null)
                {
                    replacements.Add((match.Index, match.Length, $"
    [{configMatch.TokenType}] "));
                }
            }

            var sortedReplacements = replacements
                .OrderBy(x => x.Index)
                .ThenByDescending(x => x.Length)
                .ToList();

            var cleanReplacements = new List<(int Index, int Length,
    string Text)>();
            int lastCoveredIndex = -1;

            foreach (var rep in sortedReplacements)
            {
                if (rep.Index >= lastCoveredIndex)
                {
                    cleanReplacements.Add(rep);
                    lastCoveredIndex = rep.Index + rep.Length;
                }
            }

            var sb = new StringBuilder();
            int lastPos = 0;

            foreach (var rep in cleanReplacements)
            {
                sb.Append(text.Substring(lastPos, rep.Index - lastPos));

```

```

        sb.Append(rep.Text);
        lastPos = rep.Index + rep.Length;
    }

    if (lastPos < text.Length)
    {
        sb.Append(text.Substring(lastPos));
    }

    return sb.ToString();
}

public string ValidateLogicTokens(string text)
{
    return Regex.Replace(text, @"(\[AND\])", match =>
    {
        int index = match.Index;

        string textBefore = text.Substring(0, index).TrimEnd();
        bool leftIsCity = HasTokenAtEnd(textBefore, "CITY");

        string textAfter = text.Substring(index +
match.Length).TrimStart();
        bool rightIsCity = HasTokenAtStart(textAfter,
"CITY");

        if (leftIsCity && rightIsCity)
        {
            return match.Value;
        }
        else
        {
            return "";
        }
    });
}

private bool HasTokenAtEnd(string text, string
tokenTypePrefix)
{
    var match = Regex.Match(text, @"^\[([A-
Z_]+)(?=[^\]]*)?\]$");
    return match.Success &&
match.Groups[1].Value.StartsWith(tokenTypePrefix);
}

private bool HasTokenAtStart(string text, string
tokenTypePrefix)
{
    var match = Regex.Match(text, @"^\[([A-
Z_]+)(?=[^\]]*)?\]$");
    return match.Success &&
match.Groups[1].Value.StartsWith(tokenTypePrefix);
}

private bool IsOverlapping(int index, int length, List<(int
Index, int Length, string Text)> replacements)
{
    int end = index + length;
    foreach (var r in replacements)
    {
        int rEnd = r.Index + r.Length;
        if (index < rEnd && end > r.Index) return true;
    }
    return false;
}

private bool IsValueToken(string type)
{

```

```

        return type == "THREAT" || type == "COUNT";
    }

    private (string Token, string Canonical)?
FindTokenForWord(string inputWord)
    {
        var currentWordTokens =
_semanticTokenService.GetCurrentTokens().WordTokens;

        var exactMatch = currentWordTokens.FirstOrDefault(x
=>
            string.Equals(x.Pattern, inputWord,
StringComparison.OrdinalIgnoreCase));

        if (exactMatch != null)
        {
            return (exactMatch.TokenType,
exactMatch.CanonicalValue);
        }

        if (inputWord.Length <= 3) return null;

        foreach (var token in currentWordTokens)
        {
            var keyword = token.Pattern;
            var tokenType = token.TokenType;
            var canonical = token.CanonicalValue;

            if (keyword.Length <= 3) continue;

            int maxEdits = (keyword.Length > 8) ? 2 :
(keyword.Length > 5 ? 1 : 0);
            if (Math.Abs(keyword.Length - inputWord.Length) >
maxEdits) continue;

            if
(LevenshteinDistance(inputWord.ToLowerInvariant(),
keyword.ToLowerInvariant()) <= maxEdits)
            {
                return (tokenType, canonical);
            }
        }
        return null;
    }

    private int LevenshteinDistance(string s, string t)
    {
        if (string.IsNullOrEmpty(s)) return
string.IsNullOrEmpty(t) ? 0 : t.Length;
        if (string.IsNullOrEmpty(t)) return s.Length;
        int n = s.Length;
        int m = t.Length;
        int[,] d = new int[n + 1, m + 1];
        for (int i = 0; i <= n; d[i, 0] = i++; )
        for (int j = 0; j <= m; d[0, j] = j++; )
        for (int i = 1; i <= n; i++)
        {
            for (int j = 1; j <= m; j++)
            {
                int cost = (t[j - 1] == s[i - 1]) ? 0 : 1;
                d[i, j] = Math.Min(Math.Min(d[i - 1, j] + 1, d[i, j - 1]
+ 1), d[i - 1, j - 1] + cost);
            }
        }
        return d[n, m];
    }
}

```

```

public class PatternParser
{
    private readonly IGeoNormalizer _geoNormalizer;
    private readonly IParsingRuleService _parsingRuleService;
    private readonly TokenProcessor _tokenProcessor;

    public PatternParser(IGeoNormalizer geoNormalizer,
        LocationRepository locationRepository,
        IParsingRuleService parsingRuleService,
        TokenProcessor tokenProcessor)
    {
        _geoNormalizer = geoNormalizer;
        _parsingRuleService = parsingRuleService;
        _tokenProcessor = tokenProcessor;
    }

    public async Task<List<ThreatEvent>> Parse(string
        tokenizedText, List<GeoEntry>? geoEntries = null)
    {
        var tokens =
            _tokenProcessor.ExtractTokens(tokenizedText);
        string signature = new string(tokens.Select(t =>
            t.SignatureChar).ToArray());

        foreach (var rule in _parsingRuleService.GetRules())
        {
            if (signature == rule.SignatureRegex)
            {
                Console.WriteLine($"Matched Rule: {rule.Id}");
                return MapData(tokens, rule.Mapping, 0,
                    tokenizedText, geoEntries);
            }
        }
        return new List<ThreatEvent>();
    }

    private List<ThreatEvent> MapData(List<ParsedToken>
        tokens, Dictionary<string, List<int>> mapping, int matchIndex,
        string fullText, List<GeoEntry>? preResolvedEntries)
    {
        var results = new List<ThreatEvent>();

        string? GetTokenValue(int offset)
        {
            int index = matchIndex + offset;
            return (index >= 0 && index < tokens.Count) ?
                tokens[index].Value : null;
        }

        string? rawType = mapping.ContainsKey("Type") &&
            mapping["Type"].Count > 0 ?
            GetTokenValue(mapping["Type"][0]) : null;
        string? rawCount = mapping.ContainsKey("Count") &&
            mapping["Count"].Count > 0 ?
            GetTokenValue(mapping["Count"][0]) : null;
        string? rawCurrentRegion =
            mapping.ContainsKey("CurrentRegion") &&
            mapping["CurrentRegion"].Count > 0 ?
            GetTokenValue(mapping["CurrentRegion"][0]) : null;

        var threatType = rawType != null ?
            ParseThreatType(rawType) : ThreatType.Unknown;
        var count = rawCount != null ? ParseCount(rawCount) : 1;

        GeoEntry? currentRegion = null;
        GeoEntry? targetRegion = null;

```

```

        string? currentRegionNameForFilter = null;

        if (rawCurrentRegion != null)
        {
            currentRegion = ParseGeoEntry(rawCurrentRegion,
                null, preResolvedEntries);
            currentRegionNameForFilter =
                currentRegion?.Canonical;
        }

        var vectorIndices = mapping.ContainsKey("TargetPoint")
            ? mapping["TargetPoint"] : new List<int>();
        var locationIndices =
            mapping.ContainsKey("CurrentPoint") ?
            mapping["CurrentPoint"] : new List<int>();
        if (locationIndices.Count == 0 &&
            mapping.ContainsKey("Origin")) locationIndices =
            mapping["Origin"];

        var locOffsetIndices =
            mapping.ContainsKey("CurrentPointOffset") ?
            mapping["CurrentPointOffset"] : new List<int>();
        var vecOffsetIndices =
            mapping.ContainsKey("TargetPointOffset") ?
            mapping["TargetPointOffset"] : new List<int>();

        var absVectorIndices = vectorIndices.Select(idx =>
            matchIndex + idx).Where(idx => idx < tokens.Count).ToList();
        var absLocationIndices = locationIndices.Select(idx =>
            matchIndex + idx).Where(idx => idx < tokens.Count).ToList();

        var groupedVectors =
            GroupConnectedIndices(absVectorIndices, tokens, fullText);
        var groupedLocations =
            GroupConnectedIndices(absLocationIndices, tokens, fullText);

        int iterations = Math.Max(1,
            Math.Max(groupedVectors.Count, groupedLocations.Count));

        for (int i = 0; i < iterations; i++)
        {
            var vecGroup = groupedVectors.Count > 0 ?
                groupedVectors[Math.Min(i, groupedVectors.Count - 1)] : new
                List<int>();
            var locGroup = groupedLocations.Count > 0 ?
                groupedLocations[Math.Min(i, groupedLocations.Count - 1)] :
                new List<int>();

            int vecOffRelIdx = vecOffsetIndices.Count > 0 ?
                vecOffsetIndices[Math.Min(i, vecOffsetIndices.Count - 1)] : -
                1;
            int locOffRelIdx = locOffsetIndices.Count > 0 ?
                locOffsetIndices[Math.Min(i, locOffsetIndices.Count - 1)] : -1;

            string? vecOffsetVal = vecOffRelIdx != -1 ?
                GetTokenValue(vecOffRelIdx) : null;
            string? locOffsetVal = locOffRelIdx != -1 ?
                GetTokenValue(locOffRelIdx) : null;

            GeoEntry? currentLocation =
                ResolveGeoLocation(locGroup, tokens,
                    currentRegionNameForFilter, preResolvedEntries);

            if (currentLocation != null && currentLocation.Type
                == "region")
            {
                currentRegionNameForFilter =
                    currentLocation.Canonical;
            }

```

```

        GeoEntry? movementVector =
        ResolveGeoLocation(vecGroup, tokens,
        currentRegionNameForFilter, preResolvedEntries);

        currentRegion ??=
        ParseGeoEntry(currentLocation?.Region, null,
        preResolvedEntries);
        targetRegion ??=
        ParseGeoEntry(movementVector?.Region, null,
        preResolvedEntries);

        var threat = new ThreatEvent
        {
            Id = Guid.NewGuid(),
            Type = threatType,
            Count = count,
            CurrentRegion = currentRegion,
            TargetRegion = targetRegion,
            CurrentPoint = currentLocation,
            TargetPoint = movementVector,
            CurrentPointOffset = locOffsetVal,
            Timestamp = DateTime.UtcNow
        };

        results.Add(threat);
    }

    return results;
}

private GeoEntry? ResolveGeoLocation(List<int>
groupIndices, List<ParsedToken> tokens, string?
contextRegion, List<GeoEntry?> preResolvedEntries)
{
    if (groupIndices == null || groupIndices.Count == 0) return
    null;

    if (groupIndices.Count == 1)
    {
        var val = tokens[groupIndices[0]].Value;
        return ParseGeoEntry(val, contextRegion,
        preResolvedEntries);
    }

    var resolvedPoints = new List<GeoEntry>();
    var names = new List<string>();

    foreach (var idx in groupIndices)
    {
        var val = tokens[idx].Value;
        var geo = ParseGeoEntry(val, contextRegion,
        preResolvedEntries);

        if (geo != null && geo.Lat != 0 && geo.Lon != 0)
        {
            resolvedPoints.Add(geo);
            names.Add(geo.Canonical);
        }
    }

    if (resolvedPoints.Count > 1)
    {
        double avgLat = resolvedPoints.Average(p => p.Lat);
        double avgLon = resolvedPoints.Average(p => p.Lon);

        return new GeoEntry
        {
            Canonical = string.Join(" / ", names),

```

```

            Lat = avgLat,
            Lon = avgLon,
            Type = "RegionsBorder",
            Region = resolvedPoints.First().Region
        };
    }

    string combinedName = string.Join(" ",
    groupIndices.Select(idx => tokens[idx].Value));
    var combinedResult = ParseGeoEntry(combinedName,
    contextRegion, preResolvedEntries);

    return combinedResult ?? (resolvedPoints.Count > 0 ?
    resolvedPoints.First() : null);
}

private List<List<int>> GroupConnectedIndices(List<int>
absIndices, List<ParsedToken> tokens, string fullText)
{
    if (absIndices == null || absIndices.Count == 0) return new
    List<List<int>>();

    absIndices.Sort();
    var groups = new List<List<int>>();
    var currentGroup = new List<int> { absIndices[0] };

    for (int i = 1; i < absIndices.Count; i++)
    {
        int prevIdx = absIndices[i - 1];
        int currIdx = absIndices[i];

        if (IsSemanticallyConnected(prevIdx, currIdx, tokens,
        fullText))
        {
            currentGroup.Add(currIdx);
        }
        else
        {
            groups.Add(currentGroup);
            currentGroup = new List<int> { currIdx };
        }
    }
    groups.Add(currentGroup);
    return groups;
}

private bool IsSemanticallyConnected(int startIdx, int
endIdx, List<ParsedToken> tokens, string fullText)
{
    if (startIdx >= tokens.Count - 1) return false;

    bool hasAreaQualifierNearby = false;
    for (int j = Math.Max(0, startIdx - 3); j <=
    Math.Min(tokens.Count - 1, endIdx + 3); j++)
    {
        if (tokens[j].Type == "AREA_QUALIFIER")
        {
            hasAreaQualifierNearby = true;
            break;
        }
    }

    for (int i = startIdx + 1; i < endIdx; i++)
    {
        var type = tokens[i].Type;
        if (type == "SEPARATOR") return false;
        if (type == "SLASH") return true;
        if (type == "THREAT") return false;
    }
}

```

```

        var val = tokens[i].Value.ToLower();
        if (val == "i" || val == "ra" || val == "або" || val ==
"and" || val == ",")
        {
            return hasAreaQualifierNearby;
        }
    }

    var prevToken = tokens[startIdx];
    var nextToken = tokens[endIdx];
    int gapStart = prevToken.Index + prevToken.Length;
    int gapEnd = nextToken.Index;

    if (gapEnd > gapStart)
    {
        string gapText = fullText.Substring(gapStart, gapEnd -
gapStart);
        if (gapText.Contains(",") || gapText.Contains(";"))
            return hasAreaQualifierNearby;
    }

    return true;
}

/// <summary>
/// Основний метод пошуку. Спочатку шукає в
preResolvedEntries, якщо не знаходить — йде в
нормалізатор.
/// </summary>
private GeoEntry? ParseGeoEntry(string? rawValue, string?
context = null, List<GeoEntry>? preResolvedEntries = null)
{
    if (string.IsNullOrWhiteSpace(rawValue)) return null;

    if (preResolvedEntries != null &&
preResolvedEntries.Count > 0)
    {
        var candidates = preResolvedEntries
            .Where(e => rawValue.StartsWith(e.Canonical,
StringComparison.OrdinalIgnoreCase))
            .ToList();

        if (candidates.Count > 0)
        {
            if (!string.IsNullOrEmpty(context))
            {
                var regionMatch = candidates.FirstOrDefault(c =>
!string.IsNullOrEmpty(c.Region) &&
c.Region.Contains(context,
StringComparison.OrdinalIgnoreCase));

                if (regionMatch != null)
                {
                    return regionMatch;
                }
            }

            return candidates.First();
        }
    }

    var result = _geoNormalizer.Normalize(rawValue,
context);
    return result;
}

private int ParseCount(string rawValue)
{
    if (string.IsNullOrWhiteSpace(rawValue)) return 0;
    if (rawValue.Contains("група",
StringComparison.OrdinalIgnoreCase) ||
rawValue.Contains("няски",
StringComparison.OrdinalIgnoreCase)) return 0;

    var digitsOnly = Regex.Replace(rawValue, @"\\D", "");
    return int.TryParse(digitsOnly, out int result) ? result : 0;
}

private ThreatType ParseThreatType(string rawValue)
{
    var val = rawValue.ToUpperInvariant();
    if (val.Contains("БІЛІА") || val.Contains("ІІІАХЕД") ||
val.Contains("DRONE") || val.Contains("UAV")) return
ThreatType.Drone;
    if (val.Contains("БАЛІСТ") ||
val.Contains("ІКАНДЕР") || val.Contains("BALLISTIC"))
return ThreatType.Ballistic;
    if (val.Contains("ПАКЕТ") || val.Contains("КР") ||
val.Contains("MISSILE")) return ThreatType.Missile;
    if (val.Contains("КАБ") || val.Contains("KAB")) return
ThreatType.KAB;
    if (val.Contains("АБІАІ") || val.Contains("БОПТ") ||
val.Contains("ТІ-95")) return ThreatType.Aviation;
    return ThreatType.Unknown;
}

public class TrackerService
{
    private readonly ThreatRepository _repository;
    private readonly ThreatPositionResolver _positionResolver;
    private readonly TrajectoryService _trajectoryService;
    private readonly ThreatMergeService _threatMergeService;
    private readonly ILogger<TrackerService> _logger;

    public TrackerService(ThreatRepository repository,
ThreatPositionResolver positionResolver,
TrajectoryService trajectoryService,
ThreatMergeService threatMergeService,
ILogger<TrackerService> logger)
    {
        _repository = repository;
        _positionResolver = positionResolver;
        _trajectoryService = trajectoryService;
        _threatMergeService = threatMergeService;
        _logger = logger;
    }

    public List<ThreatEvent>
ProcessIncomingBatch(List<ThreatEvent> threatsBatch,
ThreatEvent? parentThreat = null)
    {
        if (threatsBatch == null || threatsBatch.Count == 0) return
new List<ThreatEvent>();

        var activeThreatsRaw =
_repository.GetAllActiveThreats();
        var activeDtos =
_repository.GetAllActiveThreatsDTO().ToDictionary(d =>
d.Id);

        var allTrackingThreatsRaw = activeDtos.Keys
            .Select(id => _repository.GetById(id))
            .Where(t => t != null)
            .ToList();

        var matchedIds = new HashSet<Guid>();

```

```

var proceseedThreats = new List<ThreatEvent>();
var threatsToSave = new List<(ThreatEvent Raw,
ThreatEventDTO Dto)>();

ThreatEventDTO? parentDto = null;

if (parentThreat != null)
{
    parentDto =
    _positionResolver.ResolvePosition(parentThreat,
    activeThreatsRaw);
    var incomingThreat = threatsBatch.FirstOrDefault(t =>
    t.Type == parentThreat.Type);

    if (incomingThreat != null && parentDto != null)
    {
        var newCurrentPoint = new GeoEntry
        {
            Lat = parentDto.AnchorGeo.Lat,
            Lon = parentDto.AnchorGeo.Lon,
            Canonical = parentDto.AnchorGeo.Canonical,
            Type = GeoEntryTypes.Updated
        };

        var newThreat = new ThreatEvent
        {
            Id = parentThreat.Id,
            ParentThreatId = incomingThreat.ParentThreatId,
            Type = parentThreat.Type,
            Count = Math.Max(incomingThreat.Count,
parentThreat.Count),
            Timestamp = incomingThreat.Timestamp,
            CurrentPoint = newCurrentPoint,
            TargetPoint = incomingThreat.TargetPoint,
            Link = parentThreat.Link
        };

        var updatetTargetDto =
        _positionResolver.ResolvePosition(newThreat,
        activeThreatsRaw);

        if (updatetTargetDto != null)
        {
            var updatedDto =
            UpdateExistingThreat(parentThreat, parentDto,
            updatetTargetDto, incomingThreat.Link);

            matchedIds.Add(parentThreat.Id);
            proceseedThreats.Add(parentThreat);
            threatsToSave.Add((parentThreat, updatedDto));
        }
        threatsBatch.Remove(incomingThreat);
    }
}

foreach (var incomingRaw in threatsBatch)
{
    var incomingDto =
    _positionResolver.ResolvePosition(incomingRaw,
    allTrackingThreatsRaw);

    if (incomingDto == null || incomingDto.AnchorGeo ==
    null) continue;

    var distanceToPoint =
    GeoMath.CalculateDistanceKm(incomingDto.AnchorGeo,
    incomingDto.TargetGeo ?? incomingDto.AnchorGeo);
    var targetIsRegion =
    _positionResolver.IsRegion(incomingDto.TargetGeo);

```

```

    var targetHasBadDistance = (targetIsRegion &&
    distanceToPoint > 300) || (!targetIsRegion && distanceToPoint
    > 150);
    var isUnlogicalType = incomingRaw.Type ==
    ThreatType.KAB || incomingRaw.Type ==
    ThreatType.Ballistic;

    if (!isUnlogicalType && targetHasBadDistance &&
    _positionResolver.IsValidGeo(incomingDto.TargetGeo))
    {
        _logger.LogError($"[Tracker] --- Threat
        {incomingDto.Id} has unrealistic distance to target
        ({distanceToPoint:F1} km). Skipping.");
        continue;
    }

    var candidateDtos = activeDtos.Values
    .Where(t => AreTypesCompatible(t.Type,
    incomingRaw.Type) && !matchedIds.Contains(t.Id))
    .ToList();

    bool wasMerged =
    _threatMergeService.TryMergeWithExisting(incomingDto,
    candidateDtos, out var mergedDto);

    if (wasMerged && mergedDto.Id != incomingDto.Id)
    {
        var existingRaw =
        _repository.GetById(mergedDto.Id);
        if (existingRaw != null)
        {
            existingRaw.CurrentPoint =
            mergedDto.AnchorGeo;
            existingRaw.TargetPoint = mergedDto.TargetGeo;
            existingRaw.Timestamp = mergedDto.Timestamp;
            existingRaw.Count = mergedDto.Count;
            if (!string.IsNullOrEmpty(incomingRaw.Link))
            existingRaw.Link = incomingRaw.Link;

            activeDtos[mergedDto.Id] = mergedDto;

            matchedIds.Add(mergedDto.Id);
            proceseedThreats.Add(existingRaw);
            threatsToSave.Add((existingRaw, mergedDto));

            _logger.LogInformation($"[Tracker] Threat
            {incomingDto.Type} seamlessly MERGED with
            {mergedDto.Id} (Abstract Resolution).");

            continue;
        }
    }

    var bestMatchDto =
    FindBestMatchUsingPhysics(incomingDto, candidateDtos);

    if (bestMatchDto != null)
    {
        var bestMatchRaw =
        _repository.GetById(bestMatchDto.Id);

        if (bestMatchRaw != null)
        {
            var updatedDto =
            UpdateExistingThreat(bestMatchRaw, bestMatchDto,
            incomingDto, incomingRaw.Link);

```

```

        activeDtos[updatedDto.Id] = updatedDto;

        matchedIds.Add(bestMatchRaw.Id);
        proceseedThreats.Add(bestMatchRaw);
        threatsToSave.Add((bestMatchRaw, updatedDto));
    }
    else
    {
        CreateNewThreat(incomingRaw, incomingDto);
        proceseedThreats.Add(incomingRaw);
        threatsToSave.Add((incomingRaw,
incomingDto));
    }
    }
    else
    {
        CreateNewThreat(incomingRaw, incomingDto);
        proceseedThreats.Add(incomingRaw);
        threatsToSave.Add((incomingRaw, incomingDto));
    }
    }

    if (threatsToSave.Any())
    {
        _repository.AddOrUpdateRange(threatsToSave);
    }

    return proceseedThreats;
}

private bool AreTypesCompatible(ThreatType type1,
ThreatType type2)
{
    if (type1 == type2) return true;

    bool isMissileGroup = (type1 == ThreatType.Missile ||
type1 == ThreatType.Ballistic) &&
        (type2 == ThreatType.Missile || type2 ==
ThreatType.Ballistic);

    if (isMissileGroup) return true;

    return false;
}

private ThreatEventDTO UpdateExistingThreat(ThreatEvent
existingRaw, ThreatEventDTO matchedDto, ThreatEventDTO
incomingDto, string? link = null)
{
    var currentDynamicPos = matchedDto.AnchorGeo;

    bool isArrived = incomingDto.Status ==
ThreatStatus.Static || incomingDto.Status ==
ThreatStatus.Unactive;

    GeoEntry newAnchor;
    if (isArrived && incomingDto.AnchorGeo != null)
    {
        newAnchor = incomingDto.AnchorGeo with { Type =
GeoEntryTypes.Updated };
    }
    else
    {
        newAnchor = currentDynamicPos with
        {
            Type = GeoEntryTypes.Updated,
            Canonical = incomingDto.AnchorGeo?.Canonical ??
currentDynamicPos.Canonical,

```

```

            Region = incomingDto.AnchorGeo?.Region ??
currentDynamicPos.Region
        };
    }

    matchedDto.AnchorGeo = newAnchor;

    if (incomingDto.TargetGeo != null)
        matchedDto.TargetGeo = incomingDto.TargetGeo;
    else if (incomingDto.AnchorGeo?.Type ==
GeoEntryTypes.Stationary || incomingDto.AnchorGeo?.Type
== GeoEntryTypes.Expired)
        matchedDto.TargetGeo = incomingDto.AnchorGeo;

    matchedDto.Count = Math.Max(matchedDto.Count,
incomingDto.Count);
    matchedDto.Timestamp = DateTime.UtcNow;
    matchedDto.UpdatedAt = DateTime.UtcNow;
    matchedDto.Status = incomingDto.Status !=
ThreatStatus.Active ? incomingDto.Status :
ThreatStatus.Active;

    existingRaw.CurrentPoint = newAnchor;
    existingRaw.TargetPoint = matchedDto.TargetGeo;
    existingRaw.Timestamp = matchedDto.Timestamp;
    if (!string.IsNullOrEmpty(link)) existingRaw.Link = link;
    existingRaw.Count = matchedDto.Count;

    return matchedDto;
}

private void CreateNewThreat(ThreatEvent incomingRaw,
ThreatEventDTO incomingDto)
{
    incomingRaw.CurrentPoint = incomingDto.AnchorGeo;
    incomingRaw.TargetPoint = incomingDto.TargetGeo;
}

private ThreatEventDTO?
FindBestMatchUsingPhysics(ThreatEventDTO incomingDto,
List<ThreatEventDTO> candidates)
{
    ThreatEventDTO? bestCandidate = null;
    double minPenalty = double.MaxValue;
    double MATCH_THRESHOLD = incomingDto.Type
switch
    {
        ThreatType.Drone => 60, // БПЛА: залишаємо
старий великий радіус (макс. допуск)
        ThreatType.Missile => 40, // Ракети: трохи менший
(середня точність)
        ThreatType.Ballistic => 10.0, // Балістика:
найменший (траєкторії дуже жорсткі)
        ThreatType.KAB => 20, // КАБ: специфічний
допуск
        _ => 50 // Default для невідомих типів
    };

    var profile =
TrackingPhysics.GetProfile(incomingDto.Type);
    var incomingGeo = incomingDto.AnchorGeo!;
    var effectiveIncomingTarget = incomingDto.TargetGeo ??
incomingDto.AnchorGeo;

    bool isStaticArrival = incomingDto.AnchorGeo?.Type ==
GeoEntryTypes.Stationary;

```

```

_logger.LogInformation($"[TRACKER DEBUG]
Incoming {incomingDto.Type} at {incomingGeo.Canonical ??
"Unknown"}. Checking {candidates.Count} candidates...");

```

```

foreach (var candidateDto in candidates)
{
    double penalty = 0.0;
    double timeDiffHours = (incomingDto.Timestamp -
candidateDto.Timestamp).TotalHours;

    if (timeDiffHours < -0.005) continue;
    if (timeDiffHours > 3.0)
    {
        _logger.LogInformation($"[TRACKER DEBUG]
Candidate {candidateDto.Id} rejected: Too old
({timeDiffHours:F2} hours)");
        continue;
    }

    double safeTimeDiff = Math.Max(timeDiffHours,
0.003);
    var candidateGeo = candidateDto.AnchorGeo;

    if (candidateGeo == null) continue;

    bool isCandidateLoitering = candidateDto.Status ==
ThreatStatus.Static ||
                                candidateDto.Status ==
ThreatStatus.Unactive;

    bool isIncomingOffset = incomingGeo.Type ==
GeoEntryTypes.Offset;
    double distKm =
GeoMath.CalculateDistanceKm(candidateGeo.Lat,
candidateGeo.Lon, incomingGeo.Lat, incomingGeo.Lon);
    double effectiveDistForSpeed = distKm;

    if (isCandidateLoitering || isIncomingOffset)
    {
        effectiveDistForSpeed = Math.Max(0, distKm -
10.0);
    }

    double requiredSpeed = effectiveDistForSpeed /
safeTimeDiff;

    if (requiredSpeed > profile.MaxSpeedKmh * 1.5)
    {
        _logger.LogInformation($"[TRACKER DEBUG]
Candidate {candidateDto.Id} rejected: Required speed too
high! Dist: {distKm:F1}km (Effective:
{effectiveDistForSpeed:F1}km), TimeDiff: {safeTimeDiff *
60:F1}min, Speed: {requiredSpeed:F0}km/h (Max:
{profile.MaxSpeedKmh * 1.5})");
        continue;
    }

    if (candidateDto.TargetGeo != null &&
effectiveIncomingTarget != null)
    {
        string oldTarget =
candidateDto.TargetGeo.Canonical ?? "";
        string newTarget =
effectiveIncomingTarget.Canonical ?? "";
        bool isRefinement = oldTarget.Contains(newTarget,
StringComparison.OrdinalIgnoreCase) ||
newTarget.Contains(oldTarget,
StringComparison.OrdinalIgnoreCase);

```

```

if (isRefinement)
{
    if (!isCandidateLoitering)
    {
        penalty -= 25.0;
        _logger.LogInformation($"[TRACKER
DEBUG] Candidate {candidateDto.Id} got -25 BONUS for
matching target ({newTarget})");
    }
    else
    {
        if (!isStaticArrival && !isCandidateLoitering)
        {
            penalty += 70.0;
            if (incomingDto.Type == ThreatType.Missile)
                penalty += 20.0;
            _logger.LogInformation($"[TRACKER
DEBUG] Candidate {candidateDto.Id} got +70 penalty for
target change ({oldTarget} -> {newTarget})");
        }
        else if (isCandidateLoitering)
        {
            penalty += 15.0;
            _logger.LogInformation($"[TRACKER
DEBUG] Candidate {candidateDto.Id} got +15 penalty for
static re-activation to new target ({newTarget})");
        }
    }

    if (candidateDto.TargetGeo != null &&
effectiveIncomingTarget != null)
    {
        var currentDynamicPos =
_trajectoryService.CalculateCurrentPosition(candidateDto);
        double? oldBearing =
GetSmartBearing(candidateGeo, candidateDto.TargetGeo);
        double? newBearing =
GetSmartBearing(currentDynamicPos,
effectiveIncomingTarget);

        if (oldBearing.HasValue && newBearing.HasValue)
        {
            double angleDiff = Math.Abs(oldBearing.Value -
newBearing.Value);
            if (angleDiff > 180) angleDiff = 360 - angleDiff;

            double maxAllowedTurn = incomingDto.Type ==
ThreatType.Drone ? 120.0 : 90.0;
            bool ignoreTurnLimit = isStaticArrival ||
isCandidateLoitering;

            if (!ignoreTurnLimit && angleDiff >
maxAllowedTurn)
            {
                _logger.LogInformation($"[TRACKER
DEBUG] Candidate {candidateDto.Id} rejected: Turn angle too
sharp! ({angleDiff:F1} > {maxAllowedTurn})");
                continue;
            }
            if (!ignoreTurnLimit) penalty += angleDiff / 2.0;
        }
    }

    GeoEntry predictedPos;
    if (isCandidateLoitering)
    {
        predictedPos = candidateGeo;

```

```

    }
    else
    {
        predictedPos =
TrackingPhysics.PredictPosition(candidateGeo,
candidateDto.TargetGeo, safeTimeDiff,
profile.CruisingSpeedKmH);
    }

    double predictionErrorKm =
GeoMath.CalculateDistanceKm(predictedPos.Lat,
predictedPos.Lon, incomingGeo.Lat, incomingGeo.Lon);
    penalty += predictionErrorKm * 1.5;

    _logger.LogInformation($"[TRACKER DEBUG]
Candidate {candidateDto.Id} passed. Loitering:
{isCandidateLoitering}, Prediction Error:
{predictionErrorKm:F1}km, Total Penalty: {penalty:F1}");

    if (penalty < minPenalty && penalty <
MATCH_THRESHOLD)
    {
        minPenalty = penalty;
        bestCandidate = candidateDto;
    }
}

if (bestCandidate != null)
{
    _logger.LogInformation($"[TRACKER DEBUG]
MATCH FOUND! Winner ID: {bestCandidate.Id} with
penalty {minPenalty:F1}");
}
else
{
    _logger.LogInformation($"[TRACKER DEBUG] NO
MATCH FOUND. Creating new threat.");
}

return bestCandidate;
}

private double? GetSmartBearing(GeoEntry start, GeoEntry
target)
{
    if (Math.Abs(target.Lat) > 0.1 || Math.Abs(target.Lon) >
0.1)
    {
        return GeoMath.CalculateBearing(start, target);
    }
    return
TrackingPhysics.VectorToDegrees(target.Canonical);
}
}

```