ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО «ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД «МІЖРЕГІОНАЛЬНА АКАДЕМІЯ УПРАВЛІННЯ ПЕРСОНАЛОМ»

Факультет комп'ютерно-інформаційних технологій

Кафедра комп'ютерних інформаційних систем і технологій

Залевський Денис Вікторович

КВАЛІФІКАЦІЙНА РОБОТА

на тему: **«Система моніторингу рухомих об'єктів на основі геоінформаційних даних»**

Група: ІК-9-22-Б1ІПЗ (4.0ддс)

Спеціальність: Інженерія програмного забезпечення

Рівень вищої освіти: перший (бакалаврський)

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

_____ Денис ЗАЛЕВСЬКИЙ

(підпис)

Науковий керівник

_____ Андрій ДУДНІК, д.т.н., професор

(підпис)

Допущено до захисту ЕК

Завідувач кафедри _____ Сергій КАВУН, д.е.н., професор

(підпис)

Київ 2026

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до атестаційної роботи: 61 с., 13 рис., 43 джерела.

СИСТЕМА МОНІТОРИНГУ РУХОМИХ ОБ’ЄКТІВ НА ОСНОВІ ГЕОІНФОРМАЦІЙНИХ ДАНИХ, JAVASCRIPT, REST API, NIFI.

Об’єктом дослідження є процеси моніторингу рухомих об’єктів і обробки геоінформаційних даних, зокрема у диспетчерських системах.

Предметом дослідження є методи візуалізації координат на карті, керування геозонами та інтеграції з потоковою обробкою даних.

Метою роботи є розробка програмного забезпечення для моніторингу рухомих об’єктів на основі геоінформаційних даних.

Основні інструменти розробки безпосередньо використовуються для реалізації: JavaScript, NiFi, REST-подібний HTTP API.

Розроблене програмне забезпечення дозволяє відображати поточні координати об’єктів на карті та у таблиці, підтримує геозони у вигляді полігонів із повним циклом CRUD-операцій (створення, редагування, видалення), формує сповіщення для об’єктів, які довго не надсилають координати, та забезпечує підключення до різних серверів із локальним збереженням налаштувань у браузері.

MOBILE OBJECT MONITORING SYSTEM BASED ON GEO-INFORMATION DATA, JAVASCRIPT, REST API, NIFI.

The object of the research is the processes of mobile objects monitoring and processing of geo-information data, in particular in dispatching systems.

The subject of the research is methods of coordinates visualizing on the map, geo-zones managing, and integrating with streaming data processing.

The goal of the work is to develop software for monitoring mobile objects based on geo-information data.

The main development tools are directly used for implementation: JavaScript, NiFi, REST-like HTTP API.

The developed software allows you to display the current coordinates of objects on the map and in the table, supports geo-zones in the form of polygons with a full cycle of CRUD operations (creation, editing, deletion), generates alarms for objects that do not send coordinates for a long time, and provides connection to various servers with local storage of settings in a browser.

ЗМІСТ

ВСТУП

В умовах сучасної логістики, сервісних служб, енергетики, безпеки та муніципального транспорту моніторинг рухомих об’єктів на карті перестав бути додатковою функцією і став регулярною вимогою операційного управління. Диспетчерські підрозділи потребують актуальної інформації про місцезнаходження об’єктів для зменшення простоїв і затримок, контролю виконання робіт у визначених локаціях, швидкого реагування на інциденти та забезпечення прозорості процесів. Особливої ваги набуває зонування як механізм формалізації подій та оперативного керування об’єктами залежно від їх просторового контексту.

Актуальність теми також підсилюється інтеграційним характером сучасних систем моніторингу. Реальні впровадження майже завжди потребують зв’язку з декількома контурами, наприклад, телематикою, внутрішніми системами підприємства, довідниками локацій, а у спеціалізованих сценаріях, такими як радіозв’язок, необхідно враховувати специфіку обладнання та пропрієтарних протоколів. Тому ключовою вимогою до рішення є не лише візуалізація координат, а й наявність керованого інтеграційного контуру, який здатен приймати потоки даних, обробляти їх, застосовувати правила та виконувати необхідні менеджерські дії.

Метою даної роботи є розробка програмного забезпечення для моніторингу рухомих об’єктів на основі геоінформаційних даних з акцентом на диспетчерський Web UI, який забезпечує відображення координат на карті, керування геозонами та індикацію проблем з актуальністю даних. Об’єктом дослідження є процеси моніторингу рухомих об’єктів і обробки геоінформаційних даних, зокрема у диспетчерських системах. Предметом дослідження є методи візуалізації координат, керування геозонами, організація обміну даними через REST-подібний HTTP API, а також інтеграція з потоковою обробкою даних та керуванням групами радіостанцій у бекенд-контурі.

Розроблюване рішення передбачає розділення відповідальностей між інтерфейсом диспетчера та серверним інтеграційним шаром. Web UI відображає об’єкти, зони, стан актуальності координат та сповіщення, а також дає можливість виконувати операції редагування над зонами та довідниками, що необхідні для роботи системи. Інтеграційний контур забезпечує обробку координат, визначення належності об’єкта до геозони, та при потребі ініціює перемикання групи відповідно до зони знаходження об’єкта. Завдяки цьому диспетчер може працювати з об’єктами в межах своєї геозони і виконувати комунікаційні дії на основі групового контексту.

Для досягнення поставленої мети в роботі необхідно виконати такі основні завдання: проаналізувати предметну область геомоніторингу та типові архітектурні схеми отримання і обробки координат, сформувати функціональні та нефункціональні вимоги до системи, обґрунтувати вибір технологій для реалізації рішення, реалізувати веб-інтерфейс моніторингу та описати інтеграцію з бекендом, виконати перевірку ключових сценаріїв.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Актуальність та роль моніторингу рухомих об’єктів

Геомоніторинг рухомих об’єктів розглядається як сукупність програмно-апаратних засобів, що забезпечують збір координат, їх обробку та візуалізацію на карті з підтримкою правил просторового контролю (геозон). У контексті цієї роботи рухомими об’єктами виступають трекери або радіопристрої, позиції яких надходять у систему у вигляді координат (широта, довгота) з позначенням ідентифікатора об’єкта. Ключовим результатом для оператора є наочна ситуаційна картина на карті, доповнена механізмами контролю актуальності даних та керування геозонами.

У наш час геомоніторинг став базовою функцією операційного управління. У транспорті, логістиці, сервісних службах, муніципалітетах, енергетиці, безпеці та агро секторі відстеження об’єктів в реальному часі переходить із опції в обов’язковий мінімум. Це пов’язано з оптимізацією маршрутів, витрат пального, часу простоїв, контролем SLA (вчасність прибуття, виконання робіт), прозорістю для клієнта (розрахунковим часом подій, відстеження), потребою в ситуаційній обізнаності (оперативні рішення на основі карти). Моніторинг рухомих об’єктів дає змогу диспетчеру швидко приймати рішення щодо маршрутизації, розподілу ресурсів, реагування на інциденти, контролю часу на точках і підтвердження факту виконання логістичних операцій.

У реальних логістичних процесах значна частина втрат ефективності виникає через затримки у прийнятті рішень та відсутність актуальної інформації про місцезнаходження і стан транспорту. Системи моніторингу дозволяють:

- бачити розташування об’єктів на карті в режимі, наближеному до реального часу;
- швидко визначати, який транспорт знаходиться найближче до точки завантаження або клієнта;

– зменшувати кількість ручних комунікацій з уточнення місця знаходження та прискорювати диспетчерські рішення.

Сучасний бізнес очікує операційну видимість у реальному часі. У логістиці висока вартість затримок, простоїв і помилок планування. Без оперативної карти диспетчер часто працює «всліпу» і змушений витрачати час на уточнення. Системи управління автопарком і телематика розвиваються як «платформи видимості»: відслідковування в реальному часі, диспетчеризація, інтеграції, аналітика. Наприклад, Samsara прямо позиціонує реальний трекінг, диспетчеризацію, оптимізацію маршрутів і аналітику як основу підвищення ефективності флоту. [1] Grand View Research описує драйвери росту саме через інтеграцію IoT, телематики та аналітики для флотів. [2] Зростання цього ринку прогнозується з 7 млрд. долларів США у 2023 році до 20,6 млрд. долларів США у 2030 році (рис.1.1.1). Аналогічно Geotab підкреслює функціональність відслідковування в майже реальному часі, історію поїздок, роботу з зонами, маршрутами та інструменти управління ефективністю автопарку. [3]

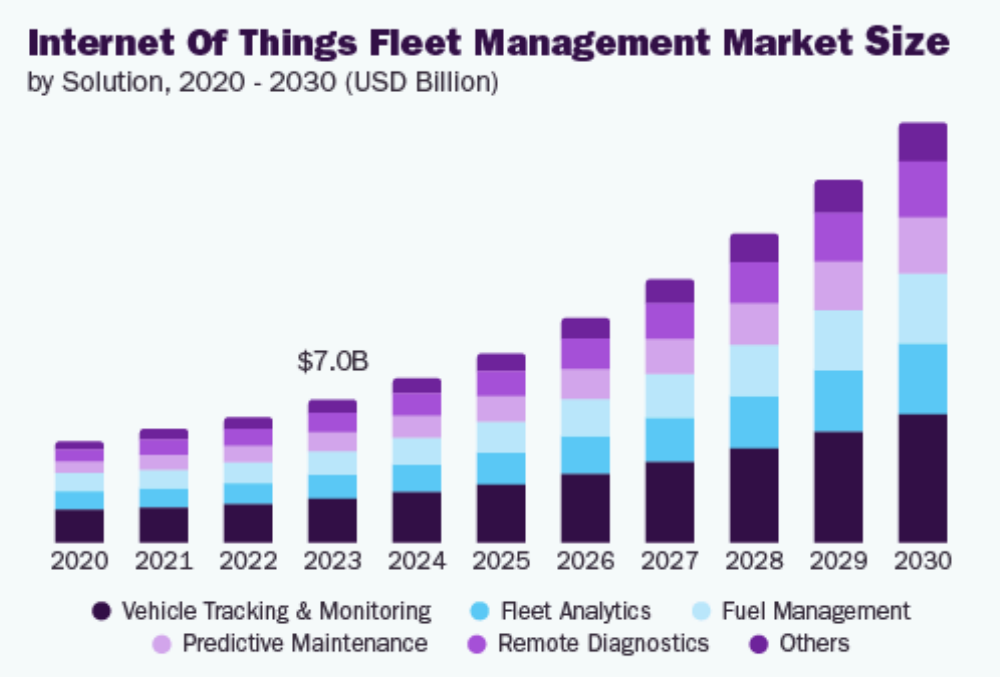

Рис. 1.1.1. Розмір ринку IoT управління рухомим складом

Моніторинг геопозиціювання є критично важливим для динамічної логістики, коли ситуація змінюється в реальному часі: затори, зміни в графіку, нові запити клієнтів, перенаправлення транспорту. Це напряму пов’язано з класом задач динамічної маршрутизації, які активно досліджуються в науковій літературі. У таких роботах наголошується, що наявність реального часу і актуальної інформації дозволяє коригувати план доставки та підвищувати якість рішень порівняно зі статичним планом. Наприклад, у дослідженні з динамічних часів руху розглядаються стратегії диспетчеризації, які покращують рішення завдяки допустимим відхиленням від поточного плану. [4] Аналогічно, наявність оновленої в реальному часі інформації (трафік, нові заявки) розглядається як ключовий чинник у задачах dynamic vehicle routing. [5] Сучасні підходи також застосовують методи штучного інтелекту для динамічних і невизначених умов маршрутизації, що підтверджує технологічний тренд розвитку програмного забезпечення у цій сфері. [6]

На сьогодні геозони стали базовим інструментом диспетчера через те, що логістичні процеси прив’язані до географії: склади, хаби, точки доставки – це все можна позначити окремими зонами. Геозона (Geofencing) – це віртуальна ділянка на географічній карті, що використовується для контролю переміщень, автоматизації певних дій та для цілей безпеки. Вони дають можливість не лише візуалізувати, а і формалізувати логістичні події. Без геозон складно автоматизувати контроль прибуття/виїзду, стоянок, відповідності маршруту, а також аналіз простоїв. Геозони є стандартною функцією трекінгу: у відкритих платформах на кшталт Traccar геозони є ключовим модулем, підтримуються різні типи зон (полігон/коло/маршрут) та пов’язані події входу/виходу (рис. 1.1.2.) [7]

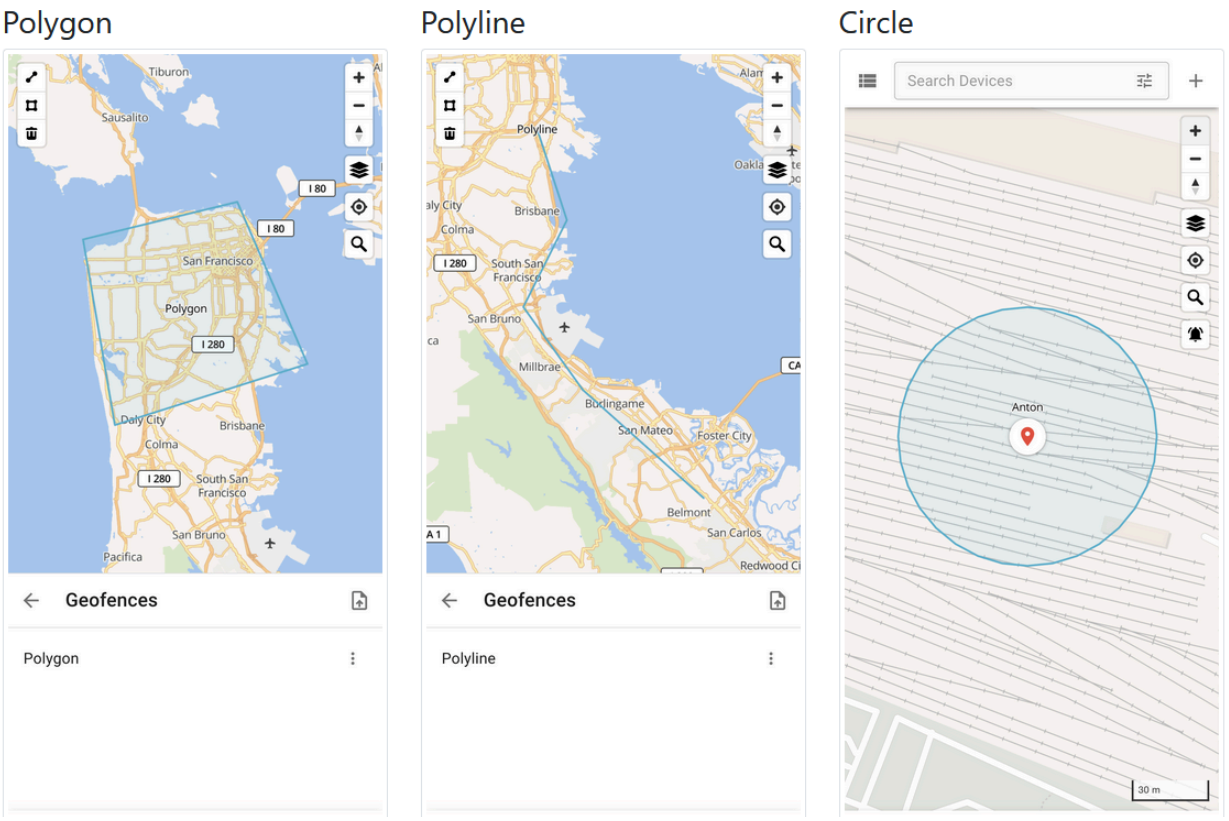

Рис. 1.1.2. Типи геозон

З наукової точки зору роль геозон у логістиці підтверджується дослідженнями, які розглядають geofencing як механізм підвищення керованості логістичних потоків, автоматизації контролю та зменшення втрат в логістичному ланцюгу. Наприклад, у роботі, присвяченій застосуванню геозон і суміжних технологій у логістичному менеджменті, підкреслюються можливості підвищення ефективності прийняття рішень і зменшення втрат у процесі логістичного потоку.[8] Також в оглядових матеріалах щодо geofencing у логістиці відзначаються сценарії автоматичних сповіщень під час входу/виходу із зон, контроль відхилень та оперативне реагування. [9]

Основні сценарії використання цього інструменту в логістиці наведені в Таблиці 1.1.1.

Таблиця 1.1.1.

Сценарії використання геозонування

Логістична подія	**Геоправило (умова)**	**Очікуваний ефект для диспетчеризації**
В’їзд у зону складу/хабу	Об’єкт перетнув межу geofence (enter)	Автоматичний чек-ін; фіксація часу прибуття; скорочення ручної звітності та дзвінків
Виїзд із зони складу/клієнта	Об’єкт перетнув межу у зворотному напрямку (exit)	Фіксація завершення операції; контроль фактичного часу обслуговування; підстава для KPI time on site
Простій на точці доставки	Перебування в межах geofence понад N хв (dwell time)	Виявлення затримок; ескалація проблем (черга, документи, відсутність готовності вантажу); зменшення простоїв
Відхилення від маршруту	Вихід за межі дозволеного коридору/маршруту (route corridor) або перетин забороненої зони	Контроль дисципліни маршруту; зниження ризиків (безпека вантажу, порушення регламентів); швидке реагування
Потенційна втрата сигналу або неактуальні координати	Останнє оновлення координат старіше за поріг T (stale data threshold)	Оперативний alarm «об’єкт не на зв’язку»; контроль працездатності трекерів; зниження «сліпих зон» у диспетчеризації
Наближення до зони (підготовка ресурсу)	Об’єкт на відстані R від зони (proximity rule)	Підготовка складу/персоналу до прибуття; оптимізація завантаження рамп/ресурсів; підвищення швидкості обробки

Наведені сценарії показують, що геозони є не лише інструментом візуалізації, але й основою для формалізації подій та управлінських реакцій. Це безпосередньо впливає на вимоги до сучасного програмного інтерфейсу систем могіторингу: підтримка полігональних зон, операції створення, редагування, видалення, перевірка коректності конфігурації, а також однозначне відображення подій і станів (вхід, вихід, тривале перебування, відсутність оновлень координат).

Потреба зв’язуватись з водієм або керувати рухомими об’єктами в межах окремої геозони виникає тоді, коли локація автоматично означає певний етап процесу або ризик. Наприклад, при в’їзді в зону складу диспетчер може одразу надіслати інструкцію щодо відповідної рампи, порядку оформлення або точки очікування. У сервісах GPS-моніторингу геозони часто використовують для автоматичного визначення ключових місць і контролю активності в них, зокрема у Verizon Connect Reveal «місця» застосовуються для моніторингу відвідувань і часу виконання робіт у визначених локаціях, що напряму підтримує диспетчеризацію за місцем. [10] Якщо транспорт довго знаходиться в зоні – це сигнал про затримку на завантаженні або проблеми з документами, і диспетчер має підставу зв’язатися з водієм або складом. У Verizon Connect геозони можуть бути тригерами подій на кшталт «зупинка в- геозоні» або «перебування в геозоні довше встановленого часу», що є типовим механізмом контролю простоїв.

У випадку заборонених зон (наприклад, приватні території, зони ризику, небезпечні ділянки) геоправило «в’їзд у заборонену зону» дає підставу негайно зупинити рейс, змінити маршрут або підтвердити, що рух санкціонований. У Quartix це прямо формулюється як «дозволена/заборонена зони» зі сповіщеннями в реальному часі при порушенні, що ілюструє практику оперативного керування флотом через правила зон. [11] Інший регулярний сценарій, коли техніка або вантаж повинні залишатися в межах визначеної території (місто, об’єкт, зона обслуговування), і диспетчер отримує сигнал, якщо транспорт виходить за межі «дозволеної зони». Для охорони активів (причепів, контейнерів, спецтехніки) геозона «периметр» дозволяє одразу реагувати на несанкціоноване переміщення, а

підхід «отримати сповіщення при перетині меж» описаний як базова функція у постачальника сервісу Naviху. [12]

Геозони застосовують також для автоматизації обліку робіт на об’єктах: коли транспорт приїхав на майданчик, система фіксує початок робіт і це зменшує ручні звіти. Наприклад, ABAX описує сценарій автоматичного документування поїздки при вході в геозону і використання часу в зоні для точнішого обліку. [13] Для логістичних операторів важливий також сценарій «контроль дотримання маршруту або сервісної зони», де відхилення від дозволених зон вимагає контакту з водієм і перепланування. У Wialon геозони описуються як віртуальні периметри для контролю руху, звітів і безпеки, тобто як основа для автоматизації операційних реакцій. [14] Таким чином, геозони виступають тригером для комунікації (зв’язатися з водієм/складом/клієнтом) та для керування (перепризначення задач, зміна маршруту, ескалація інциденту) саме тоді, коли транспорт потрапляє в конкретний просторовий контекст.

Логістика майже завжди використовує кілька систем для автоматизації бізнесу. Це і CRM (Customer Relationship Management) для управління відносинами з клієнтами з функціями автоматизації продажів, контролю угод, історії комунікацій, і ERP (Enterprise Resource Planning) для планування ресурсів підприємства з функціями обліку, управління фінансами, закупівель, управління персоналом, управління виробництвом, аналітикою, і WMS (Warehouse Management System) для управління складом з функціями контролю залишків, адресного зберігання, інвентаризації, оптимізації збору та відвантаження товарів, і TMS (Transportation Management System) для управління транспортом та логістикою з Функціями планування маршрутів, контролю завантаженості машин, відстеження вантажів у дорозі, розрахунку витрат на пальне. Саме з цієї причини ринок рухається до платформ з відкритими API та інтеграціями. І тому сучасні рішення позиціонуються як платформи з API. Приклад: Geotab наголошує на «open platform» та SDK/API для інтеграцій. [15] Samsara також підкреслює «open API» та екосистему інтеграцій. [16]

В Україні одним із прикладів практичного використання геомоніторингу рухомих об’єктів є сервіс EasyWay, що надає інформацію про маршрути і зупинки громадського транспорту, а в окремих містах забезпечує відображення GPS-даних транспорту в реальному часі. Сервіс також надає API для доступу до даних маршрутів, зупинок та GPS-позицій на маршруті, що демонструє типову клієнт-серверну модель публікації геоінформаційних даних та їх повторного використання зовнішніми застосунками. [17]

Дані стають «розумнішими»: відслідковується тенденція руху від простих координат до поведінкової аналітики. Сучасні системи все рідше обмежуються результатом «точка на карті». Тренд у розробці галузевого ПЗ це: аналіз траєкторій (знаходження шаблонів, аномалії, прогноз прибуття, кластеризація маршрутів), комбінування даних (GNSS (Global Navigation Satellite System, глобальна навігаційна супутникова система), сенсори, події), застосування ML/DL для моделювання мобільності.

Науковий огляд щодо deep learning для даних траєкторій показує, що «trajectory data» розглядаються як ключовий об’єкт аналізу мобільності, з багатьма сценаріями використання. [18]

«Розумні міста» і ITS потребують просторово-часового моніторингу. Міста та інфраструктурні оператори переходять до систем, які мають не просто «дивитись» на потік, а контролювати умови/вимоги у просторі й часі (перевантаження, безпека, інциденти, дотримання правил). Приклад наукового напряму, який підкреслює потребу в реальному часі моніторити міські процеси на основі IoT і просторово-часових специфікацій. [19]

Отже, сучасні потреби ринку включають:

- скорочення простоїв і затримок, коли диспетчер бачить об’єкти на карті та швидше реагує на відхилення;
- керування зонами складів/хабів, коли є можливість оперативно створювати/коригувати геозони під реальні зміни;

- надійність моніторингу;
- інтеграційність, коли API-підхід дозволяє підключати систему до внутрішніх сервісів компанії.

Тенденції, які підсилюють актуальність вибраної теми, включають відео з дронів та супутників як додаткові джерела геоданих. Ажде, окрім GNSS, ринок активно використовує комп’ютерний зір, моніторинг мобільності через БПЛА і детекцію об’єктів. [20] Трекінг рухомих об’єктів на супутниковому відео (малих/щільних/розмитих), що актуально для великих територій та інфраструктури. [21] Таким чином, «геоінформаційні дані» сьогодні, це не лише GPS-координати, бо сучасні системи еволюціонують у мультиджерельні платформи.

1.2 Особливості сучасних схем геомоніторингу транспорту

Традиційно геомоніторинг транспорту у логістиці та диспетчеризації реалізується як багаторівнева система, що поєднує супутникове позиціювання (GNSS), бортову телематику, канали зв’язку, серверну обробку даних та клієнтські інтерфейси. Така схема сформувалась як індустріальний стандарт, оскільки дозволяє одночасно забезпечити регулярне отримання координат, централізоване зберігання і аналітику, а також швидке відображення інформації диспетчеру на карті.

Основою визначення місцезнаходження транспортного засобу є приймач GNSS, який обчислює координати за сигналами супутників. Сучасні трекери часто підтримують мультиконстеляційне позиціювання (GPS, GLONASS, Galileo, BeiDou), оскільки це підвищує доступність супутників і покращує точність та надійність у складних умовах (міська забудова, часткові перекриття неба). Переваги мультиконстеляційного підходу підтверджуються дослідженнями, де показано, що поєднання кількох GNSS-систем підвищує видимість супутників, геометрію розташування, показники точності та надійності позиціювання. Наприклад, у роботі щодо моделі позиціювання з використанням GPS, GLONASS, Galileo, BeiDou демонструються покращення точності й стійкості позиціювання в обмежених середовища. Аналогічно, дослідження точного позиціювання з використанням чотирьох констеляцій показує зростання доступності та покращення точності при комбінуванні систем. [22]

Другий рівень схеми, це бортовий трекер, тобто телематичний модуль, який:

- отримує координати з GNSS-приймача;
- може збирати додаткові параметри (швидкість, напрямок руху, стан запалення, дані з CAN/OBD, датчики тощо);
- формує телеметричні повідомлення та передає їх у систему моніторингу;

- інколи буферизує дані при втраті зв’язку, щоб не втрачати історію.

На практиці така «клієнтська частина» працює в парі із сервером, що приймає повідомлення від трекерів та надає інтерфейс для перегляду, що добре ілюструється прикладом типової платформи GPS-трекінгу Traccar, де система описується як серверний компонент із веб-інтерфейсом для моніторингу пристроїв.[7]

Передача даних від транспортного засобу до серверної інфраструктури виконується переважно через стільникові мережі (2G/3G/4G/LTE). Для IoT-пристосувань також застосовуються легкі протоколи та моделі обміну повідомленнями. Поширеним прикладом є MQTT, який розроблений як легкий pub/sub протокол для телеметрії у мережах із обмеженою пропускною здатністю або нестабільними каналами і широко використовується в IoT-сценаріях. [23] Однак, даний перелік не обмежується зазначеними системами, особливо у місцях де відсутнє чи не доцільне покриття відповідними мережами. Там можуть бути застосовані пропрієтарні протоколи спеціалізованого обладнання, як, наприклад, радіостанції.

Сервер телематики виконує роль централізованого ядра, що:

- приймає потоки координат і телеметрії;
- нормалізує та валідує дані (прив’язує до об’єкта, коригує/фільтрує некоректні значення);
- зберігає дані у базі (поточний стан та історія);
- генерує події (наприклад, в’їзд/виїзд із геозони, перевищення швидкості, простій, відсутність оновлень);
- надає дані клієнтам через API.

Типовий ланцюг проходження даних від трекера до диспетчерського інтерфейсу наведено на рис. 1.2.1. Кожна ланка цього ланцюга може бути джерелом затримок, втрат або спотворення даних, тому система має проєктуватися як така, що працює в умовах часткових відмов і невизначеності якості геоданих.

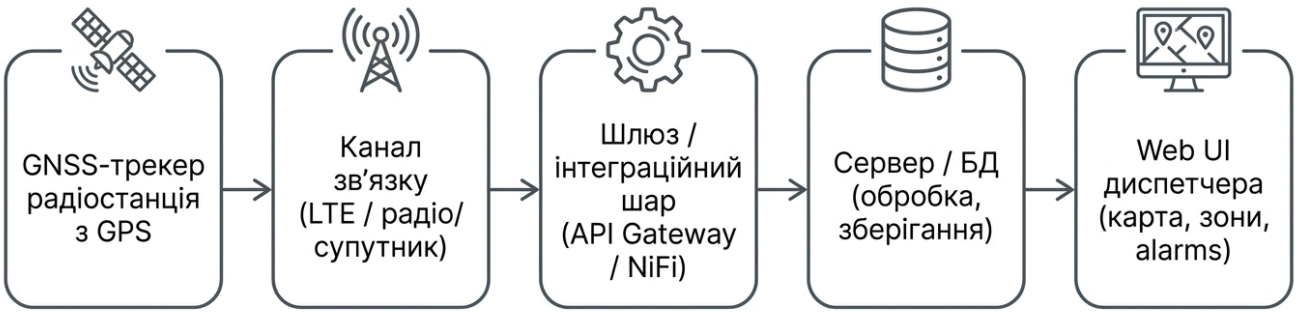

Рис. 1.2.1. Архітектурний ланцюг геомоніторингу рухомих об'єктів

Наявність інтеграцій (з CRM/ERP/WMS/TMS) означає потребу у стабільному API-контурі та чіткому розділенні операційних даних моніторингу і конфігураційних даних (геозони, довідники), щоб система могла розвиватися як інтегрований модуль у більшій екосистемі.

Типовою ознакою сучасних систем є наявність REST API для інтеграції та інколи каналу «живих» оновлень (наприклад WebSocket) для оперативної синхронізації з інтерфейсом диспетчера. Як приклад, Traccar описує REST-доступ до функціоналу сервісу, а також WebSocket endpoint для live updates. [24]

Останній рівень, це інтерфейс користувача (найчастіше web-додаток), який відображає карту з маркерами транспорту, список об'єктів, фільтрацію, статуси, події та сповіщення, геозони та їх параметри.

У традиційній web-реалізації обмін даними з сервером виконується через HTTP-запити. Базовим механізмом у браузері є XMLHttpRequest або fetch, які дозволяють отримувати дані без перезавантаження сторінки, що і реалізує класичну клієнт-серверну модель у веб-інтерфейсах. [25]

Попри практичну корисність геозон, у реальних логістичних системах виникають обмеження, які потрібно враховувати при проєктуванні. Похибка GNSS

і дрейф координат може спричиняти хибні спрацювання (особливо на межі зони). Міські умови (щільна забудова) ускладнюють стабільність координат, що впливає на enter/exit події. Налаштування зон також можуть вплинути на ефективність відстежування рухомих об’єктів: надто малі зони збільшують ризик хибних подій, надто великі зменшують точність операційного контролю.

Ці ризики відзначаються і в оглядових матеріалах про geofencing у логістиці, де поряд з перевагами згадуються виклики точності GPS, інтеграції та приватності.[9]

Одні з найпідступніших реальних технічних ризиків сучасності полягають в тому, що дані можуть зникати або спотворюватись. Диспетчеризація залежить від достовірності координат. На практиці трапляються випадки: відсутність зв’язку/живлення у трекера, застарілі координати, перешкоди GNSS або навмисні впливи (jamming/spoofing) у певних регіонах. Загроза GNSS jamming/spoofing постійно зростає. Моніторинг рухомих об’єктів часто довіряє GNSS як джерелу істини. Але сьогодні GNSS дедалі частіше піддається перешкодам і підміні сигналу:

- Європейські інституції та проєкти прямо називають maritime spoofing проблемою безпеки та надійності навігації; приклад, ініціатива з протидії.[26]
- В авіації EASA оновлює бюлетені про GNSS outages/alterations, підкреслюючи ризики jamming/spoofing, особливо біля зон конфліктів, і потребу альтернативних процедур. [27]

Це важливо, адже сучасна система моніторингу повинна враховувати, що геодані можуть бути неточними або атакованими. Це породжує попит на механізми валідації, крос-перевірки джерел, а також на «geofence/zone rules» і сповіщення по аномаліях.

Окрім всього, варто зважати на регуляторні та етичні вимоги до даних геолокації. У логістиці геодані часто прямо або опосередковано стосуються водія.

Це означає, що моніторинг має бути законним, пропорційним, прозорим і контрольованим (особливо для службового транспорту). [28]

Описані ризики та обмеження геомоніторингу напряму впливають на архітектурні рішення та функціональні вимоги системи моніторингу. Необхідно контролювати якість координат і актуальність даних. Через можливі втрати зв'язку, застарілі координати або спотворення GNSS система повинна мати механізми виявлення неактуальних даних і відображати це у вигляді сповіщень та статусів об'єктів. Необхідна також і певна толерантність до похибок позиціювання при геозонуванні. Дрейф координат біля межі зони може спричиняти хибні enter/exit-події. Тому при проєктуванні підсистеми геозон доцільно закладати мінімальні вимоги до розміру та форми зон (наприклад, уникати надто малих полігонів для реальних умов GNSS), регламентувати правила конфігурації зон (зокрема контроль перетинів, узгодження меж), передбачити можливість оперативного редагування зон, оскільки межі реальних зон часто змінюються (тимчасові зони об'їзду, зони ризику). Окрім цього, оскільки джерела даних можуть бути різними (LTE-трекери, IoT-датчики, пропрієтарні радіомережі), архітектура повинна підтримувати відокремлений від UI проміжний інтеграційний шар для нормалізації, валідації і маршрутизації даних.

Таким чином, типова схема геомоніторингу є багаторівневою системою, де якість кінцевого результату визначається не лише точністю GNSS, але й надійністю передачі, коректною обробкою та зручністю диспетчерського інтерфейсу. З урахуванням ризиків (дрейф координат, застарілі дані, можливі спотворення сигналу) інтерфейс моніторингу має забезпечувати механізми контролю актуальності координат, прозоре відображення станів та надійний цикл керування геозонами.

1.3 Висновки до першого розділу

У першому розділі виконано аналіз предметної області моніторингу рухомих об’єктів на основі геоінформаційних даних та показано, що для логістики, диспетчеризації, сервісних служб і безпекових сценаріїв геомоніторинг фактично перетворився з додаткової функції на базовий інструмент операційного управління. Наявність актуальної інформації про місцезнаходження об’єктів знижує втрати часу на ручні уточнення, прискорює прийняття диспетчерських рішень, підтримує контроль SLA, а також формує основу для більш складних задач (прогноз часу прибуття, аналіз відхилень, оптимізація маршрутів, виявлення аномалій).

Окремо обґрунтовано роль геозон як механізму «прив’язки» операційних подій до простору. Геозони дозволяють не лише відображати об’єкти на карті, а й формалізувати логістичні стани та правила: в’їзд/виїзд із зони, контроль простою, відхилення від маршруту, наближення до ключових точок. Таким чином, геозони виступають тригерами для реакції диспетчера та для автоматизації типових операційних дій.

Разом з тим визначено ключові ризики, що впливають на вимоги до системи: похибки GNSS і дрейф координат, нестабільність зв’язку, застарілі дані, а також загрози jamming/spoofing. Додатково враховано організаційні та регуляторні аспекти, оскільки геодані можуть стосуватися персоналу і потребують відповідального використання.

Підсумком розділу є висновок, що сучасне рішення моніторингу має будуватися як інтегрована клієнт-серверна система з API-орієнтованим обміном даними: джерело координат, канал зв’язку, інтеграційний шар, сервер/БД і Web UI диспетчера. Це логічно підводить до необхідності формалізації вимог та проєктування архітектури рішення у наступному розділі.

2 ПРОЄКТУВАННЯ СИСТЕМИ ТА ОБГРУНТУВАННЯ ЗАСОБІВ РОЗРОБКИ

2.1 Постановка задачі та сценарії використання

Виходячи з потреб та особливостей галузі, розглянутих у Розділі 1, можливостей та напрямків для реалізації різноманітних функціональностей дуже багато. Для наших цілей потрібно сконцентруватись на декількох ключових функціях, які будуть фундаментом для подальшого розвитку програмного продукту. У логістиці висока вартість затримок, простоїв і помилок планування. Також логістичні процеси прив’язані до географії. Отже, базовим функціоналом має стати відображення рухомих об’єктів на карті, а функціоналом, який надає системі специфіки використання має стати геозонування.

Функціональні вимоги до системи

FR1 Система повинна показувати поточні координати об’єктів на карті та у таблиці Radios. Система дозволяє користувачу швидко відцентрувати карту на потрібний об’єкт.

FR2 Система повинна показувати alarms для об’єктів, які довго не надсилають координати.

FR3 Система повинна підтримувати геозони у вигляді полігонів. Система дозволяє створювати і редагувати зони прямо на карті.

FR4 Система чітко розділяє «операційні дані» (координати об’єкта) і «керування/налаштування». Під час моніторингу неможливо вносити зміни в геозони. Під час створення чи редагування геозон рухомі об’єкти не відслідковуються. Для режиму налаштування геозон необхідно авторизуватись.

FR5 Система повинна забезпечувати автоматичне визначення цільової групи рухомого об’єкта на підставі належності його поточних координат до геозони. У разі, якщо поточна група відрізняється від цільової, серверний контур повинен ініціювати зміну групи. Web UI повинен відображати поточну (визначену) геозону або ідентифікатор групи, що відповідає зоні перебування об’єкта.

FR6 Система повинна забезпечувати валідацію конфігурації геозон та запобігати збереженню зон із перетинами, оскільки перетин зон може призводити до неоднозначного визначення належності точки до зони.

FR7 Геозони повинні бути сконфігуровані із зазначенням відповідних ідентифікаторів геозон.

Таким чином закриваються основні потреби систему моніторингу. Карта та маркери відповідають базовій потребі видимості логістики. Геозони-полігони підтримують контроль ділянок/складів/хабів/точок, дають можливість робити певні дії, як от зв’язатись з усіма об’єктами конкретної геозони. Alarms по неактуальним координатам відповідають реальній проблемі втрати даних.

Нефункціональні вимоги до системи

NFR1. Продуктивність. Час першого завантаження сторінки: до 3–5 секунд (за нормального інтернету). Оновлення координат не повинно «заморожувати» інтерфейс при типовій кількості об’єктів (орієнтовно 100 об’єктів). Частота polling задається параметром; при великій кількості об’єктів має бути можливість збільшити інтервал.

NFR2. Надійність та відмовостійкість. При відсутності відповіді сервера UI має показувати попередження/статус «No response», продовжувати спроби згідно інтервалу. При помилках API CRUD має бути коректне повідомлення користувачу і збереження консистентного стану UI.

NFR3. Юзабіліті. Диспетчер має виконувати основні операції без навчання (подивитись об’єкт, знайти в таблиці, побачити alarms). Візуальні стани повинні бути однозначними: активний/неактивний. Всі критичні операції (видалення зон/об’єктів) потребують підтвердження.

NFR-4. Сумісність. Підтримка основних браузерів (Chrome/Edge/Firefox). Коректна робота при різних роздільних здатностях.

Очікувані результати

У результаті виконання роботи має бути отриманий веб-інструмент (web UI) для моніторингу та базового керування просторовими даними.

Основний функціонал – диспетчерська карта з відображенням рухомих об’єктів. На карті показуються маркери об’єктів (ідентифіковані як RSSI) з координатами, групами та часом останнього оновлення. Користувач отримує наочну операційну картину в реальному часі (у межах частоти оновлення).

Механізм контролю доступності/актуальності даних реалізований через логіку сповіщень: якщо об’єкт довго не оновлює координати, він потрапляє в список попереджень. Це дозволяє швидко виявляти ситуації типу «трекер не на зв’язку», «дані застаріли», «об’єкт зник з моніторингу».

Підсистема геозон з повним циклом CRUD включає завантаження зон з сервера і відображення на карті як полігонів, створення нових зон (малювання полігону на карті та введення параметрів), редагування меж зон (перетягування/зміна форми) та редагування параметрів (наприклад, GSSI), видалення зон із підтвердженням. Таким чином створюється керований просторовий довідник (склади, хаби, зони доставки, заборонені зони).

Підключення до кількох серверів та локальні налаштування зроблені таким чином, що користувач може обирати сервер зі списку (збереженого в браузері) і перемикатися між середовищами, налаштовуються таймаути оновлення і пороги неактивності.

Базовий клієнт-серверний інтерфейс для інтеграції отримується завдяки зафіксованому набору endpoint-ів (координати, zones, radios, login), який може бути основою для подальшого розширення (історія треків, події enter/exit, ETA тощо).

Щоб показати практичну цінність розробки необхідно передбачити хто і як зможе використати результат. Для нашої системи передбачено два основні сценарії використання.

Диспетчер / логіст / керівник зміни:

- Відкриває карту, бачить актуальні позиції об’єктів;
- Фокусує увагу на об’єктах з проблемами (alarms);
- Перевіряє, чи об’єкти знаходяться в потрібних зонах (візуально);
- Виконує певні дії з об’єктами в одній геозоні (наприклад, оповіщення по радіо зв’язку).

Адміністратор системи / оператор сервісу:

- Додає сервери, налаштовує інтервали оновлення;
- Додає/видаляє RSSI (об’єкти), підтримує актуальність довідників;
- Підтримує актуальність геозон.

2.2 Обґрунтування вибору інструментів і технологій

2.2.1 Платформа NiFi

Apache NiFi – це платформа з відкритим вихідним кодом, призначена для автоматизації потоків та обробки даних у реальному часі. Вона дозволяє легко збирати, фільтрувати, трансформувати та переміщувати інформацію між різними системами. Її найчастіше використовують як інструмент для інтеграції даних і керування потоками між системами: джерела даних, черги, сховища, API, БД, файли тощо. NiFi розроблено для масштабування в кластерах, які пропонують гарантовану доставку даних. [29]

Переваги використання NiFi включають візуальне проєктування потоків (drag-and-drop), коли потоки даних збираються з готових процесорів без великої кількості коду, швидке підключення до багатьох джерел і приймачів через наявність багатьох готових конекторів: HTTP(S), SFTP/FTP, Kafka, JMS, бази даних (через JDBC), файли, object storage, різні формати даних тощо. Також варто зазначити надійність доставки за допомогою вбудованих черг між кроками, повторних спроб і обробки тимчасових збоїв, а механізми backpressure контролюють обчислювальні ресурси. Дуже сильна сторона NiFi: видно, що відбувається з кожним фрагментом даних, де застряг процес, чому падає. TLS, керування доступами, інтеграція з корпоративними механізмами автентифікації, шифрування та контроль доступу до flow формують безпекові переваги платформи. Також рішення можна масштабувати горизонтально і робити high availability. [30]

Використання NiFi є раціональним для цілей реалізації рішення даної роботи через наступні переваги:

- асинхронні callback-и від RCM,
- повторні спроби при раптовому розірванні з’єднання між UI та сервером,
- просте моделювання інтеграційних сценаріїв,
- візуальний контроль стану потоків.

2.2.2 Інструменти розробки інтерфейсу користувача

Для обґрунтування засобів розробки веб-додатку доцільно розглянути їх функціональні можливості та переваги.

Отримання даних у реальному часі можливе з технологіями Polling, WebSocket, MQTT, SSE.

Polling – це метод взаємодії між клієнтом і сервером, при якому клієнт (браузер, бот або пристрій) регулярними інтервалами часу запитує сервер про наявність нових даних. Якщо дані є, сервер їх надсилає; якщо немає – сервер відповідає, що змін немає. Переваги Polling: не потрібні додаткові серверні компоненти або підтримка «постійних» з’єднань, легко контролювати навантаження (інтервал опитування задається), демонструє «майже real-time» моніторинг без ускладнення архітектури. Обмеження Polling: дані не миттєві, а з дискретністю інтервалу, і у деяких випадках може давати зайве навантаження при великій кількості клієнтів або високій частоті опитування.

WebSocket – це сучасний протокол зв'язку, який забезпечує постійний, повнодуплексний канал обміну даними між браузером та сервером через одне TCP-з'єднання. На відміну від HTTP, він дозволяє серверу надсилати дані клієнту в реальному часі без запиту, забезпечуючи низьку затримку. Клієнт і сервер можуть надсилати повідомлення одночасно. Один раз встановлене з'єднання залишається відкритим, що усуває потребу постійно відкривати/закривати HTTP-запити. Ідеально підходить для чатів, ігор, біржових котирувань, сповіщень, бо все відбувається в реальному часі. З'єднання встановлюється через HTTP-запит з заголовком Upgrade, після чого відбувається перехід на протокол WebSocket. WebSocket найкраще себе проявляє у випадках з дуже частими оновленнями (секунди/соті секунди), де багато об’єктів, та де потрібна мінімальна затримка. Переваги: сервер пушить оновлення одразу як з’явились, мінімальна затримка, ефективніше за часті GET якщо оновлення часті. Обмеження: реалізація складніше на бекенді через менеджмент сесій, масштабування, а також потрібно продумати безпеку каналу і авторизацію. Не підтримує автоматичне

перепідключення при обриві з'єднання або аутентифікацію за замовчуванням (ці механізми реалізуються через спеціальні бібліотеки). [31]

MQTT (Message Queue Telemetry Transport) – це легкий мережевий протокол обміну повідомленнями, який працює поверх TCP/IP. Він використовується для зв'язку між пристроями IoT (Інтернет речей), забезпечуючи передачу даних навіть в умовах обмеженої пропускної здатності, нестабільного зв'язку або малої потужності пристроїв. MQTT найкраще себе проявляє у випадках якщо трекери/датчики вже знають протокол MQTT і потрібна «шина подій» для багатьох споживачів. Клієнти не спілкуються прямо один з одним. Вони обмінюються даними через центральний вузол – брокер. Видавець відправляє дані (повідомлення) у певну тему на брокер. Підписник підписується на теми на брокері для отримання даних. Брокер отримує всі повідомлення, фільтрує їх і розсилає підписникам. Переваги: промисловий стандарт для IoT телеметрії, а також використовується для автомобільної промисловості та розумних будинків, добре масштабується для пристроїв. Обмеження: потрібен брокер (Mosquitto/EMQX тощо), складніша інфраструктура, у вебі часто потрібен MQTT over WebSocket. [32]

SSE (Server-Sent Events) – це технологія, яка дозволяє веб-серверу надсилати оновлення даних на клієнт (браузер) в режимі реального часу через єдине довготривале HTTP-з'єднання. Це односторонній канал зв'язку, де інформація йде тільки від сервера до клієнта, ідеально підходить для сповіщень, стрічок новин або оновлення графіків. Сервер штовхає (push) дані клієнту, клієнт не може надсилати дані через це ж з'єднання. Працює поверх стандартного протоколу HTTP. Якщо з'єднання розривається, браузер автоматично намагається відновити його. Дані передаються у форматі UTF-8, зазвичай як текстові блок. Переваги: простіший за WebSocket, односторонній стрім з сервера до браузера, добре для оновлень «сервер -> клієнт» у реальному часі, ідеально для стрічок новин, живих трансляцій та дашбордів. Обмеження: менш універсальний, ніж WebSocket, складніший для двосторонньої взаємодії. [33]

Обрано Polling через setTimeout та GET запити. Для цільового рішення polling є логічним компромісом: простота, зрозумілість, дозволяє отримувати оновлення близькі до реального часу без брокера і без websocket інфраструктури. Для продуктивних систем з високими вимогами до латентності варто розглянути доцільність переходу на WebSocket/SSE/MQTT.

Карта та GIS-відображення можливо реалізувати з технологіями: Google Maps API, Leaflet, Mapbox, OpenLayers

Google Maps API надає потужну модель об’єктів (Marker/Polygon/InfoWindow) та подій, вбудована Drawing Library суттєво прискорює реалізацію геозон оскільки дає можливість редагування полігону «з коробки» та демонструють, що джерело тайлів може бути стороннім. Обмеження цього підходу наступні: Google Maps API зазвичай вимагає коректного підключення/ключа (залежно від політик).

Leaflet – це популярна JavaScript-бібліотека з відкритим кодом для створення інтерактивних карт, зручних для мобільних пристроїв. Вона легка (близько 38 КБ), швидка та працює на всіх основних десктопних і мобільних платформах, забезпечуючи роботу з різними шарами (Tile, WMS, GeoJSON) та маркерами. Забезпечує високу продуктивність. Підтримує перетягування (drag panning), масштабування (zoom), роботу з маркерами та спливаючими вікнами. Дозволяє працювати з шарами, векторами, зображеннями та GeoJSON. Використовує сучасні технології HTML5 та CSS3. Leaflet часто використовується для відображення карт на веб-сайтах і є одним з основних інструментів для розробників інтерактивних мап. Переваги: легка бібліотека, добре працює з OSM та багатьма тайл-провайдерами, велика екосистема плагінів (геофенсинг, heatmaps, clustering), просте підключення, часто без важкої інфраструктури. Обмеження: редагування полігонів потребує окремих плагінів (Leaflet.Draw тощо), деякі advanced можливості треба збирати з плагінів. [34]

OpenLayers – це потужна JavaScript-бібліотека з відкритим вихідним кодом, призначена для створення інтерактивних карт у веб-браузерах. Вона дозволяє

відображати мапи, географічні дані (векторні, растрові) та взаємодіяти з ними, підтримуючи різні формати та джерела даних (WMS, TMS, GeoJSON тощо). Багатофункціональна бібліотека, яка підтримує складні географічні завдання. Дозволяє створювати карти, що масштабуються, переміщуються та оновлюються в режимі реального часу. Підтримує широкий спектр картографічних джерел, зокрема OpenStreetMap, Bing Maps, Google Maps (через спеціальні шари) та інші. Надає потужний API для розробників, написаний на JavaScript. Розповсюджується на умовах дозвільної ліцензії BSD. Переваги: дуже сильний GIS-інструмент, зручний для професійних картографічних задач, шарів, проєкцій, форматів. Обмеження: вища складність освоєння і коду, ніж у Leaflet, для певних задач UI може бути надмірним. [35]

Mapbox GL JS – це також потужна JavaScript-бібліотека, що використовує WebGL для візуалізації векторних плиток та стилів Mapbox, дозволяючи створювати інтерактивні 2D та 3D карти. Вона забезпечує високу продуктивність (60 FPS) у браузері, дозволяє динамічно змінювати стиль та відображати складні геодані в реальному часі. На відміну від растрових карт, векторні плитки завантажуються швидше і дозволяють змінювати стиль на стороні клієнта. WebGL рендеринг забезпечує плавну анімацію, плавне масштабування та роботу з 3D-моделями, будівлями та рельєфом. Можливість змінювати зовнішній вигляд карти (кольори, шари) безпосередньо під час взаємодії користувача. Підтримка маркерів, спливаючих вікон, запитів до даних та фільтрації функцій на карті. Підтримка 3D-будівель, рельєфу та режимів відображення глобуса. Є частиною великої екосистеми Mapbox, включаючи SDK для мобільних платформ (iOS, Android). Переваги: сучасний рендеринг, стилізація, 3D, висока продуктивність, гарні можливості візуалізації. Обмеження: ліцензійні/тарифні аспекти, залежність від сервісу, потребує певного стилю роботи з даними та шарами. [36]

Обрано варіант Google Maps JavaScript API через ImageMapType. Google Maps API та Drawing дає найшвидший шлях реалізувати «зони» та інтерактивну мапу в одному файлі. Drawing library дозволяє зробити редактор зон без GIS-фреймворків.

Архітектура фронтенду може бути реалізована за допомогою Vanilla JS, React, Vue, Angular.

Vanilla JavaScript – чиста, нативна мова програмування JavaScript, написана без допомоги сторонніх бібліотек, плагінів чи фреймворків. Код виконується безпосередньо в браузері, оскільки це і є мова JavaScript у її сирому вигляді. Не потрібно підключати зовнішні файли бібліотек, що зменшує вагу сайту та час завантаження. Оскільки немає накладних витрат на роботу фреймворків, код часто працює швидше. Розуміння Vanilla JS є фундаментальним для розробника перед тим, як переходити до складних фреймворк. Переваги: легко показати, як усе працює, без складної збірки, мінімальні залежності, просте розгортання, відсутність зайвих абстракцій забезпечує миттєву реакцію, сучасні браузери мають відмінну підтримку стандартів ES6+ (EcmaScript), що робить код універсальним. Ідеально підходить коли пріоритетом є максимальна швидкодія та легкість сайту. Обмеження можуть виникнути з необхідністю ускладнювати функціонал, оскільки таке рішення складніше масштабувати через підтримку стану, модулів, тестування, типізацію, а також з часом код стає важче підтримувати без структурування.[37][38]

React, Vue та Angular – це провідні JavaScript-інструменти для розробки фронтенду. Вибір серед них залежить від масштабу проекту, необхідної швидкості розробки та складності архітектури.

React (від Meta) – найпопулярніша бібліотека (UI-компоненти) для створення динамічних інтерфейсів, використовує компонентний підхід та віртуальний DOM. Переваги: велика спільнота, гнучкість, використання JSX, легка інтеграція. Ідеальна для великих SPA, проектів, де важлива швидкість розробки та гнучкість.

Vue.js – це легкий прогресивний фреймворк. Поєднує найкраще з React та Angular. Його легко вивчати та інтегрувати в існуючі проекти. Переваги: простота, чудова документація, висока продуктивність, двостороннє прив'язування даних. Ідеальний для швидкої розробки, проектів будь-якого розміру, невеликих додатків.

Angular (від Google) – потужний повнофункціональний фреймворк. Масштабне рішення, що надає повний набір інструментів (маршрутизація, робота з формами, запити) з коробки. Переваги: сувора архітектура, TypeScript за замовчуванням, відмінно підходить для великих команд. Використовується для складних корпоративних додатків та довгострокових проеків.

Таблиця 2.2.2.1.

Порівняння JavaScript-інструментів

Критерій	**Vanilla JS**	**React**	**Vue.js**	**Angular**
Продуктивність	Висока	Варіюється	Варіюється	Варіюється
Розмір бандла	Мінімальний	Середній	Середній	Великий
Гнучкість і контроль	Повний контроль над кодом, гнучкість у виборі рішень	Обмежена гнучкість через прив’язку до компонентів React	Обмежена гнучкість через прив’язку до компонентів Vue	Обмежена гнучкість через модульну структуру Angular
Завантаження та ініціалізація	Швидке завантаження та ініціалізація	Залежить від розміру і складності програми, може бути повільною		
Оновлення та обслуговування	Легкий процес оновлення та підтримки коду	Можуть виникати проблеми		
Сумісність із браузерами	Підтримка у всіх сучасних браузерах			

Загальні переваги React/Vue/Angular: краща структура, компоненти, керування станом (Redux/Pinia тощо), зручне тестування, легше масштабувати UI і командну розробку. Обмеження: потрібна збірка (Vite/Webpack), більше інфраструктури, більше залежностей, для обмеженого функціоналу може бути зайвим. [39][40]

Обрано: Vanilla JS, single-file, оскільки такий підхід раціональний для реалізації обраного функціоналу, але під потреби розширення функцій можна розглянути міграцію на компонентну архітектуру.

Протокол взаємодії з сервером під задачі системи можна вибирати між REST-подібний HTTP або GraphQL.

REST-подібний HTTP API – це підхід до проектування веб-сервісів, який використовує стандартні можливості протоколу HTTP (методи, URL, коди відповідей) для маніпуляції ресурсами. Це не суворий стандарт, а архітектурний стиль, який забезпечує просту та зрозумілу взаємодію між клієнтом і сервером. Сервер не зберігає інформацію про попередні запити клієнта (Stateless). Кожен запит містить усю необхідну інформацію. Найчастіше використовується формат JSON, рідше XML, для обміну даними. Переваги: використання основних HTTP-методів в рази спрощує розробку та інтеграцію, REST не прив'язаний до жодної мови програмування чи платформи, що забезпечує високу сумісність, принцип безстанової взаємодії дозволяє легко розподіляти запити між багатьма серверами (балансування навантаження) без втрати контексту, чіткий поділ клієнта та сервера дає змогу розробникам оновлювати фронтенд та бекенд незалежно одне від одного, використовує вбудовані HTTP-механізми для кешування відповідей, що зменшує мережевий трафік та навантаження на сервер.

Обмеження: клієнт часто отримує більше даних, ніж йому необхідно (Over-fetching), оскільки кінцева точка (endpoint) повертає фіксований набір полів. Часом для отримання повного набору пов’язаних даних клієнту часто доводиться робити кілька послідовних запитів (наприклад, спочатку запит на список товарів, потім – окремі запити на деталі кожного товару). Реалізація REST може сильно

відрізнятися між різними розробниками, ускладнюючи уніфікацію. Хоча REST є безстановим, для деяких операцій необхідно розробляти додаткові механізми для збереження стану на боці клієнта. Безпека даних в межах REST API покладається на зовнішні механізми (наприклад, SSL/TLS, OAuth, токени), неправильна реалізація цих механізмів може призвести до витоку даних. [41]

GraphQL – це мова запитів для API з відкритим вихідним кодом та середовище виконання для обробки цих запитів, розроблена Facebook у 2012 році та випущена у 2015 році. Вона дозволяє клієнтам точно вказувати, які дані їм потрібні, запобігаючи надлишковому завантаженню та забезпечуючи гнучкість взаємодії. Запити будуються на основі схеми – суворого опису типів даних та зв'язків між ними. Єдина точка доступу дозволяє агрегувати дані з декількох джерел. Переваги: гнучкі запити «потрібні поля», один endpoint, менше надлишковості даних, frontend-розробники можуть самостійно змінювати запити без участі backend-команди. Обмеження: складніший сервер, потрібна схема, резолвери, контроль доступу, для невеликого API не завжди виправдано. [42]

Обрано: простий REST-подібний набір endpoint-ів через простоту інтеграції, масштабування та дебагування. REST API відрізняється від GraphQL підходом до обміну даними та рівнем гнучкості. REST API базується на простих HTTP-методах і частіше застосовує JSON, що робить його легшим в інтеграції. GraphQL дозволяє клієнту вимагати лише необхідні дані, тоді як REST повертає фіксовані структури. Завдяки цьому REST API вважається оптимальним балансом між простотою та універсальністю. [43]

Для дій над ресурсами будуть використовуватись стандартні методи:

- /zones (GET/POST/PUT/DELETE)
- /radios (POST/DELETE)
- /login (POST)
- GET / (координати)

Зберігання налаштувань клієнта можна реалізувати локально, або через профіль на сервері. Для потреб системи обрано localStorage, бо не потрібна база

даних для персональних налаштувань та простіше реалізовується коли список серверів зберігається локально. При цьому налаштування не синхронізуються між пристроями/користувачами, а також можлива втрата налаштувань при очистці браузера, але навіть для продуктивної системи із диспетцерським сценарієм використання це не виглядає критичним. До того ж для серверного зберігання профілю/налаштувань необхідно реалізовувати аудит і політики (особливо важливо для безпеки).

Можливі варіанти подальшого удосконалення рішення (по-за рамками даної роботи) при підготовці до перенесення на продуктивну систему:

- Переведення на HTTPS.
- Auth через короткоживучі токени (JWT) або сесії.
- Окремий endpoint для отримання токена, refresh token.
- RBAC (доступ на основі ролей), логи для аудиту.

У вибраному наборі інструментів рішення очікується архітектурно простим, швидким щодо побудови GIS-функціоналу (карта, зони, маркери, CRUD), і мінімізуватиме інфраструктурну складність.

2.3 Висновки до другого розділу

У другому розділі виконано формулювання вимог та очікуваних результатів програмного продукту моніторингу рухомих об’єктів та обґрунтовано вибір технологічних рішень відповідно до цілей дипломної роботи і сформованих вимог. На основі аналізу предметної області визначено, що практична цінність диспетчерських систем безпосередньо залежить від швидкого отримання актуальних координат, наявності геозон як інструменту операційного контролю, а також від інтеграційності рішення через API, що дозволяє включати систему в реальні контури бізнесу та зв’язку.

Ключовим є те, що геозони у проєкті мають не лише візуальну функцію, а і прикладну диспетчерську роль. Таким чином, Web UI виступає інструментом ситуаційної обізнаності, а бекенд забезпечує виконання керуючих дій у контурі зв’язку.

Вибір Apache NiFi як інтеграційного шару є доцільним для потокових сценаріїв, оскільки платформа підтримує візуальне конструювання процесів, надійність доставки, повторні спроби, контроль навантаження та інтеграцію з різними джерелами і протоколами. Для Web UI як раціональний підхід обрано односторінкову реалізацію на Vanilla JavaScript з Google Maps JavaScript API та REST-подібними endpoint-ами; механізм polling визначено як оптимальний компроміс між простотою та достатньою актуальністю даних.

Таким чином, у розділі закладено цілісну архітектурну основу, на базі якої у подальшій частині роботи наведені рішення будуть реалізовані у вигляді конкретних модулів, сценаріїв роботи та перевірки функціоналу системи.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Проектування системи

Для забезпечення прозорого зв’язку між вимогами, реалізацією та перевіркою у роботі використано матрицю трасування, яка показує, у якому модулі реалізовано відповідну вимогу та яким сценарієм вона підтверджена під час тестування.

Таблиця 3.1.1.

Матриця трасування вимог

Вимога	Компоненти реалізації	Опис реалізації
FR1	Web UI	Карта з маркерами, таблиця Radios, фокусування на об’єкті
FR2	Web UI	Обчислення неактуальності за порогом, відображення у Alarms
FR3	Web UI, API /zones	Геозони-полігони, створення/редагування/видалення
FR4	Web UI, API /login	Розділення режимів View/Edit, авторизація
FR5	NiFi	Визначення зони, ініціювання зміни групи, відображення в UI
FR6	Серверна валідація зон	Перевірка перетинів під час збереження зон
FR7	Web UI, API /zones	Обов’язкові ідентифікатори зони (name, gssi), валідація

У системі використовуються такі базові сутності:

- Zone (геозона), кожна зона в системі повинна бути визначена з наступними параметрами:
  - zone_name – унікальний ідентифікатор зон;

- zone_gssi – ідентифікатор радіо групи, що відповідає зоні;
- zone_polygon – полігон, заданий у вигляді групи координат.

- Radio (об’єкт моніторингу): ідентифікатор об’єкта (RSSI/ISSI), координати (lat, lon), атрибути стану (час останнього оновлення, поточна група).
- Alarm (сповіщення): похідна сутність, що формується за правилами неактуальності даних і відображається в інтерфейсі.

Основним завданням рішення є автоматичне перемикання рухомих об’єктів у групи, що відповідає зонам дії диспетчерів, відповідальних за ділянки.

Рішення будується на базі компонентів:

1) Tracking IL – Integration Layer – компонент для комунікації з компонентом мережі SDR. Компонент виконує два завдання:

- Приймає координати рації від SDR і відправляє їх через restAPI інтерфейс підсистему ADG, реалізовану з урахуванням платформи Nifi.
- Приймає повідомлення через restAPI від підсистеми ADG і відправляє його вказаною в повідомленні рації в мережу

2) Nifi - платформа, де реалізована бізнес логіка рішення та інтеграційні адаптери для взаємодії RCM. На платформі Nifi реалізовані наступні підсистеми:

- GFS – підсистема геозонування, що надає функцію визначення знаходження точки (заданої координатами lat, long) у передконфігурованих зонах (полігонах, заданих групою географічних координат). Кожній зоні відповідає ідентифікатор радіогрупи – gssi.

- ADG - підсистема, що виконує приймання та обробку координат рації, запит до підсистеми GFS для визначення групи, що відповідає поточним координата рації, відправлення команд і приймання відповідей у/від RCM для зміни групи рації (при необхідності). Також, підсистема ADG, при отриманні координат від SDR, виконує їхнє відправлення на визначений ISSI, через взаємодію з Tracking IL.

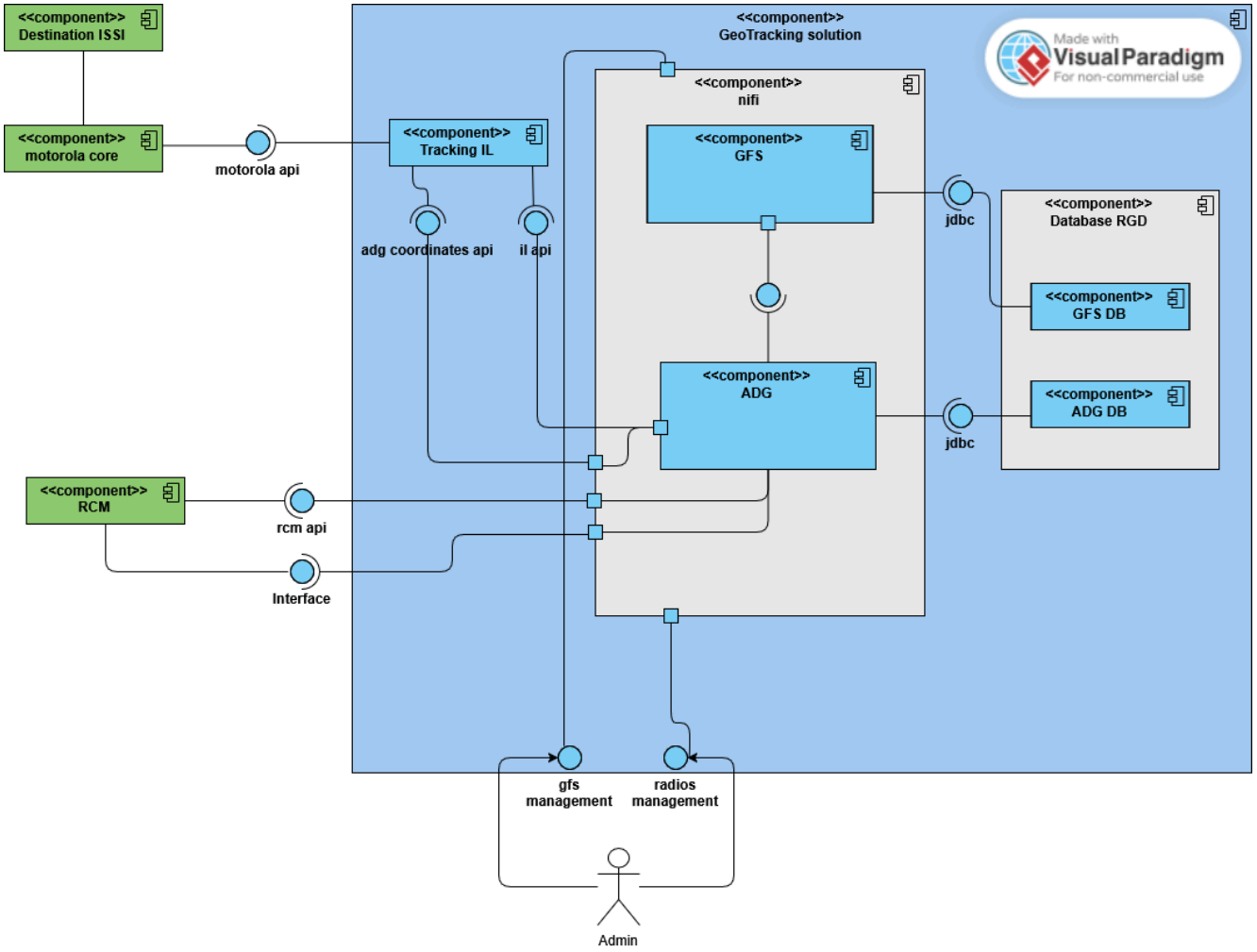

Рисунок 3.1.1 Архітектура системи

ADG підсистема складається з наступних процесів груп:

- AutomaticDetectingGroupService - функціонал отримання координат та визначення необхідності виконати зміну групи.
- RcmRequestSender – функціонал виклику методів API RCM для зміни групи

- RcmCallbackListener - функціонал отримання та обробки callback від RCM
- SendCoordinatesExternalSystem - функціонал пересилання координат рації на зовнішню систему
- RadioCheckGroup – функціонал отримання поточної групи за допомогою RCM та збереження БД
- RadioApi - функціонал для керування раціями, за якими виконується отримання координат
- ConfigApi - функціонал для керування параметрами конфігурації, які доступні для редагування
- ADG DB – база даних.

GFS підсистема складається з GFS DB і групи ZoneAPI для управління конфігурацією зон (полігонах, заданих групою географічних координат).

Запропоноване frontend рішення є веб-інтерфейсом для моніторингу рухомих об’єктів на карті та керування геозонами (полігонами). Воно реалізує карту диспетчера з даними в реальному часі, що надходять із зовнішнього сервера, а також дає інструменти для редагування довідкових/керуючих сутностей (зони, список RSSI, список серверів).

Візуалізація рухомих об’єктів на карті реалізована у вигляді маркерів для об’єктів, ідентифікованих як RSSI. Кожен об’єкт має координати (широта, довгота) та додаткові атрибути (групи, час останнього оновлення).

Система відображає територіальні зони у вигляді полігонів і дозволяє: створювати нові зони, редагувати форму (координати вершин), редагувати властивості, видаляти зони.

Передбачено механізм сповіщень для оперативного контролю стану, якщо координати давно не оновлювались – система сигналізує, що об’єкт не надсилає координати.

Користувач може обрати сервер зі збереженими геозонами із локально збереженого списку, залогінитись і працювати з його даними. Тобто, реалізовано підключення до різних джерел даних.

Це реалізація геоінформаційної підсистеми (UI, візуальний контроль та редагування геопросторових об’єктів), яка накладається на дані моніторингу (координати об’єктів).

Для роботи системи, необхідно щоб у системі були налаштовані зони, із зазначенням ідентифікаторів груп, відповідальних за ці зони.

3.2 Логіка роботи рішення

Реалізація UI вийшла у форматі Single Page Application (SPA) без фреймворків. Один HTML-файл із вбудованими CSS та JavaScript. Причини вибору цього підходу в наступному. Простота розгортання: один файл можна відкрити локально або розмістити на будь-якому веб-сервері чи навіть на локальній машині диспетчера. Мінімальні залежності: легше демонструвати як прототип. Швидкий старт і прозора логіка.

Використано Google Maps JavaScript API: google.maps.Map, Marker, Polygon, InfoWindow та Drawing Library: google.maps.drawing.DrawingManager для створення полігонів. Google Maps API дає готову інфраструктуру відображення, подій, об’єктів на мапі. Drawing library різко зменшує складність реалізації «редактора зон»: полігони можна малювати та редагувати «з коробки».

Клієнт-серверна взаємодія по HTTP (REST-подібні endpoints). REST-подібний підхід є стандартом інтеграції між UI і бекендом. За цього підходу легко розширювати функціонал: додати /history, /events, /routes тощо. Факично використано XMLHttpRequest з GET/POST/PUT/DELETE та серверні ресурси

- /zones (керування зонами),
- /radios (керування списком RSSI),
- /login (автентифікація),
- базовий GET / (отримання координат).

Нижче наведено приклади типових запитів/відповідей, що демонструють узгодженість формату обміну даними між Web UI та серверним контуром.

GET /zones (відповідь):

```
[
  {
    "zone_name": "Hub01",
    "zone_gssi": "16029",
    "zone_polygon": [
      {"lat": 50.45010, "lng": 30.52340},
      {"lat": 50.45100, "lng": 30.52420},
      {"lat": 50.44950, "lng": 30.52510}
    ]
  }
]
```

POST /zones (запит):

```
{
  "zone_name": "Hub02",
  "zone_gssi": "16030",
  "zone_polygon": [
    {"lat": 50.45000, "lng": 30.52000},
    {"lat": 50.46000, "lng": 30.52000},
    {"lat": 50.46000, "lng": 30.53000}
  ]
}
```

GET / (координати об’єктів, приклад відповіді):

```
[
  {
    "rssi": "123456",
    "lat": 50.43395,
    "lon": 30.49110,
    "group": "16029",
    "last_update": "2026-05-24T22:57:32"
  }
]
```

Періодичне опитування (polling) застосовано як метод отримання даних у «майже реальному часі». Використано setTimeout з циклічним викликом loadCoords(). Це простіше, ніж WebSockets/MQTT, і достатньо для функціоналу, який буде реалізовано у даному проекті. Також це дає контроль над частотою запитів через config.coords_retrive_timeout. Працює майже всюди без додаткових серверних компонентів.

Збереження налаштувань реалізовано локально (localStorage). Використано localStorage("adg-servers") для переліку серверів і «selected server», localStorage("adg-config") для таймаутів. За цього підходу користувачу не треба щоразу вводити список серверів та конфіг. Це швидкий спосіб «персоналізації» без окремої БД для налаштувань.

Реалізовано розділення режимів View/Edit як елемент керування ризиками і зручності. Використано editMode, що блокує polling координат у режимі редагування. У edit mode вмикаються інструменти та елементи CRUD. Це зменшує конфлікти, оскільки оновлення з сервера не перезатирає UI, у процесі коли користувач редагує. Отримуємо чітку UX-модель «оператор дивиться» проти «оператор змінює конфіг».

Отже, тут реалізовано тип архітектури веб застосунку з акцентом на візуалізацію і CRUD, що спирається на UI шар (карта, таблиці, модальні діалоги), data access шар (requestServer), domain objects у клієнті (markers Map, polygons array, zone properties), та сервер як джерело істини для координат і конфігурації зон/RSSI.

При запуску сторінки UI відображається мапа і додається тип карти з тайлами GoogleMap, ініціалізуються таблиці (Radios/Zones/Alarms), модальні діалоги, обробники подій.

Вибір сервера: список серверів відновлюється з localStorage, після вибору серверу програма скидає поточний стан і перезапускає ініціалізацію.

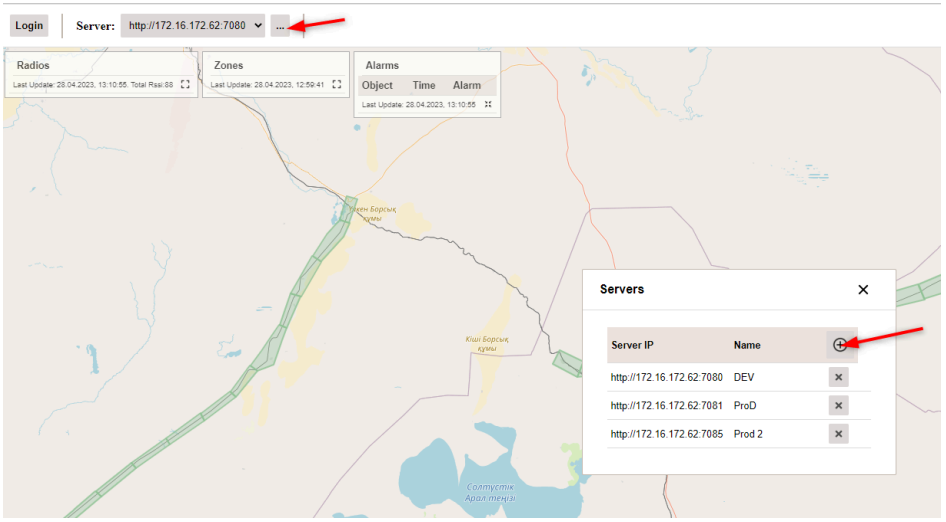

Рис. 3.2.1 Порядок дій для додавання сервера

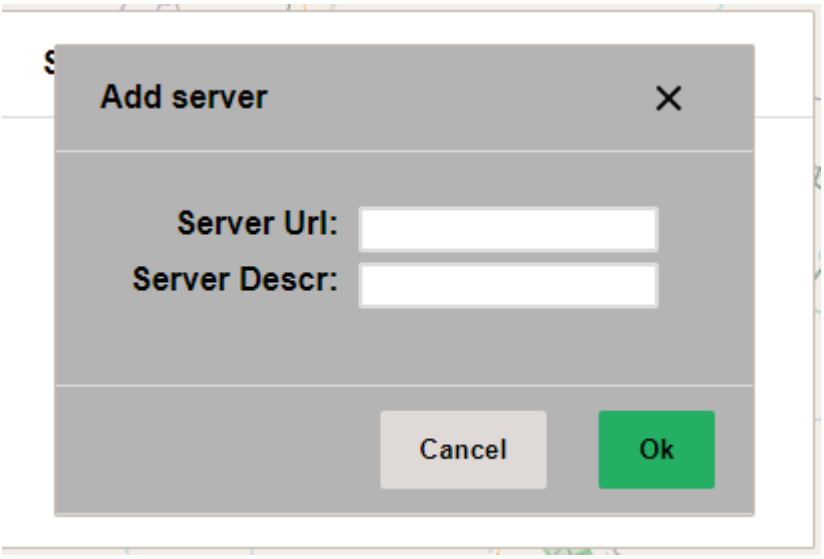

Рис. 3.2.2 Вікно додавання сервера

Для того щоб зайти в режим редагування потрібно авторизуватись. Користувач вводить логін/пароль у модальному вікні, дані надсилаються на endpoint /login. При успішному логіні відкривається доступ до режиму редагування.

Завантаження зон: запит GET /zones повертає список зон (з координатами полігонів), зони малюються на мапі як google.maps.Polygon. Відбувається циклічне завантаження координат (моніторинг). З певним інтервалом виконується GET / (базовий endpoint), який повертає поточні координати об’єктів. На основі відповіді оновлюються/створюються маркери, заповнюється таблиця Radios, генеруються Alarms, якщо дані застарілі.

У режимі View система працює в режимі моніторингу (polling координат).

У режимі Edit опитування координат зупиняється (щоб не заважати редагуванню), вмикаються інструменти малювання полігонів, відкриваються кнопки додавання/видалення RSSI та редагування зон.

CRUD операції: створення/оновлення/видалення зон: POST/PUT/DELETE /zones, додавання/видалення RSSI: POST/DELETE /radios.

У <head> підключено Google Maps JavaScript API та бібліотека drawing (для малювання полігонів) і geometry а також FontAwesome CSS для іконок.

Основні змінні, які керують станом:

- editMode – активний режим редагування;
- polygons – масив усіх зон (google.maps.Polygon) на мапі;
- editedPolygon, selectedPolygon, savedPath – поточна редагована/вибрана зона та її збережений шлях для скасування;
- map, infoWindow, drawingManager – об’єкти Google Maps;
- markers = new Map() – мапа маркерів з ключем rssi зі значенням google.maps.Marker;
- selectedServer – вибраний сервер (URL) з випадаючого списку;
- animatedMarker – маркер, який підстрибує при фокусі;
- config – налаштування таймінгів:
    - coords_retrive_timeout (сек) – як часто опитувати сервер за координатами;
    - inactive_marker_timeout (хв) – коли вважати маркер неактивним;
    - hide_marker_timeout (хв) – коли приховувати маркер з мапи;
- server_user = { user_name, pwd } – введені логін/пароль (після успішного login).

Головна ініціалізація:

1. initMap() – створює мапу, створює infoWindow.
2. DrawingTools() – налаштовує drawingManager (малювання полігонів).
3. MapListener() – клік по мапі, щоб закривати infoWindow і контролювати редагування.
4. Відновлює список серверів з localStorage:

selectedServer = Server.restoreServerList(...)

5. Читає налаштування config з localStorage ("adg-config"), якщо вони є.
6. Очищає таблиці: radios, zones, alarms.

7. Якщо сервер уже був вибраний раніше:
    - loadZones() – завантажує зони з /zones і малює полігони
    - loadCoords() – стартує періодичне опитування сервера за координатами
8. setEditorMode(false) – за замовчуванням запускається у View Mode.
9. Підв’язує обробники:
    - кнопка «mode» перемикає edit/view: setEditorMode(!editMode)
    - кнопка «login» викликає login()
    - у DOM пишеться version
    - підв’язується пошук по таблицях (input .searchInput фільтрує рядки)
10. Оновлює статусбар таблиці alarms.

Періодичне завантаження координат (polling)

loadCoords()

Функція робить самопланування через setTimeout кожні config.coords_retrive_timeout секунд викликає сама себе. Потім перевіряє умови:

- якщо editMode == true, вона виходить (не опитує сервер під час редагування).
- якщо selectedServer не вибраний, виходить.

Інакше:

- робить requestServer("", "GET", "") (тобто GET на базовий URL сервера),
- якщо відповідь без errorId, передає масив у showMarkers(response),
- якщо помилка, оновлює статусбар таблиці radios «No response».

Завантаження зон: loadZones() робить GET /zones і якщо ок, викликає showZones(response).

Відображення:

- showZones(data) для кожної зони бере zone_name, zone_gssi та zone_polygon
- перетворює координати в масив {lat, lng}
- створює google.maps.Polygon(...) зі стилями
- дописує в об’єкт полігона: zone.zone_name, zone.zone_gssi
- додає полігон на мапу, підключає PolygonListener(zone)
- додає запис у таблицю зон через Tab.addTableRow(...)
- оновлює статусбар зон.

Створення зон відбувається через DrawingManager: DrawingTools() налаштовує drawingManager для малювання полігонів (тільки polygon).

DrawingCompletionListener() слухає polygoncomplete:

1. Якщо полігон має менше 3 точок, видаляє його.
2. Показує Dialog.showPrompt(...) для введення:
    - zone_name (тільки букви та цифри, 3-9 знаків)
    - zone_gssi (тільки цифри, 3-9 знаків)
3. Якщо Cancel/Close, прибирає полігон.
4. Інакше викликає addZone(polygon): формує JSON з координатами (getCoordinates), POST /zones
5. Якщо сервер повернув не ok, прибирає полігон і показує помилку.
6. Якщо ok:

- робить полігон не редагованим, не перетягуваним
- додає у polygons
- додає у таблицю Zones.

Редагування/видалення існуючої зони відбувається наступним чином:

При кліку по полігону (PolygonListener) відкривається infoWindow з HTML, який генерує GetMessage(polygon). У Edit Mode там є кнопки:

- Edit: editPolygon() ховає drawing tools, робить полігон доступним для редагування та переміщення, зберігає старий шлях у savedPath
- Edit Properties: editProperties() дає змінити лише zone_gssi, потім PUT /zones
- Save: saveChanges() PUT /zones з новими координатами та вимикає editable/dragging, повертає drawing tools
- Cancel: cancelChanges() відновлює шлях із savedPath, вимикає редагування
- Delete: DeletPolygon() підтвердження, потім DELETE /zones, видаляє з масиву, з таблиці і з мапи.

3.3 Перевірка використання основних функцій системи

Перевірка використання основних функцій системи виконувалась у вигляді сценарного (функціонального) тестування Web UI, який взаємодіє з бекендом через REST-подібні HTTP endpoint-и. Метою перевірки було підтвердити виконання ключових функціональних вимог (FR1–FR7), а також базових нефункціональних вимог щодо стабільності роботи інтерфейсу під час типових дій диспетчера та адміністратора. Тестування проводилось у двох логічних режимах: View Mode (моніторинг) та Edit Mode (налаштування/CRUD), оскільки система навмисно розділяє операційні дані та операції редагування.

Першим етапом була перевірка підключення до сервера моніторингу. Користувач відкриває сторінку, після чого інтерфейс ініціалізує карту (Google Maps API), таблиці Radios, Zones, Alarms, а також зчитує локальні налаштування з localStorage (список серверів і параметри таймінгів). Далі користувач додає сервер у список або обирає вже збережений. Очікувана реакція: сервер відображається у вікні керування серверами, стає доступним у випадаючому списку, а після вибору UI виконує скидання стану та повторну ініціалізацію (завантаження зон і запуск моніторингу). У ході перевірки підтверджено, що перемикання між серверами коректно ізолює дані різних середовищ, а конфігурація зберігається локально і не вимагає повторного введення при наступному запуску сторінки.

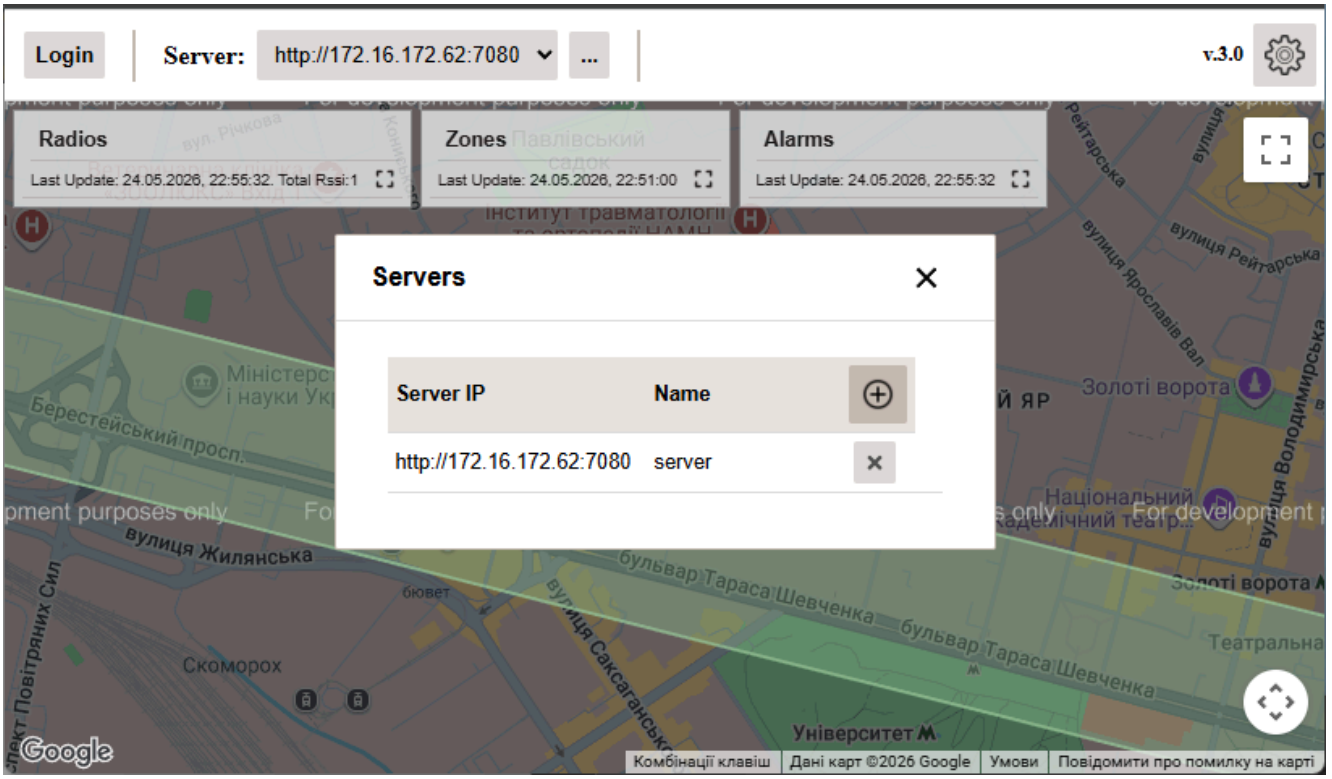

Рис. 3.3.1 Результат додавання сервера

Другим етапом була перевірка режиму моніторингу координат (FR1). Після вибору сервера запускається циклічне опитування endpoint-а GET / з інтервалом coords_retrive_timeout. У відповідь сервер повертає набір поточних координат об’єктів (RSSI) та супровідні атрибути (групу, час останнього оновлення). Очікувана реакція: на карті з’являються або оновлюються маркери об’єктів, а у таблиці Radios формується актуальний перелік з координатами та часом останнього оновлення. Додатково перевірявся сценарій фокусування на об’єкті через взаємодію з таблицею (центрування карти на потрібному маркері), що є частиною зручності диспетчерської роботи. Результат тесту підтвердив, що система забезпечує відображення координат у двох формах (карта і таблиця) та дозволяє оперативно знаходити потрібний об’єкт.

Третім етапом була перевірка механізму сповіщень для неактуальних координат (FR2). У системі використовується поріг неактивності inactive_marker_timeout, після якого об’єкт вважається таким, що давно не передавав дані. Очікувана реакція: об’єкт потрапляє в таблицю Alarms з

відповідним повідомленням, а його стан в UI стає однозначним для диспетчера (візуальне відображення проблеми з актуальністю). Також перевірено, що при відновленні надсилання координат alarms зникає або оновлюється відповідно до поточного стану. Окремо перевірявся випадок відсутності відповіді сервера (NFR2): якщо запит GET / не повертає коректну відповідь, інтерфейс відображає статус «No response» і продовжує спроби з заданою періодичністю, не блокуючи роботу сторінки.

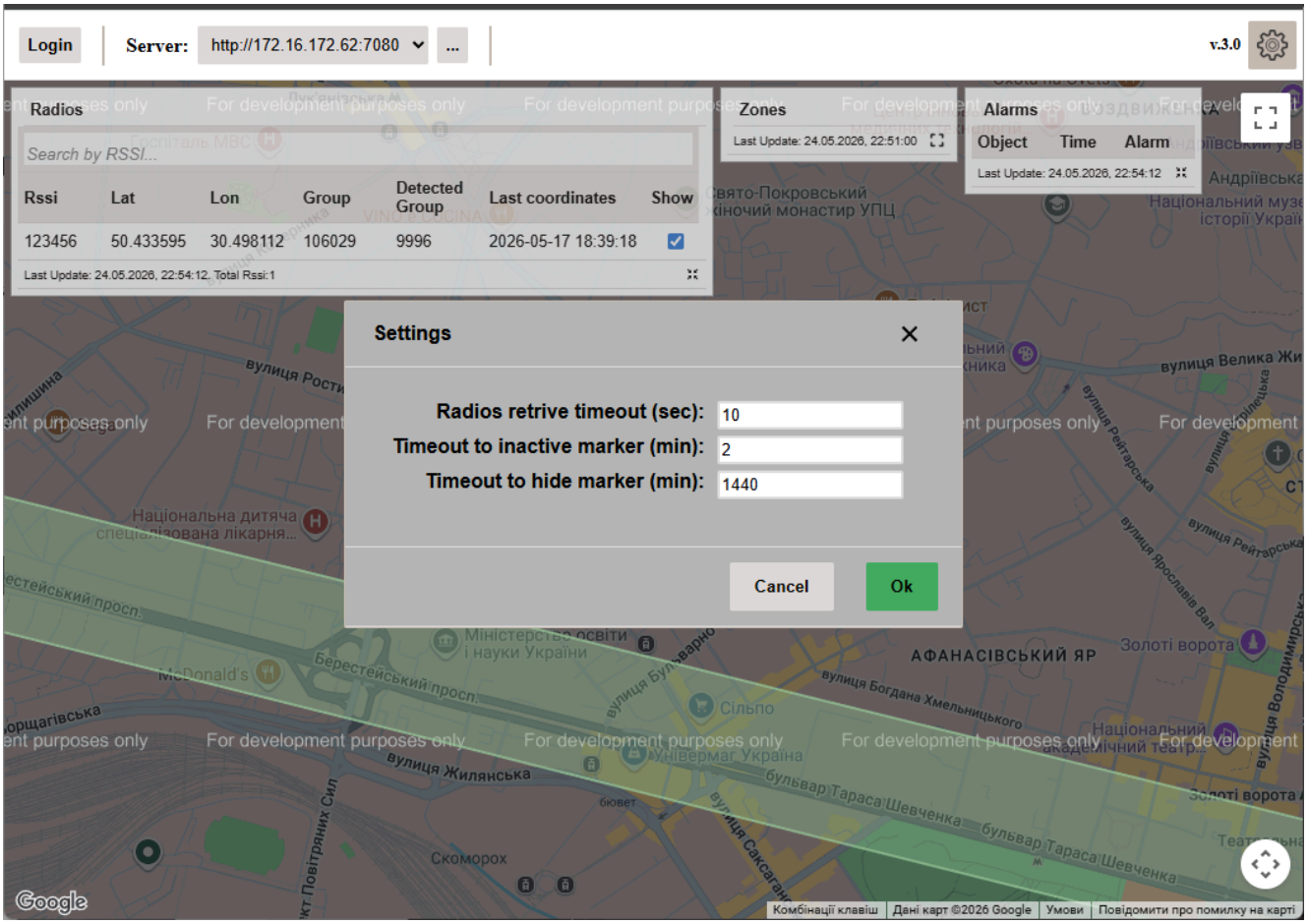

Рис. 3.3.2 Налаштування порогу неактивності

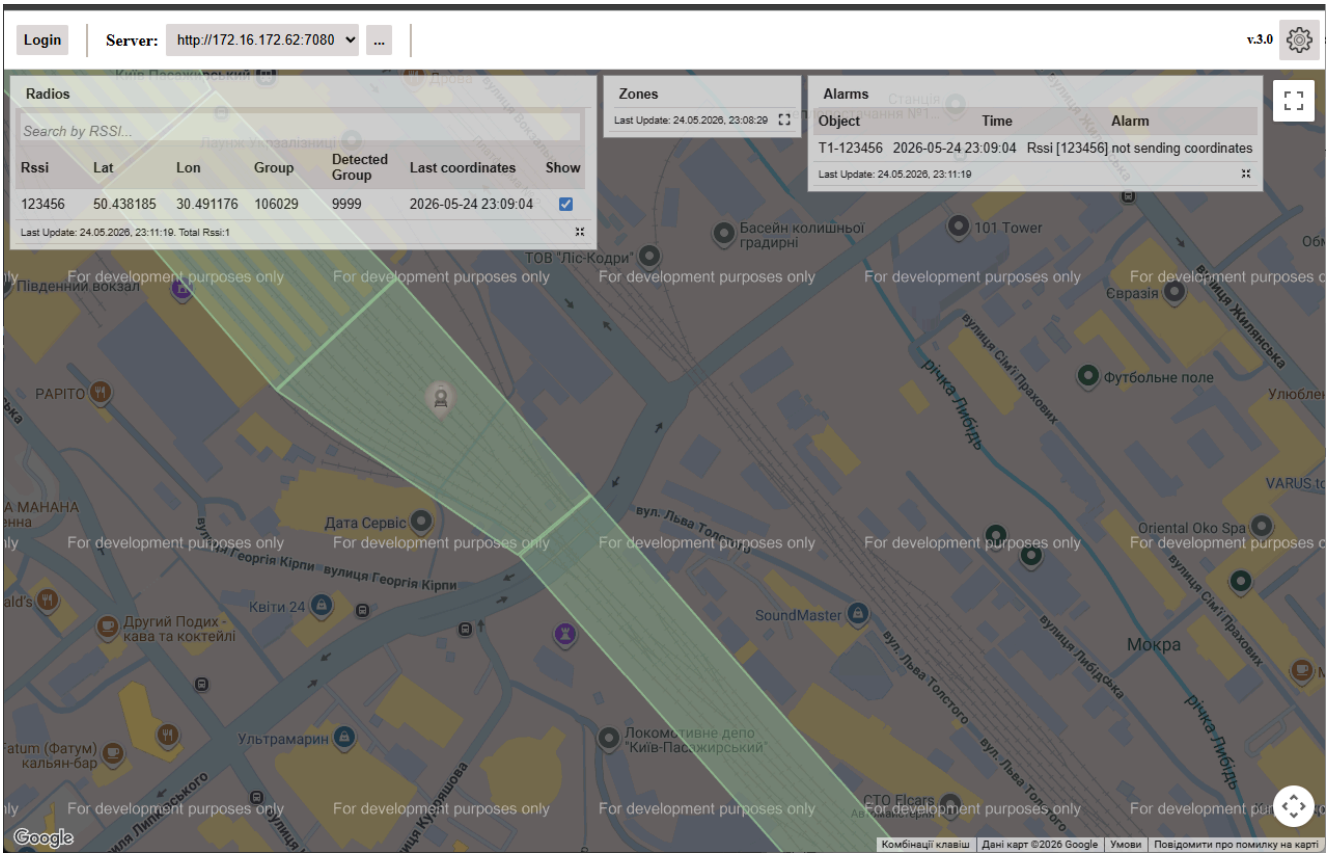

Рис. 3.3.3 Неактивний маркер та сповіщення в Alarms

Четвертий етап охоплював тестування керування геозонами у вигляді полігонів (FR3) і повного циклу CRUD-операцій. Перед початком редагування виконувалась перевірка авторизації (FR4): користувач у модальному вікні вводить логін/пароль, UI надсилає запит POST /login. Очікувана реакція: у разі успішної авторизації відкривається доступ до Edit Mode, а без авторизації редагування недоступне. Це підтвердило реалізацію розмежування доступу до керуючих функцій. У режимі Edit додатково перевірено, що polling координат припиняється, щоб уникнути конфліктів між оновленням операційних даних і редагуванням зон.

Далі перевірявся сценарій створення зони: користувач активує інструмент малювання (DrawingManager) і формує полігон на карті. Після завершення система запитує параметри зони (zone_name, zone_gssi) і виконує POST /zones з координатами полігону. Очікувана реакція: у разі коректних параметрів зона з’являється на карті, додається до таблиці Zones, стає доступною для перегляду і подальших операцій. Також перевірялась валідація введення (формат та довжина

zone_name, числовий формат zone_gssi, мінімум три точки полігону). Аналогічно тестувався сценарій редагування геометрії (переміщення вершин полігону) та збереження через PUT /zones, а також сценарій зміни властивостей (редагування лише zone_gssi). Для видалення зони перевірено наявність підтвердження (NFR3) і коректність виконання DELETE /zones із синхронним оновленням карти та таблиці.

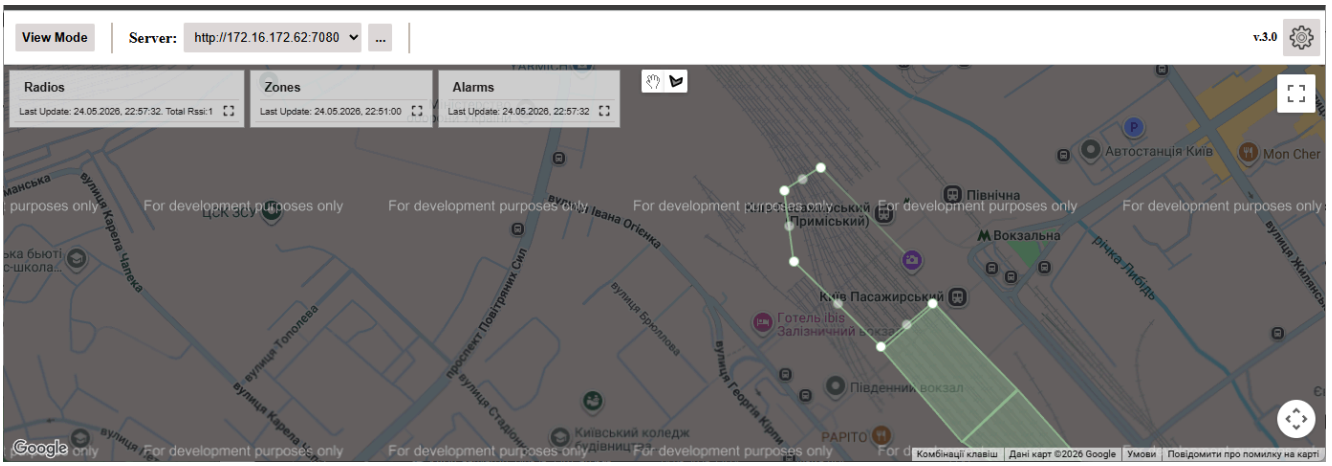

Рис. 3.3.4 Створення зони

Окремо оцінювалась логіка вимог FR6–FR7, що стосується коректної конфігурації зон (відсутність перетинів, наявність ідентифікаторів). На рівні UI підтверджено, що зона не може бути створена без заповнення ідентифікаторів, а також що інтерфейс демонструє параметри зони (назву, GSSI, координати) у вікні перегляду. Контроль перетинів зон у поточній реалізації розглядається як вимога до коректної конфігурації (організаційне правило), тому під час тестування перевірялось, що UI забезпечує зручне редагування і швидке виправлення конфігурації, якщо зона задана некоректно.

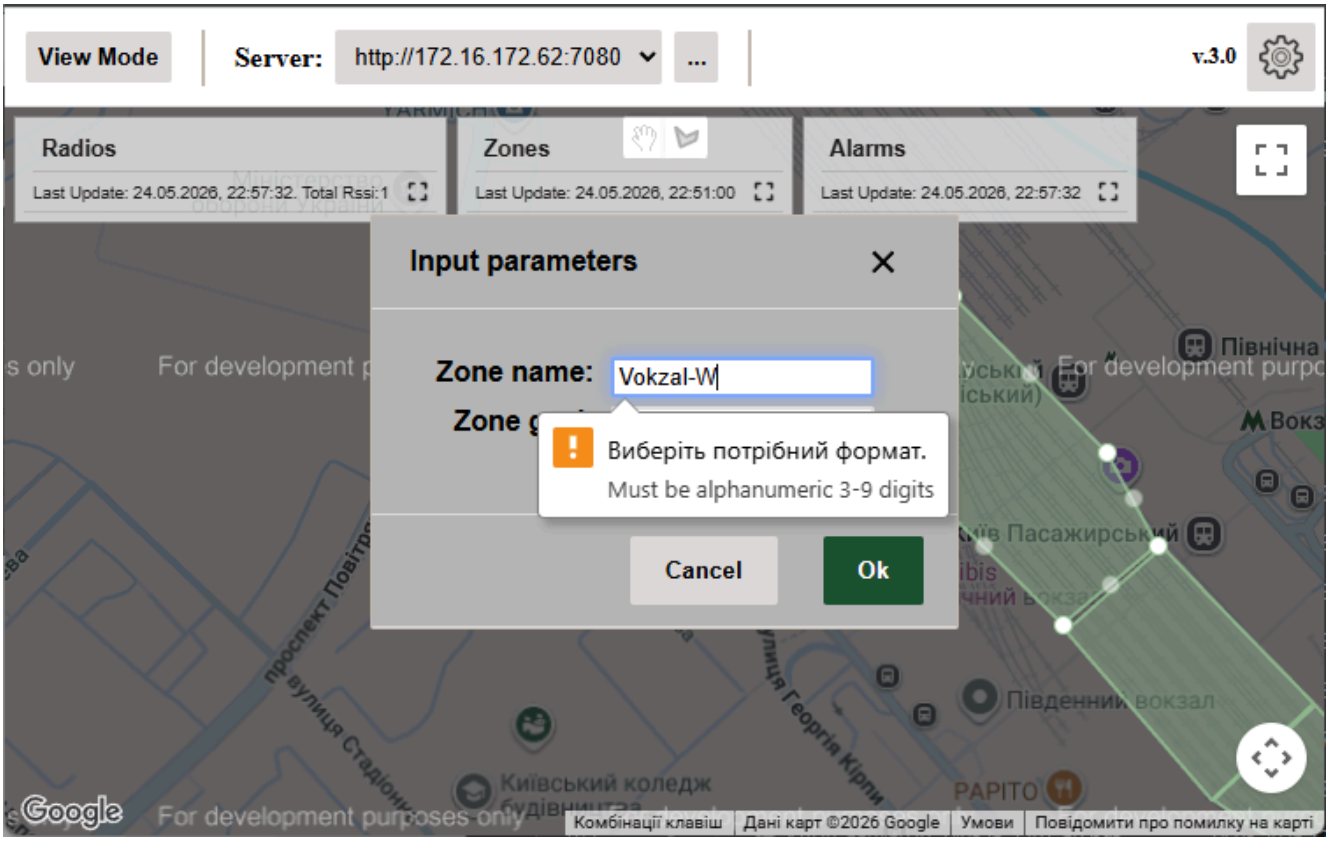

Рис. 3.3.5 Найменування зони

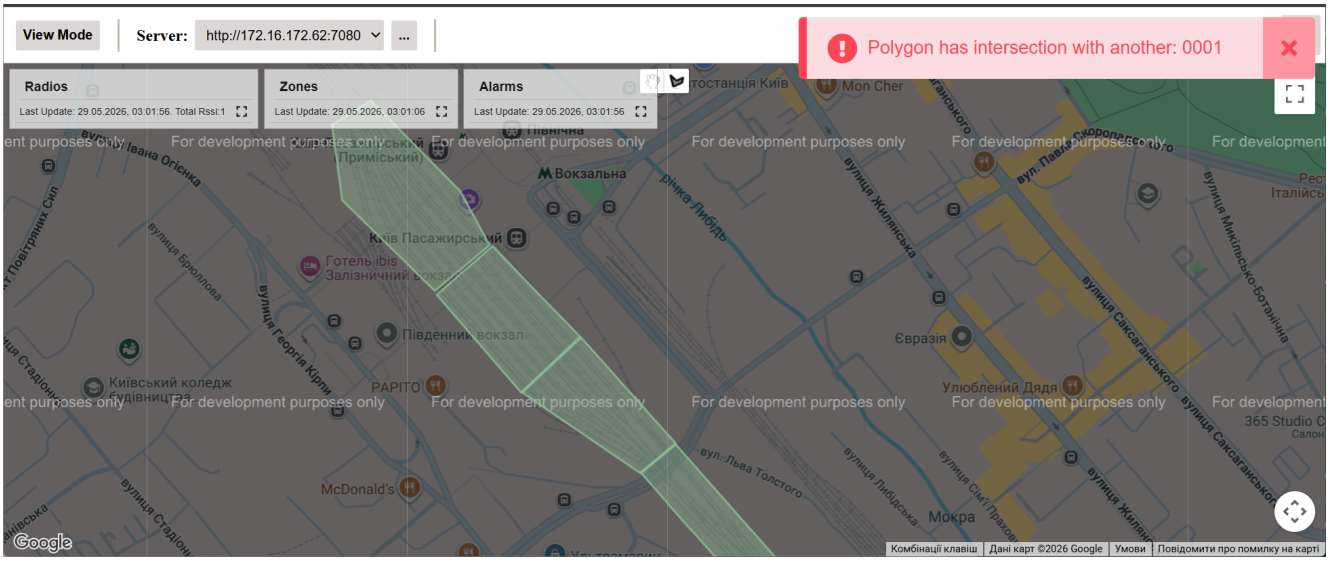

Рис. 3.3.6 Спроба створити зону, що перетинається з існуючою

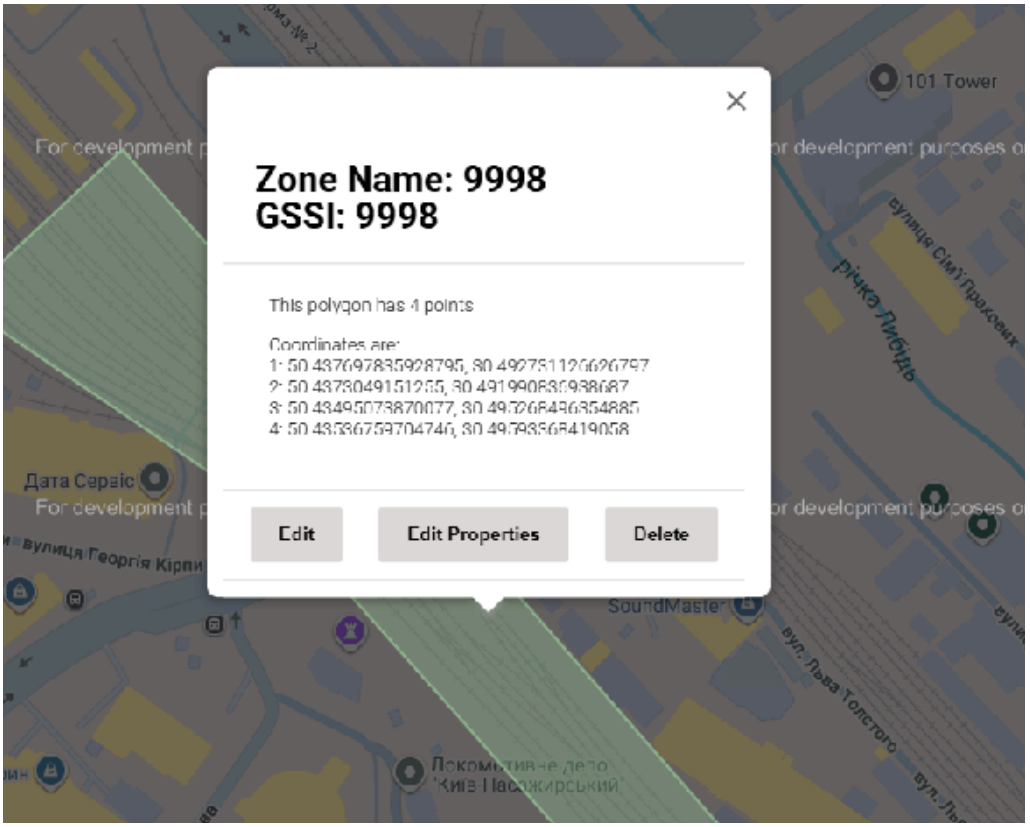

Рис. 3.3.7 Вікно редагування зони

Таким чином, сценарне тестування підтвердило працездатність основних функцій системи: відображення координат у реальному часі через polling, формування alarms при втраті актуальності даних, керування геозонами у вигляді полігонів з повним CRUD-циклом, а також розділення режимів моніторингу та редагування з обов’язковою авторизацією для керуючих операцій. Це доводить виконання базових функціональних вимог і готовність рішення до демонстраційного використання та подальшого розвитку.

3.4 Висновки до третього розділу

Отримана програма моніторингу рухомих об’єктів має практичну цінність як легкий диспетчерський веб-інструмент для моніторингу рухомих об’єктів на карті та оперативного керування просторовими даними (геозонами). Її основна користь полягає у забезпеченні операційної видимості і швидкого прийняття рішень у щоденній роботі диспетчера.

До практично значущих результатів можна віднести наступні перераховані нижче характеристики. Швидкий доступ до поточного місцезнаходження об’єктів. Відображення маркерів на карті та табличний перелік дозволяють диспетчеру швидко знаходити потрібний об’єкт і переходити до нього на карті.

Оперативне виявлення проблем із даними. Індикація неактивності та таблиця сповіщень дозволяють вчасно реагувати, якщо об’єкт перестав надсилати координати, що знижує ризик «сліпих зон» у моніторингу.

Керування геозонами як частиною логістичного довідника. Геозони (склади, хаби, точки доставки, заборонені зони) є ключовими для диспетчеризації. Можливість створювати та редагувати полігони без сторонніх GIS-інструментів підвищує гнучкість і зменшує час на підтримку актуальних меж зон.

Мультисерверна робота і локальні налаштування. Підтримка списку серверів і параметрів оновлення в локальному сховищі полегшує використання у кількох середовищах (тестове/продуктивне, різні регіони, різні контури).

Рішення UI може бути використано як легкий клієнт для конкретного бекенду, що особливо корисно для швидкої перевірки гіпотез і погодження вимог з користувачами.

Попри практичну корисність, поточна версія рішення має потенціал для доопрацювання, який важливо враховувати під час оцінки його застосовності у виробничому середовищі. Це можуть бути напрямками розвитку продукту в залежності від вимог та потреб цільового рішення.

Оновлення даних реалізовано через polling. Інтерфейс періодично опитує сервер (GET /), що є простим, але для високих навантажень і вимог в мінімальній

затримці можуть бути використані альтернативні методи, розглянуті в даній роботі, адже при великій кількості об’єктів або високій частоті оновлень polling може створювати зайве навантаження на сервер і мережу.

У наявному рішенні є значне поле для розвитку логістичної моделі. Зараз наявні дані зосереджені навколо об’єктів (RSSI) та геозон. У майбутньому можна впровадити сутності верхнього рівня, характерні для логістики: замовлення, рейс, маршрут, часові вікна, статус доставки, ETA, виявлення аномалій маршруту, аналіз причин затримок чи простоїв.

Поточна логіка орієнтована на поточний стан, але під вимоги замовника можна реалізувати історію переміщень (трек), порівняння план/факт маршруту, статистику простоїв, звітність KPI.

Безпека даного рішення також має потенціал для доопрацювання. Для продуктивного впровадження необхідна повноцінна модель авторизації, перехід на HTTPS, контроль доступу та аудит операцій.

Програма моніторингу рухомих об’єктів має високу практичну цінність як реалізація базового диспетчерського інтерфейсу, що поєднує моніторинг геопозиціювання та керування геозонами. Потенціал розвитку рішення пов’язаний з прототипним характером. Запропоновані напрямки розвитку дозволяють еволюціонувати рішення в повноцінну систему для логістики та диспетчеризації з моделлю на базі подій, оновленнями в реальному часі, масштабованим UI та інтеграціями з бізнес-системами.

ВИСНОВКИ

У процесі роботи розглянуто та вирішено актуальну задачу розробки програмного забезпечення для моніторингу рухомих об’єктів на основі геоінформаційних даних, що є важливою складовою сучасних диспетчерських і логістичних систем. Актуальність теми зумовлена зростанням потреби в оперативному контролі місцезнаходження транспорту, техніки або інших мобільних пристроїв, зменшенні простоїв та затримок, підвищенні прозорості процесів і швидкості реагування на інциденти. В умовах, коли значна частина управлінських рішень прив’язана до географічного контексту, ключову роль відіграють не лише «точки на карті», а й інструменти зонування, які дозволяють формалізувати події та автоматизувати реакції на переміщення об’єктів.

У ході виконання роботи було проаналізовано предметну область геомоніторингу, типові архітектурні схеми отримання, передачі та обробки координат, а також визначено основні ризики та обмеження. На підставі аналізу сформульовано функціональні та нефункціональні вимоги до системи, серед яких ключовими є відображення координат на карті й у таблиці, підтримка геозон у вигляді полігонів із повним циклом CRUD-операцій, формування сповіщень щодо неактуальних даних, а також розділення режимів моніторингу та налаштування з обов’язковою авторизацією для керуючих операцій.

Розроблене рішення реалізовано як веб-застосунок на JavaScript з використанням Google Maps JavaScript API для картографічної візуалізації та інтерактивного редагування геозон. Взаємодія з серверною частиною здійснюється через REST-подібний HTTP API, а отримання координат реалізовано методом polling як компроміс між простотою реалізації та достатньою актуальністю даних для диспетчерського сценарію. Інтеграційний шар побудовано з використанням Apache NiFi, що забезпечує можливість потокової обробки координат, інтеграції з зовнішніми контурами та виконання керуючих дій.

Проведене сценарне тестування підтвердило працездатність ключових функцій: система коректно відображає об’єкти на карті, формує список у

табличному вигляді, створює alarms при втраті актуальності координат, а також забезпечує керування геозонами у форматі полігонів із створенням, редагуванням та видаленням. Отриманий програмний продукт має практичну цінність як легкий диспетчерський інструмент, який може використовуватися для демонстраційних потреб, погодження вимог та як основа для подальшого розвитку.

Таким чином, поставлену мету роботи досягнуто: спроєктовано та реалізовано програмне рішення для моніторингу рухомих об’єктів на основі геоінформаційних даних, яке забезпечує базовий функціонал диспетчерського контролю, геозонування та індикації проблем із актуальністю даних, і може слугувати основою для подальшого розвитку до повноцінної виробничої системи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. https://www.samsara.com/products/telematics/gps-fleet-tracking/
2. https://www.grandviewresearch.com/industry-analysis/internet-of-things-iot-fleet-management-market
3. https://www.geotab.com/products/fleet-management-software/
4. Potvin J., Xu Y., Benyahia I.: Vehicle routing and scheduling with dynamic travel times, Computers & Operations Research Volume 33, Issue 4, 2006, ст. 1129-1137
5. Hu T., Liao T., Lu Y.: Study of Solution Approach for Dynamic Vehicle Routing Problems with Real-Time Information, Journal of the Transportation Research Board, 2003
6. Pan W., Liu S.: Deep reinforcement learning for the dynamic and uncertain vehicle routing problem, Applied Intelligence 2022, ст. 405-422
7. https://www.traccar.org/documentation/geofences/
8. Oliveira R., Cardoso I., Barbosa J., da Costa C., Prado M.: An intelligent model for logistics management based on geofencing algorithms and RFID technology, Expert Systems with Applications, 2015
9. Prosper J.: Real-Time Tracking and Alerts Using Geofencing in Logistics and Fleet Management, 2024
10. https://reveal-help.verizonconnect.com/hc/en-us/articles/14657641103379-Create-a-geofence
11. https://www.quartix.com/vehicle-tracking/alerts/
12. https://www.navixy.com/docs/user/guide/events-and-notifications/movement-monitoring/geofence-entrance-or-exit/
13. https://www.abax.com/en-gb/features/geofence
14. https://help.wialon.com/en/wialon-platform/user-guide/geofences
15. https://www.geotab.com/platform-overview/
16. https://www.samsara.com/products/platform

17. https://www.eway.in.ua/

18. Graser, A., Naghibzadeh-Jalali, A., Lampert, J., Weißenfeld, A., & Janowicz, K. (2024). MobilityDL: A Review of Deep Learning From Trajectory Data. a, GeoInformatica, 2024, 33 ст.

19. Ma M., Bartocci E., Lifland E., Stankovic J., Feng L.: A Novel Spatial-Temporal Specification-Based Monitoring System for Smart Cities, Cornell University, 2021, 17 ст.

20. Bakirci M.: Vehicular mobility monitoring using remote sensing and deep learning on a UAV-based mobile computing platform, Measurement, 2025

21. Wang B., Sui H., Ma G., Zhou Y.: MCTracker: Satellite video multi-object tracking considering inter-frame motion correlation and multi-scale cascaded feature enhancement, ISPRS Journal of Photogrammetry and Remote Sensing, 2024, ст. 82-103

22. Precise positioning with current multi-constellation Global Navigation Satellite Systems: GPS, GLONASS, Galileo and BeiDou, https://www.nature.com/articles/srep08328

23. https://mqtt.org/mqtt-specification/

24. https://www.traccar.org/traccar-api/

25. XMLHttpRequest https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest

26. ASGARD: The ultimate response to maritime spoofing attacks, 2023 https://www.euspa.europa.eu/newsroom-events/news/asgard-ultimate-response-maritime-spoofing-attacks

27. EASA updates SIB on GNSS Outage and Alterations. 2023 https://www.easa.europa.eu/en/newsroom-and-events/news/easa-updates-sib-gnss-outage-and-alterations

28. Employer Vehicle Tracking https://dataprotection.ie/en/dpc-guidance/employer-vehicle-tracking

29. Картер Д. Підручник з Apache NiFi, 2025 https://www.guru99.com/uk/apache-nifi-tutorial.html

30. https://en.wikipedia.org/wiki/Apache_NiFi

31. WebSockets explained: What they are and how they work, 2025 https://ably.com/topic/websockets

32. https://mqtt.org/

33. Події, надіслані сервером (SSE) https://fastapi.tiangolo.com/uk/tutorial/server-sent-events/

34. https://leafletjs.com/

35. Використання OpenLayers https://switch2.openstreetmap.org.ua/using-tiles/getting-started-with-openlayers/

36. Mapbox GL JS https://docs.mapbox.com/mapbox-gl-js/guides/

37. Що таке Vanilla JS і коли використовується https://foxminded.ua/vanilla-js/

38. JavaScript https://developer.mozilla.org/en-US/docs/Web/JavaScript

39. Порівнюємо React, Angular і Vue — найпопулярніші бібліотеки й фреймворки у 2022 році https://dou.ua/forums/topic/39933/

40. Angular проти React https://foxminded.ua/angular-vs-react/

41. REST API: принципи роботи веб-сервісів https://wezom.com.ua/ua/blog/rest-api-printsipi-roboti-veb-servisiv

42. Що таке GraphQL і як з ним працювати? https://foxminded.ua/graphql-shcho-tse/

43. Що таке REST API та його можливості https://iptel.ua/blog/article/rest-api