БАКАЛАВРСЬКА РОБОТА

БР.КІ-04.00.00.000 ПЗ

Група КІ-22-1

Гавриляк Ілля

2026

Івано-Франківський національний технічний університет нафти і газу
Факультет інформаційних технологій
Кафедра комп'ютерних систем і мереж

Гавриляк Ілля Васильович

УДК 004.4

БАКАЛАВРСЬКА РОБОТА

Розробка web-додатку інформаційної підтримки сервісу охоронних систем засобами C#, ASP.NET Core, Angular, Cosmos DB

Комп'ютерна інженерія
(назва освітньої програми)

123 - Комп'ютерна інженерія
(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня _____ *Гавриляк І.В.* _____
(підпис, ініціали та прізвище здобувача)

Науковий керівник _____ *Гарасимів Т. Г., асист.* _____
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри КСМ

д.т.н.,професор _____ *С.І. Мельничук*
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2026 рік

Івано-Франківський національний технічний університет нафти і газу

(повне найменування вищого навчального закладу)

Факультет *інформаційних технологій*

Кафедра *комп'ютерних систем і мереж*

Освітній ступінь *бакалавр*

Спеціальність *123 – Комп'ютерна інженерія*

(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри КСМ

(С.І. Мельничук)

«*21*» *травня* *2026* року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Гавриляку Іллі Васильовичу

(прізвище, ім'я, по батькові)

Тема проекту (роботи*)* *Розробка web-додатку інформаційної підтримки сервісу охоронних систем засобами C#, ASP.NET Core, Angular, Cosmos DB*

керівник проекту (роботи) *Гарасимів Т. Г., асист.*

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

затверджені наказом вищого навчального закладу від *21.04.2026 № 180/7*

2. Строк подання студентом роботи *11 червня 2026 р*

3. Вихідні дані до роботи *Матеріали і результати отримані під час проходження переддипломної практики, методичні вказівки, технічна література.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) *1. Аналіз предметної області та постановка завдання. 2. Проектування архітектури та структури даних веб-додатка. 3. Практична реалізація та тестування веб-додатка.*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти розділів роботи

Розділ	Консультант	Підпис, дата

7. Дата видачі завдання 02 *лютого 2026 р.*

№ з/п	Назва етапів дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	*Аналіз предметної області та постановка завдання*	*Лютий, 2025*	
2	*Проектування архітектури та структури даних веб-додатка*	*Березень, 2025*	
3	*Практична реалізація та тестування веб-додатка*	*Травень, 2025*	
4	*Оформлення пояснювальної записки*	*Червень, 2025*	

Студент _____ *Гавриляк І.В.*
(підпис) (прізвище та ініціали)

Керівник роботи_____ *Гарасимів Т. Г.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

У кваліфікаційній роботі виконано проектування та програмну реалізацію веб-додатка сервісного обслуговування комплексів охорони на базі платформи ASP.NET Core та фреймворку Angular. Аналітичний етап дослідження присвячено аналізу процесів інсталяції, технічної підтримки та регламентного супроводження різнотипових систем захисту. Проведено систематизацію нормативних і правових вимог до організації обліку обладнання на об'єктах нерухомості. Виявлено ключові недоліки наявного програмного забезпечення у сфері планування діяльності технічного персоналу.

Наступний етап містить опис архітектурних рішень із побудови системи на основі класичної тришарової структури. Спроектовано схему розподілу даних для локального емулятора Azure Cosmos DB з використанням документ-орієнтованого API. Оптимізовано логіку збереження просторових характеристик об'єктів шляхом формування масивів географічних координат. Розроблено алгоритм планування періодичних інспекцій на основі часових інтервалів. Визначено структуру взаємодії між компонентами бекенду та фронтенду через HTTP REST інтерфейси.

Практична частина роботи охоплює створення серверного програмного коду мовою C# та побудову інтерфейсу користувача. Реалізовано контролери, сервіси та репозиторії для обробки сутностей користувачів, ролей, нотаток та інспекцій. Розроблено модуль для відображення меж охоронюваних об'єктів на інтерактивних картах OpenStreetMap.

Ключові слова: веб-додаток, asp.net core, angular, cosmos db, охоронні системи, openstreetmap, технічна інспекція.

SUMMARY

This thesis involves the design and software implementation of a web application for the maintenance of security systems based on the ASP.NET Core platform and the Angular framework. The analytical phase of the study focuses on analyzing the processes of installation, technical support, and routine maintenance of various types of security systems. The regulatory and legal requirements for organizing equipment accounting at real estate facilities were systematized. Key shortcomings of existing software in the area of technical staff activity planning were identified.

The second phase contains a description of architectural solutions for building the system based on a classic three-tier structure. A data distribution scheme was designed for a local Azure Cosmos DB emulator using a document-oriented API. The logic for storing spatial characteristics of objects was optimized by forming arrays of geographic coordinates. An algorithm for scheduling periodic inspections based on time intervals was developed. The structure of interaction between backend and frontend components via HTTP REST interfaces was defined.

The practical part of the work covers the creation of server-side code in C# and the development of a user interface. Controllers, services, and repositories were implemented to handle user entities, roles, notes, and inspections. Modules have been developed to display the boundaries of protected objects on interactive OpenStreetMap maps.

Keywords: web application, asp.net core, angular, cosmos db, security systems, openstreetmap, technical inspection.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ…….…… 4

ВСТУП…… 5

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ.....… 7

1.1 Опис предметної області технічного обслуговування та впровадження систем безпеки…….…….…… 7

1.2 Порівняльний огляд існуючих програмних рішень..…….……. 10

1.3 Постановака завдання……..…….……. 15

2 ПРОЕКТУВАННЯ АРХІТЕКТУРИ ТА СТРУКТУРИ ДАНИХ ВЕБ-ДОДАТКА…….……..... 19

2.1 Функціональні сценарії взаємодії та системна декомпозиція модулів веб-додатка……..……... 19

2.2 Структурна організація та оптимізація сховища NoSQL……. 24

2.3 Побудова діаграми послідовності для моделювання асинхронних процесів гео-моніторингу……..…….. 27

2.4 Розробка алгоритму логічного видалення веб-додатку……... 30

2.5 Обґрунтування вибору програмного інструментарію та системної архітектури……..……... 33

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБ-ДОДАТКА……. 37

3.1 Реалізація серверної частини веб-додатка на базі ASP.NET Core…….. 37

3.2 Розробка клієнтської частини веб-додатка на базі Angular..……... 47

3.3 Тестування та верифікація функціональних модулів веб-додатка……. 53

ВИСНОВКИ.……..…… 65

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ…..…….. 67

ДОДАТКИ…….. 69

Бібліографічна довідка

					БР.КІ-04.00.00.000 ПЗ		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Гавриляк І.В.			*Розробка web-додатку інформаційної підтримки сервісу охоронних систем засобами C#, ASP.NET Core, Angular, Cosmos DB*	Літ.	Арк. 3 / Акрушів 68
Перевір.		Гарасимів Т.Г.					
Реценз.		Гарасимів В.М.				ІФНТУНГ КІ-22-1	
Н. контр.		Лазорів А.М.					
Затверд.		Мельничук С.					

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

СКД – система контролю доступу

SPA (Single Page Application) – односторінковий застосунок

CRM (Customer Relationship Management) – управління відносинами з клієнтами

FSM (Field Service Management) – управління виїзним обслуговуванням

GUID (Globally Unique Identifier) – глобальний унікальний ідентифікатор

OSM (OpenStreetMap) – відкрита карта світу

WGS 84 (World Geodetic System 1984) – всесвітня геодезична система 1984 року

JWT (JSON Web Token) – веб-токен у форматі JSON

DTO (Data Transfer Object) – об'єкт перенесення даних

REO (Real Estate Object) – об’єкт нерухомості

SDK (Software Development Kit) – набір засобів для розробки

DI (Dependency Injection) – впровадження залежностей

ВСТУП

Актуальність теми. Забезпечення надійності комплексів інженерно-технічної безпеки безпосередньо пов'язане з якістю їх монтажу, підтримки та регулярного сервісного супроводу. Сучасні підприємства індустрії безпеки стикаються з необхідністю централізованого обліку великої кількості об'єктів нерухомості, кожен з яких має унікальні просторові межі та специфічний набір встановленого обладнання. Використання паперових носіїв або децентралізованих табличних процесорів для планування регламентних перевірок призводить до помилок та порушення термінів інспекцій. Існуючі універсальні системи керування базами даних часто вимагають значних обчислювальних ресурсів для збереження просторових геоданих. Створення веб-додатків на основі NoSQL-рішень дозволяє оптимізувати процеси структурування інформації без надмірного ускладнення схем сховища. Застосування технологічного стеку у складі платформи ASP.NET Core, фреймворку Angular та локального емулятора бази даних Cosmos DB забезпечує розробку гнучких веб-інтерфейсів для внутрішнього персоналу сервісних служб.

Об'єкт дослідження – процес технічного обслуговування, інсталяції та регламентного контролю різнотипових систем захисту об'єктів.

Предмет дослідження – методи об'єктно-орієнтованого проектування, NoSQL-моделі даних та веб-технології в процесі автоматизації обліку та планування періодичних інспекцій.

Мета роботи – розробка архітектури та програмна реалізація кросплатформеного веб-додатка для оптимізації процесів інженерного супроводження та ведення обліку технічного стану засобів безпеки.

Для досягнення мети визначено такі завдання:

1. Проаналізувати чинні процеси технічної підтримки та монтажу систем захисту для формування вимог до веб-додатка.

2. Спроектувати структуру документ-орієнтованої бази даних CosmosDB з урахуванням масивів географічних координат об'єктів.

3. Розробити серверну логіку на базі тришарової архітектури з використанням мови C# та платформи ASP.NET Core.

4. Реалізувати користувацький інтерфейс системи засобами фреймворку Angular із інтеграцією карт OpenStreetMap.

5. Провести верифікацію та тестування розроблених програмних модулів у середовищі локального емулятора СУБД.

Методи дослідження. У роботі застосовано системний аналіз для дослідження предметної області та формалізації вимог до веб-сервісу. Проектування архітектури виконано на основі принципів об'єктно-орієнтованого програмування та патерну розділення відповідальності. Організацію сховища реалізовано за допомогою методів документної денормалізації NoSQL. Перевірку працездатності системи здійснено шляхом функціонального тестування компонентів бекенду та фронтенду.

Практичне значення отриманих результатів полягає у створенні спеціалізованого програмного комплексу для внутрішнього персоналу сервісних компаній. Веб-додаток забезпечує ведення єдиного реєстру об'єктів охорони, візуалізацію їхніх меж на інтерактивній карті та автоматичний контроль термінів проведення планових інспекцій. Розроблена архітектура дозволяє адаптувати систему під обслуговування нових типів захисного обладнання без модифікації базового коду.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Опис предметної області технічного обслуговування та впровадження систем безпеки

Функціонування сучасних підприємств у сфері інженерного захисту базується на забезпеченні повного життєвого циклу технічних засобів безпеки. Цей процес починається з первинного обстеження локацій, формування проектної документації та виконання монтажних робіт на об'єктах нерухомості. У реальних умовах інженерний бізнес стикається з необхідністю супроводження значної кількості територіально розподілених точок, що потребує чіткої координації виїзного персоналу [21]. Основним чинником, який ускладнює адміністрування таких компаній, є високий рівень гетерогенності використовуваного обладнання. На одному об'єкті одночасно функціонують цифрові та аналогові камери відеоспостереження, виконавчі механізми систем контролю доступу, шлагбауми з різними типами приводів та сповіщувачі руху. Кожен апаратний елемент володіє унікальними експлуатаційними параметрами, які необхідно враховувати при плануванні сервісних виїздів [25].

Традиційна практика менеджменту в індустрії безпеки вимагає ведення безперервного технічного обліку. Кожен об'єкт охорони володіє набором статичних характеристик, до яких належать адреса, назва та внутрішній ідентифікатор контракту. Проте у реальному світі діяльність сервісних служб виходить за межі простого збереження текстових адрес. Критичною проблемою для диспетчерів та виїзних інженерів є точне визначення меж зони технічної відповідальності. Об'єкт не є абстрактною точкою на папері. На практиці виникає потреба фіксації ліній проходження зовнішнього периметра, зон огляду оптичних пристроїв та меж земельних ділянок.

Рішення цієї проблеми лежить у площині просторового моделювання. Для забезпечення прозорості сервісних процесів компанії потребують інструментів фіксації географічних координат, що формують замкнені багатокутники. Такий

підхід дозволяє чітко розмежувати зони обслуговування між різними бригадами. Він повністю виключає суперечки при виникненні аварійних ситуацій на стиках територій. Зберігання масивів послідовних точок широти й довготи вимагає впровадження гнучких моделей даних, здатних динамічно адаптуватися до геометричних параметрів конкретного інженерного комплексу. Статичний опис без прив'язки до інтерактивного координатного середовища не здатний задовольнити вимоги сучасного ринку послуг безпеки.

Повсякденна діяльність сервісного підприємства підпорядкована суворому регламенту планового та аварійного нагляду за комплексами захисту. Регулярний огляд пристроїв мінімізує ризики виникнення апаратних збоїв. У реальних умовах гнучкість планування є головним інструментом оптимізації витрат робочого часу. Різні типи систем мають індивідуальну інтенсивність зносу вузлів. Через це графік виїздів інженерів будується на основі варіативних інтервалів, які охоплюють квартальні, піврічні або річні періоди. Автоматизація розрахунку наступного візиту дозволяє компанії уникнути накладання графіків та раціонально розподілити навантаження на технічний персонал.

Процес безпосереднього виконання інспекцій генерує значні масиви службової інформації. Співробітник на об'єкті фіксує результати перевірки, формуючи структуровані текстові звіти та нотатки. Ця документація відображає поточний стан ліній зв'язку, виконані заходи з чищення оптичних елементів, налаштування програмних параметрів, а також фіксує дефекти обладнання. Збереження ретроспективних даних дозволяє керівництву проводити детальний аудит надійності використовуваних технічних засобів. Системний аналіз накопичених логів допомагає виявити серійні дефекти конкретних виробників апаратури.

Особливе місце в інженерному бізнесі посідає дотримання юридичної та технічної цілісності баз даних. Діяльність підприємств із монтажу та обслуговування систем безпеки чітко регулюється чинним законодавством України. Базовим нормативно-правовим актом у цій сфері виступає Закон України «Про охоронну діяльність» [1]. Він визначає правові межі використання

технічних засобів та правила доступу інженерного персоналу до об'єктів інфраструктури. Проектування програмного забезпечення для автоматизації таких процесів має додатково спиратися на положення ДСТУ IEC 60839 [3]. Цей стандарт встановлює жорсткі вимоги до електронних систем контролю та сигналізації. Будь-яка цифровізація інженерних сервісів неминуче пов'язана з обробкою конфіденційної інформації. Сюди належать просторові координати об'єктів, схеми розташування камер, плани приміщень та графіки чергувань. З цієї причини розробник зобов'язаний інтегрувати в архітектуру веб-додатка інструменти суворого розмежування прав доступу.

Другим юридичним вектором проектування є захист інформаційних масивів. Збирання просторових даних та фіксація робочих метрик персоналу підпадає під дію Закону України «Про захист персональних даних» [2]. Стаття 24 цього закону покладає на власників баз даних прямий обов'язок щодо забезпечення технічного захисту відомостей на всіх етапах їх життєвого циклу [5]. Записи отримують статус доказової бази під час проведення внутрішніх службових розслідувань або судових експертиз у разі компрометації контуру безпеки підприємства. Національні державні будівельні норми, зокрема державні будівельні норми вимагають безперервного протоколювання стану інженерних систем захисту [4]. Фіксація відмов апаратури у цифрових журналах є обов'язковою процедурою. Програмне ігнорування цих вимог тягне за собою юридичну відповідальність для оператора системи.

Також безповоротне видалення інформації про контракти чи проведені ремонти є неприпустимим через ризик руйнування історії обслуговування. Регламент деактивації застарілих або знятих з обліку об'єктів базується на логічному приховуванні записів за допомогою принципу м'якого видалення. Сутність не вилучається з бази даних фізично, а лише маркується відповідним статусом деактивації.

Процедура переведення запису в архівний стан вимагає обов'язкової фіксації причин з боку оператора. Бізнес-процес передбачає ручне введення детального текстового обґрунтування деактивації, що дозволяє контролювати

правомірність дій персоналу та запобігає навмисному викривленню звітності. При цьому архітектурні межі внутрішніх систем чітко ізолюються від безпосереднього приймання апаратних сигналів телеметрії. Програмний комплекс виконує виключно функції обліку, планування та менеджменту діяльності підрозділів. Він не функціонує як пульт централізованого спостереження, що дозволяє уникнути надмірних витрат на обчислювальну інфраструктуру та фокусує роботу виключно на сервісному контурі компанії.

1.2 Порівняльний огляд існуючих програмних рішень

Організація діяльності сервісних підрозділів вимагає використання інструментів автоматизації для ведення технічного обліку. На багатьох підприємствах досі зберігається практика децентралізованого адміністрування. Такий підхід базується на застосуванні традиційних паперових журналів або загальних табличних процесорів. Аналіз повсякденної діяльності інженерних служб дозволяє виділити критичні обмеження подібних засобів.

Першим рівнем організації обліку є паперові носії інформації. Фіксація відомостей про встановлене обладнання та адреси об'єктів у паперових журналах супроводжується такими системними недоліками:

– повна відсутність можливостей оперативного пошуку, сортування та швидкої фільтрації технічних даних;

– високий ризик фізичного зносу, втрати або випадкового знищення журналів під час виїзних інспекцій;

– неможливість забезпечення синхронізації відомостей між диспетчерським центром та територіально розподіленими бригадами інженерів;

– регулярне виникнення пропусків термінів планового обслуговування пристроїв безпеки через вплив людського фактора.

Більш прогресивним етапом розвитку є впровадження загальних табличних процесорів. Найчастіше для цих цілей підприємства використовують локальні файли Microsoft Excel або хмарні сервіси Google Таблиць. Проте автоматизація

бізнес-процесів за допомогою електронних таблиць має значні архітектурні обмеження. Головна проблема полягає у відсутності механізмів жорсткого контролю цілісності зв'язків між сутностями. Оператор може помилково видалити унікальний ідентифікатор відповідального співробітника. Також існує ризик випадкового порушення структури часових інтервалів планових перевірок. Електронні таблиці не здатні забезпечити валідацію введених даних при описі обладнання. Приклад табличного процесора зображено на рисунку 1.1.

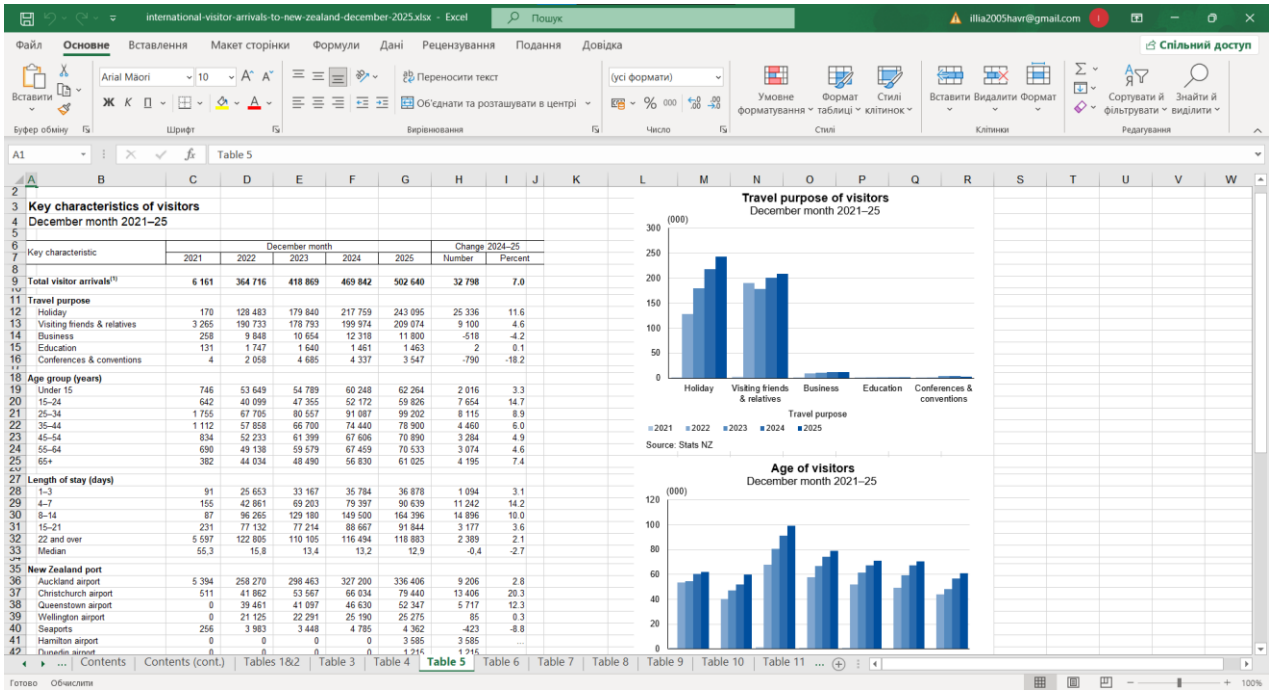

Рисунок 1.1 – Інтерфейс табличного процесора Microsoft Excel

Окремим класом програмного забезпечення є універсальні системи керування відносинами з клієнтами (CRM). Сучасний ринок пропонує широкий перелік подібних платформ, проте їхня адаптація під потреби інженерних служб виявляє архітектурні суперечності. Більшість CRM-систем фокусуються на фінансових показниках та фіксації маркетингових взаємодій. Інструментарій для детального опису технічних характеристик гетерогенного обладнання в них є обмеженим. Конфігурування специфічних полів для кожної окремої камери чи шлагбаума потребує кастомізації базового ядра системи. Це призводить до

додаткових фінансових витрат. Приклад кастомізації інтерфейсу комерційної системи наведено на рисунку 1.2.

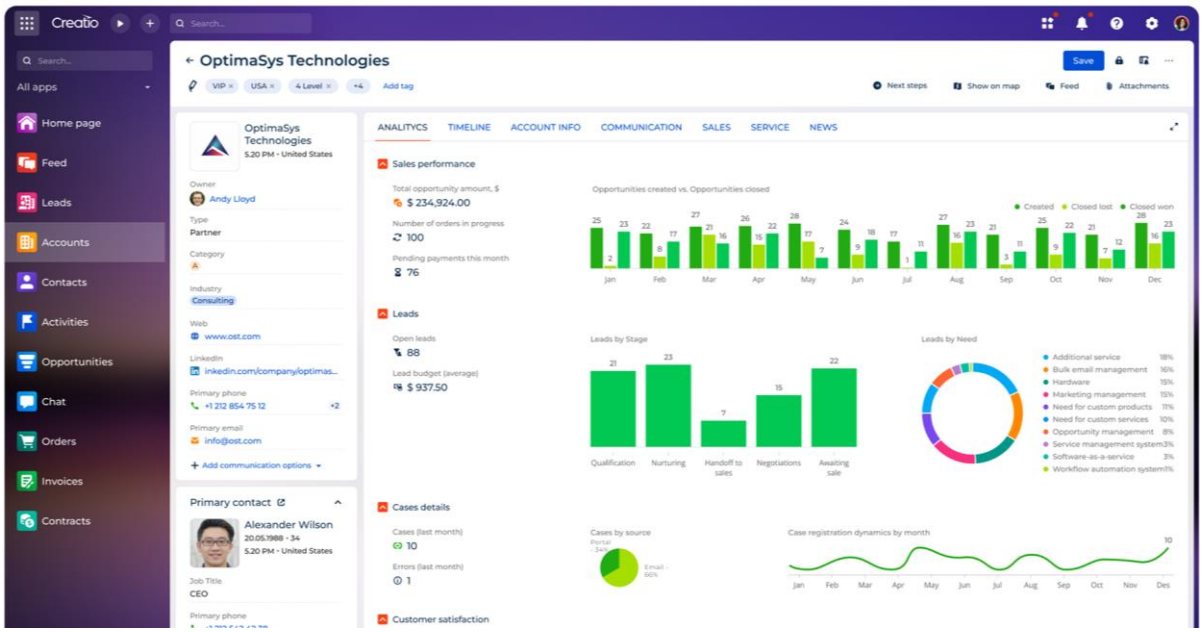

Рисунок 1.2 – Інтерфейс кастомізованої CRM-системи

Універсальні платформи автоматизації постачаються переважно за моделлю комерційного ліцензування. Періодична оплата за кожного активного користувача створює високе фінансове навантаження на підприємство. Більшість комерційних CRM мають закритий вихідний код. Це унеможливлює самостійну модифікацію логіки програми силами розробників компанії. Інтеграція сторонніх безкоштовних картографічних сервісів у такі продукти часто заблокована виробником. Користувачі змушені оплачувати комерційні API-ключі пропрієтарних карт. Такий підхід суперечить критеріям економічної оптимізації інженерної діяльності.

Для координації виїзного персоналу великі підприємства часто впроваджують спеціалізовані системи управління мобільними бригадами (Field Service Management, FSM). Передові продукти цього класу розробляються для автоматизації сервісної логістики та контролю виконання робіт на об'єктах. Впровадження подібних комплексів у діяльність компаній з обслуговування систем безпеки супроводжується низкою критичних обмежень. Основним

бар'єром є висока вартість розгортання та супроводу інфраструктури. Ці системи орієнтовані на хмарні сервіси глобальних провайдерів. Це створює постійні фінансові ризики для локального бізнесу. Приклад карти моніторингу у FSM-системі наведено на рисунку 1.3.

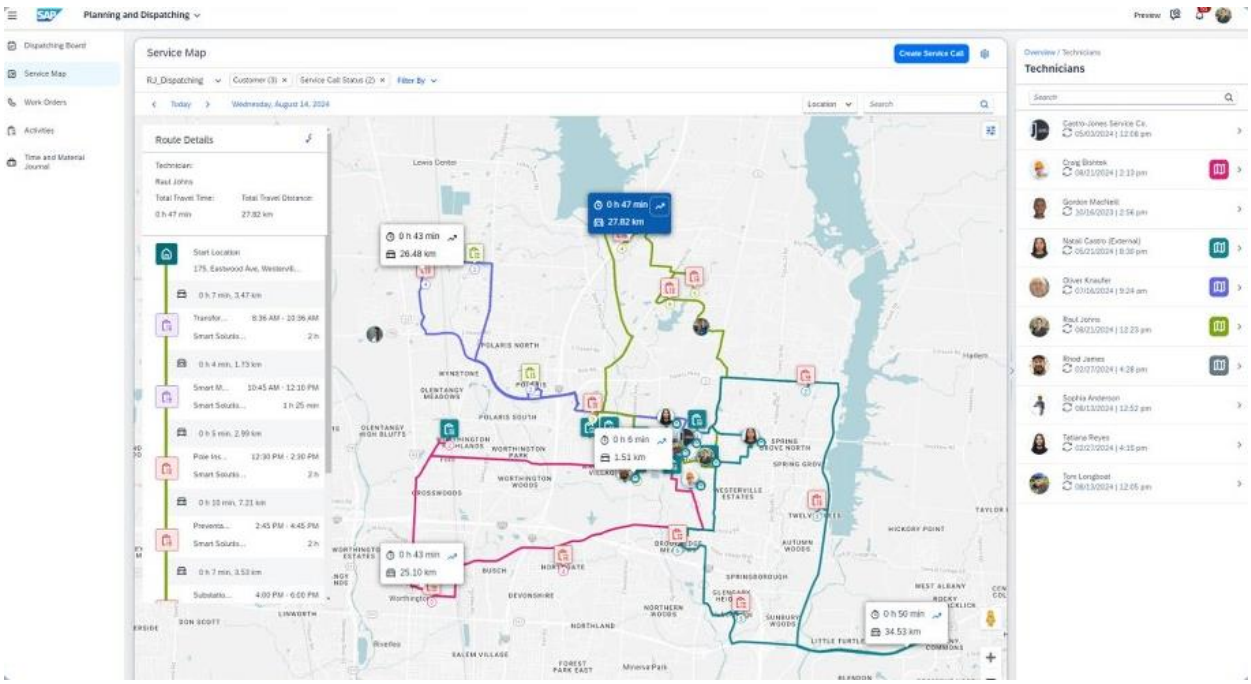

Рисунок 1.3 – Інтерфейс карти моніторингу з використанням комерційного API

Архітектура більшості комерційних FSM-платформ побудована навколо динамічної обробки екстрених заявок користувачів. Такий підхід суперечит специфіці інженерного захисту. Тут основним бізнес-процесом є плановий періодичний контроль та ведення технічного обліку. Наявність надлишкових функціональних модулів перевантажує інтерфейс користувача. Внутрішній технічний персонал змушений витрачати робочий час на заповнення необов'язкових полів. В той час як процес конфігурування та відключення непотрібних компонентів у пропрієтарному програмному забезпеченні потребує залучення сертифікованих розробників.

З порівняльного аналізу виключено паперові носії та табличні процесори типу Microsoft Excel. Обґрунтуванням цього рішення є функціональна розбіжність між статичною фіксацією даних та динамічним автоматизованим

управлінням, яке забезпечують спеціалізовані CRM і FSM системи. Традиційні форми ведення журналів не мають архітектурних засобів для координації бізнес-процесів у реальному часі. Вони не підтримують ізольовану багатокористувацьку роботу, не дозволяють реалізувати транзакційні правила на рівні СУБД та позбавлені інструментів інтеграції з геопросторовими бібліотеками. Ручне введення технічних нотаток або координат меж об'єктів супроводжується високим ризиком порушення цілісності інформації через людський чинник. Оскільки такі інструменти не піддаються серверній валідації та не мають API-інтерфейсів, їх зіставлення із іншими рішеннями є методологічно необґрунтованим. Важливим аспекти порівняння є можливість роботи з картографічними даними. Більшість комерційних платформ інтегруються виключно з закритими картографічними сервісами. Вони вимагають оплати за кожен запит до API. Для сервісної служби це призводить до збільшення операційних витрат. Реалізація функцій малювання та збереження полігонів в існуючих CRM та FSM системах зазвичай відсутня. Традиційні системи не пропонують інструментів інтеграції логічного видалення на рівні користувацького інтерфейсу. Вікна підтвердження операцій у стандартних CRM зазвичай обмежуються системними попередженнями. Вони не змушують оператора документувати детальні технічні причини деактивації об'єктів охорони. Для систематизації результатів проведеного аналізу було сформовано аналітичну таблицю 1.1.

Таблиця 1.1 – Порівняння програмних рішень

Критерій порівняння	**Універсальна CRM-система**	**Пропрієтарна FSM-платформа**	**Проектований веб-додаток**
Тип ліцензування	Комерційне (за користувача)	Комерційне (підписка)	Безкоштовне (власна розробка)
Доступ до коду	Закритий / Обмежений	Повністю закритий	Відкритий для модернізації
Контроль цілісності зв'язків	Частковий	Повний	Повний (на рівні сервісів)
Збереження полігонів геоданих	Через текстові описи	Обмежено (точково)	Повноцінно (OpenLayers)
Логіка Soft Delete	Налаштовувана	Залежить від конфігурації	Вбудована причина видалення
Облік специфічних полів	Потребує кастомізації	Обмежений шаблонами	Динамічні властивості

Проведений аналіз виявляє суттєві обмеження існуючих готових рішень у контексті захисту даних. Більшість комерційних систем автоматизації використовують реляційну модель зберігання. Фізичне видалення записів у таких архітектурах є стандартною операцією. Проте для інженерних сервісних служб безповоротне вилучення документів є критичним недоліком. Веб-додатки для внутрішнього використання не потребують кабінетів для зовнішніх замовників. Створення ізольованого робочого простору виключно для технічного персоналу дозволяє спростити структуру програми.

1.3 Постановка завдання

Проведення детального аналізу предметної області та критична оцінка існуючих комерційних аналогів дозволяють чітко визначити вектори автоматизації процесів. Основним етапом проектування системи є декомпозиція загальної мети дослідження на послідовність взаємопов'язаних інженерних кроків. Створення адаптованого під потреби користувача веб-додатка вимагає суворого узгодження архітектурних рішень між серверною та клієнтською частинами програми. Для досягнення поставленої мети автоматизації сервісного супроводу систем безпеки необхідно вирішити такі завдання:

1. Дослідити особливості бізнес-процесів технічного обслуговування комплексів безпеки та сформувати вичерпний перелік вимог до структури інформаційної моделі.

2. Спроектувати схему денормалізованих NoSQL-документів для сховища даних, передбачивши інтеграцію статичних атрибутів, вкладених масивів обладнання та просторових координат вершин багатокутників.

3. Розробити шари серверної частини додатка (backend) на базі платформи ASP.NET Core для реалізації бізнес-логіки, обробки асинхронних запитів та взаємодії із хмарною СУБД.

4. Реалізувати на рівні сервісного шару механізми динамічної фільтрації даних та обробки великих реєстрів за допомогою алгоритмів серверної пагінації сторінок.

5. Розробити та впровадити програмний регламент логічного видалення записів (SoftDelete) із забезпеченням обов'язкової інтерфейсної валідації текстового поля причини деактивації сутностей.

6. Створити користувацький інтерфейс клієнтської частини додатка (frontend) на основі компонентного підходу фреймворку Angular з інтеграцією табличного модуля DataTablesModule.

7. Інтегрувати інтерактивний картографічний модуль OpenLayers, реалізувавши алгоритми перерахунку координат для динамічного відображення і модифікації просторових багатокутників меж охорони.

8. Провести комплексне функціональне тестування розроблених програмних компонентів та інтерфейсів у середовищі локального емулятора для верифікації цілісності даних.

Наведений перелік завдань охоплює всі етапи розробки програмного продукту. Він виступає в ролі інженерного плану для подальших розділів дослідження. Кожен пункт безпосередньо пов'язаний із усуненням раніше виявлених недоліків готових CRM та FSM систем. Реалізація цих завдань дозволить створити оптимізоване середовище для ведення технічного обліку.

Визначення архітектурного базису проектованого веб-додатка базується на необхідності усунення фінансових та технічних обмежень, які притаманні комерційним програмним продуктам. Головною особливістю створюваної системи є повна відмова від пропрієтарних картографічних сервісів на користь рішень із відкритим вихідним кодом. Інтеграція бібліотеки OpenLayers у поєднанні з растровими плитками OpenStreetMap дозволить повністю виключити операційні витрати на оплату закритих API-ключів. Для інженерного бізнесу це рішення є чинником довгострокової економічної стабільності. Воно дозволить масштабувати систему без збільшення навантаження на бюджет компанії.

Майбутнє впровадження інтерактивної карти є не просто візуальним доповненням, а ключовою функціональною вимогою для ведення просторового обліку. Карта у структурі веб-додатка вирішить комплекс специфічних завдань:

– забезпечить візуальний контроль охорони об'єктів шляхом накладання напівпрозорих багатокутників на географічну підкладку в реальному часі;

– оптимізує просторову навігацію диспетчера, який отримає можливість вибору об'єкта нерухомості безпосередньо через клік на карті, минувши ручне введення текстових адрес;

– скоротить час обробки інцидентів за рахунок точного відображення меж відповідальності виїзних інженерно-технічних бригад;

– динамічно керуватиме деталізацією інтерфейсу, що буде виражатися в автоматичному приховані підписів та меж при зменшенні масштабу відображення для зниження навантаження на апаратну частину клієнтського термінала.

Картографічний модуль OpenLayers надаватиме розробнику повний контроль над векторними шарами. Це підвищить інформативність інтерфейсу для оперативного персоналу сервісної служби.

Вибір інструментів збереження інформації є критичним етапом проектування, який визначає швидкість роботи системи при обробці геопросторових даних. Традиційні реляційні бази даних вимагають побудови складних схем із великою кількістю зв'язаних таблиць. Рендеринг картки об'єкта разом із його периметром та списком встановленого обладнання у реляційній моделі потребує виконання важких операцій об'єднання. Це призводить до падіння продуктивності серверної частини при зростанні масиву даних. Для усунення цього бар'єру в проекті буде обґрунтовано використання NoSQL СУБД Azure Cosmos DB.

Документо-орієнтована модель сховища дозволить об'єднати різнотипні дані в межах єдиної структури JSON. Переваги такого підходу охоплюватимуть такі аспекти:

– забезпечення збереження статичних характеристик об'єкта нерухомості, масивів координат просторових багатокутників та списків технічних засобів у вигляді цілісного денормалізованого документа;

– виключення потреби в операціях злиття таблиць на рівні бази даних, завдяки чому зчитування всієї інформації про об'єкт виконуватиметься за один асинхронний запит;

– забезпечення гнучкості схеми даних, що дозволить додавати нові специфічні поля для інноваційних типів датчиків або відеокамер без проведення міграцій структури всієї бази;

– ефективне використання ключів партиціонування (PartitionKey) для горизонтального масштабування сховища при збільшенні кількості підконтрольних об'єктів.

Застосування NoSQL архітектури забезпечить високу швидкість відгуку клієнтської частини додатка. Поєднання технологічного стека ASP.NET Core на сервері та фреймворку Angular на клієнті дозволить створити швидкодіючий веб-додаток. Система буде орієнтована на внутрішнє використання технічним персоналом. Вона буде повністю ізольована від зовнішніх комерційних платформ, що гарантуватиме високий рівень безпеки та автономності.

2 ПРОЕКТУВАННЯ АРХІТЕКТУРИ ТА СТРУКТУРИ ДАНИХ ВЕБ-ДОДАТКА

2.1 Функціональні сценарії взаємодії та системна декомпозиція модулів веб-додатка

Проектування архітектури сучасних корпоративних веб-додатків базується на принципах уніфікації користувацьких інтерфейсів та інваріантності бізнес-логіки щодо ролей персоналу. При автоматизації сервісних процесів інженерної компанії доцільно відмовитися від надлишкового розгалуження функціональних прав на початкових етапах моделювання. Такий підхід спрощує проектування інтерфейсних рішень. Користувач отримує консолідований доступ до інструментів технічного нагляду. Центральне місце у процесі взаємодії посідає Use-Case діаграма, яка описує динаміку поведінки системи при виклику різних функціональних тригерів. Побудована модель фіксує переходи між станами інтерфейсу від моменту автентифікації до безпосереднього збереження даних. Користувачами веб-додатку виступають всі співробітники компанії, яка займається обслуговуванням охоронних систем.

Представлена нижче Use-Case діаграма демонструє циклічну структуру поведінкових сценаріїв працівника. Початкова фаза взаємодії вимагає активації форми для входу, де здійснюється введення логіну та паролю. Серверний шар системи виконує процедуру перевірки даних. При виявленні невідповідностей інформаційний потік повертається на початковий етап із виведенням повідомлення про помилку. Успішна автентифікація переводить термінал користувача на навігаційну панель. Цей елемент управління є головним розподільником системних запитів. Звідси працівник може обрати два базові вектори роботи, які охоплюють перехід до інтерактивної карти або відкриття таблиці записів сутності.

Логіку взаємодії користувача з інформаційними контурами веб-додатка відображено на рисунку 2.1.

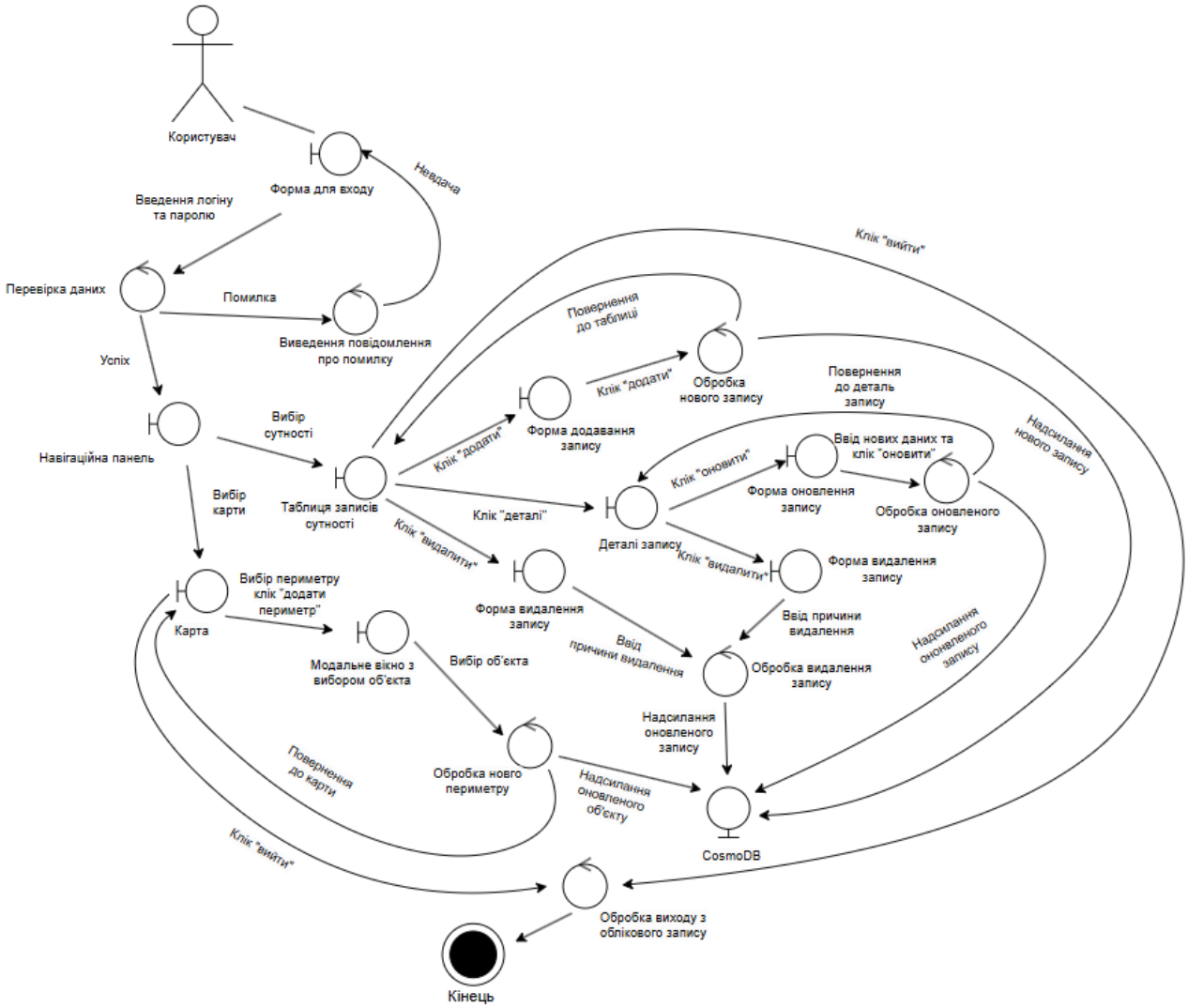

Рисунок 2.1 – Use-Case діаграма процесів взаємодії користувача з елементами системи

Аналіз Use-Case діаграми дозволяє виявити важливу архітектурну особливість проектованого веб-додатка. Вона полягає у структурній ізоморфності інформаційних контурів для різних типів даних. Програма оперує чотирма основними бізнес-сутностями, до яких належать об'єкти нерухомості, нотатки технічних інспекцій, сервісні події та картки персоналу. На рівні абстракції бази даних та користувацького інтерфейсу ці сутності мають

еквівалентний функціональний життєвий цикл. Вони володіють ідентичним набором базових маніпуляцій, попри суттєві відмінності у внутрішньому інформаційному вмісті та наборі атрибутів.

Ця ізоморфність наочно відображається у структурі зв’язків таблиці записів сутності на рисунку 2.1. Незалежно від того, яка саме сутність обрана на навігаційній панелі, користувачеві відкривається уніфікована керуюча матриця. Через неї реалізуються класичні операції життєвого циклу інформації. Клік по тригеру додавання активує форму створення нового запису з подальшим передаванням пакета даних на серверний шар обробки. Вибір елемента деталей запису дозволяє перейти до перегляду розширених характеристик сутності. Тут стають доступними функції оновлення параметрів через відповідну форму модифікації.

Процедура вилучення інформації також є уніфікованою для всіх чотирьох інформаційних контурів. Вона може бути ініційована як безпосередньо з загальної таблиці, так і з вікна детального перегляду конкретного запису. Запуск процесу активує форму видалення, яка вимагає від робітника обов'язкового введення причини деактивації. Після цього актується блок обробки, що транслює асинхронний запит до СУБД Cosmos DB.

Паралельний функціональний контур, відображений на Use-Case діаграмі, описує роботу з геопросторовими даними. При виборі карти користувач переходить в інтерактивне візуальне середовище OpenLayers. Процес проектування меж потребує вибору конкретного периметра на графічній підкладці. Після фіксації точок система викликає модальне вікно для прив'язки геометричного контуру до конкретного об'єкта нерухомості з реєстру. Блок обробки нового периметра здійснює пакування координат у денормалізований формат та надсилає оновлений об'єкт у сховище Cosmos DB. Завершення поточної сесії роботи з веб-додатком на будь-якому етапі реалізується через клік по тригеру виходу. Це активує блок очищення токенів авторизації та повертає інтерфейс до початкової форми входу. Побудована Use-Case діаграма доводить, що застосування принципу уніфікації інтерфейсних форм дозволяє суттєво

знизити когнітивне навантаження на персонал при роботі з різними типами даних.

Розподіл системних функцій та визначення внутрішньої структури взаємозв'язків між програмними компонентами здійснюється на етапі архітектурної декомпозиції. Метою цього процесу є поділ веб-додатка на логічно ізольовані модулі. Це дозволяє забезпечити автономність розробки окремих частин системи та спрощує подальшу модернізацію коду. Уніфікація чотирьох базових інформаційних контурів, виявлена на етапі аналізу користувацьких сценаріїв, безпосередньо впливає на архітектурну конфігурацію програмних блоків. Замість проектування окремих підсистем для кожної бізнес-сутності розробляється єдиний, але гнучко конфігурований модуль обробки даних. Ієрархічну структуру розподілу функціональних компонентів системи відображено на рисунку 2.2.

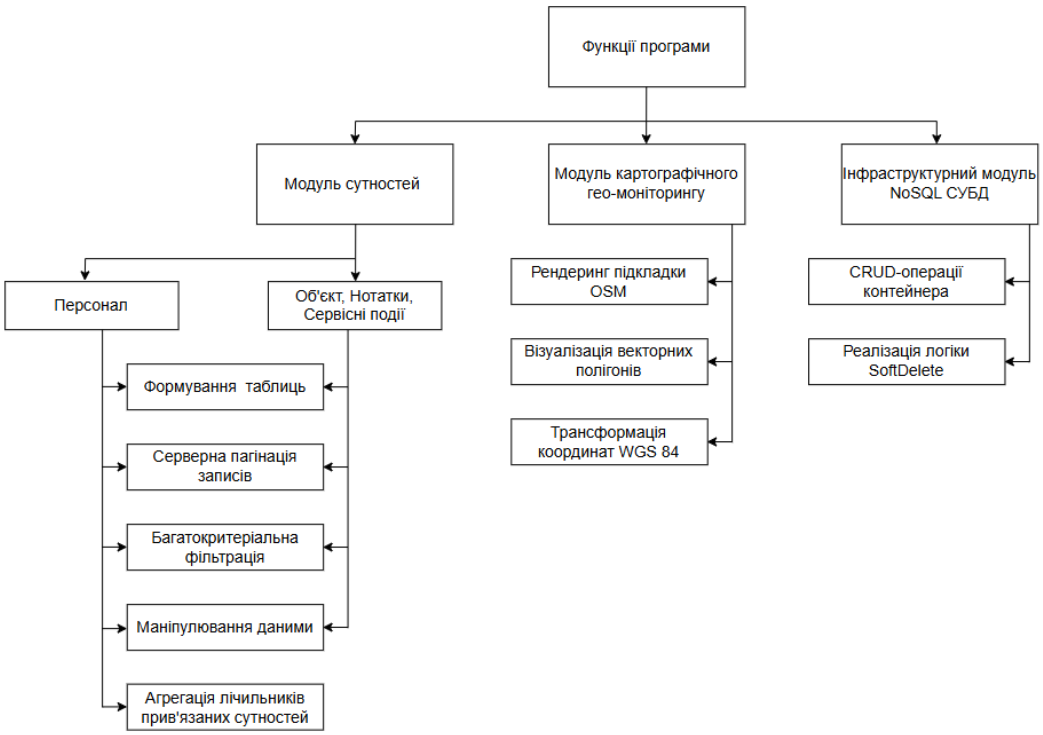

Рисунок 2.2 – Діаграма функціональної декомпозиції підсистем та модулів веб-додатка

Описова структура демонструє декомпозицію верхнього рівня на три незалежні підсистеми. Модуль сутностей виступає ядром облікової логіки веб-додатка. Він безпосередньо інкапсулює роботу з інформаційними реєстрами. На нижчих рівнях ієрархії цей модуль розгалужується на підмодуль персоналу та консолідований підмодуль об'єктів, нотаток і сервісних подій. Спільна програмна архітектура для цих елементів обумовлена тотожністю їхнього функціонального навантаження. Кожен із зазначених інформаційних блоків підпорядкований чотирьом базовим операціям, які реалізовані через відповідні підмодулі нижнього рівня, а п’ятий призначений тільки для аналітики, пов’язаної з персоналом.

Першим рівнем обробки в модулі сутностей є блок формування таблиць. Він відповідає за рендеринг інтерфейсних матриць на клієнтській стороні додатка. Наступний компонент забезпечує виконання серверної пагінації записів. Ця функція є критично важливою для оптимізації мережевого трафіку при роботі з великими масивами даних. Замість завантаження всього реєстру система виконує порційне зчитування інформації. Підмодуль багатокритеріальної фільтрації інтегрується в табличний інтерфейс для швидкого відбору записів за заданими параметрами. Блок маніпулювання даними об'єднує функції додавання, перегляду деталей та оновлення полів. Завершальним елементом цієї гілки є підмодуль агрегації лічильників прив'язаних сутностей. Він виконує математичний розрахунок пов'язаних документів у реальному часі. Наприклад, для кожної картки працівника система динамічно обчислює кількість створених ним технічних нотаток або закріплених об'єктів нерухомості.

Модуль картографічного гео-моніторингу функціонує паралельно з обліковим контуром додатка. Його архітектура розбита на три функціональні блоки нижнього рівня. Підмодуль рендерингу підкладки OSM відповідає за завантаження та відображення растрових плиток карти OpenStreetMap. Візуалізація векторних полігонів забезпечує коректне накладання графічних меж охорони поверх географічної основи. Підмодуль трансформації координат

WGS84 виконує математичне перетворення просторових даних між внутрішнім сховищем та клієнтським інтерфейсом карт.

Інфраструктурний модуль NoSQL СУБД винесений в окремий системний шар. Він ізолює логіку збереження інформації від інтерфейсних компонентів. Цей модуль ділиться на два блоки. Підмодуль CRUD-операцій контейнера безпосередньо взаємодіє з API бази даних. Він виконує запис та зчитування денормалізованих JSON-пакетів. Підмодуль реалізації логіки SoftDelete відповідає за виконання регламенту м'якого видалення. Він контролює зміну статусів документів та фіксацію супутніх технічних причин деактивації. Побудована схема функціональної декомпозиції демонструє високий рівень зв'язності всередині модулів при низькій залежності між окремими підсистемами.

2.2 Структурна організація та оптимізація сховища NoSQL

Вибір моделі збереження інформації визначає швидкість функціонування та можливості масштабування інформаційної системи. Для веб-додатка технічного обліку об'єктів охорони обґрунтовано використання документо-орієнтованої NoSQL СУБД Azure Cosmos DB. Традиційний реляційний підхід вимагає створення множини зв'язаних таблиць та виконання операцій об'єднання при кожній генерації інтерфейсу. NoSQL архітектура дозволяє уникнути цих обмежень за допомогою принципу денормалізації. Всі відомості про конкретну одиницю обліку упаковуються в єдиний цілісний документ JSON. Це мінімізує кількість дискових операцій введення-виведення на серверній стороні при зчитуванні карток.

Логічну структуру NoSQL-документа для основного контейнера сховища відображено на рисунку 2.3.

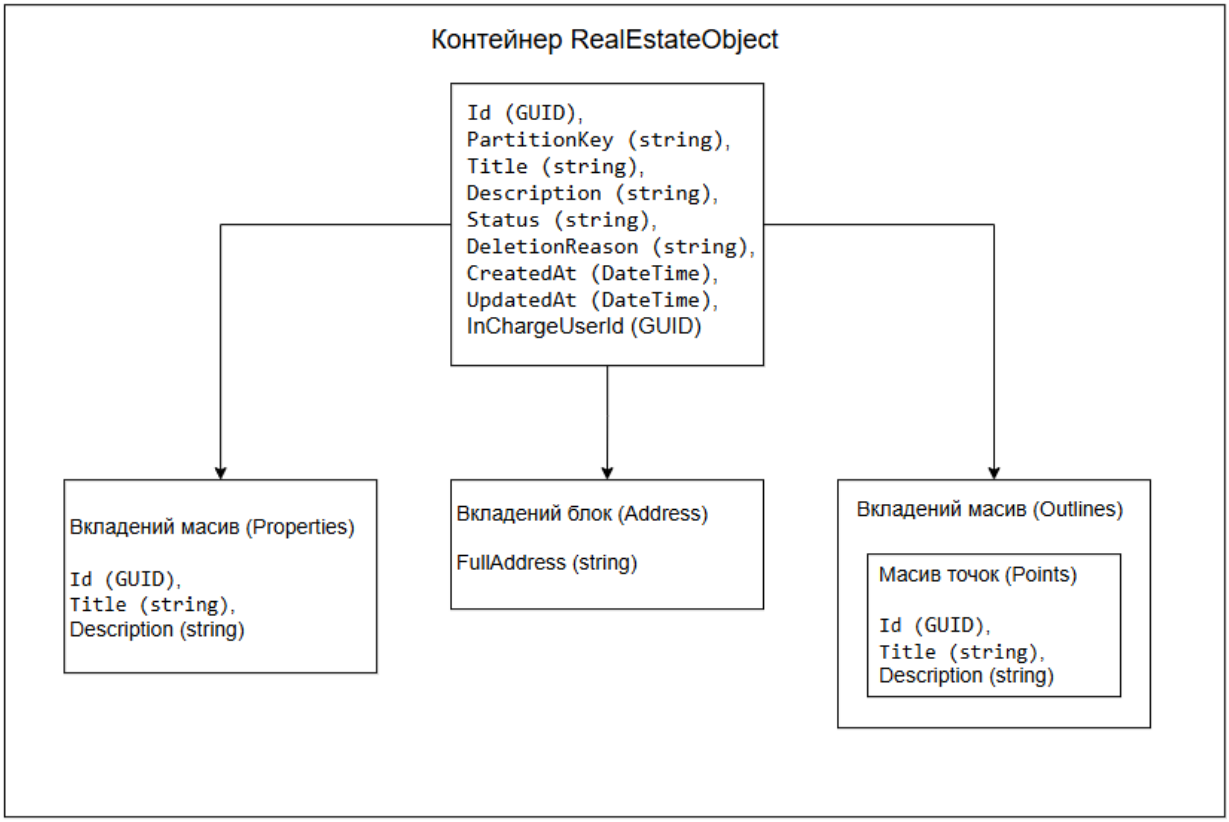

Рисунок 2.3 – Схема логічної структури NoSQL-документа Azure Cosmos DB

Представлена логічна модель демонструє побудову центрального контейнера RealEstateObject. Кореневий блок сутності містить набір обов'язкових системних та бізнес-атрибутів. Поле Id зберігає унікальний ідентифікатор запису у форматі глобального унікального індексу GUID. Атрибут PartitionKey визначає логічний ключ партиціонування для горизонтального масштабування бази. Текстові поля Title та Description відповідають за збереження назви та детального технічного опису об'єкта. Стан сутності у межах регламенту м'якого видалення контролюється через маркер Status. Якщо об'єкт деактивується, супутнє текстове обґрунтування записується в поле DeletionReason. Часові мітки створення та оновлення фіксуються в атрибутах CreatedAt та UpdatedAt. Зв'язок із відповідальним працівником реалізовано через ідентифікатор InChargeUserId.

Ієрархічна NoSQL структура передбачає інтеграцію трьох вкладених блоків безпосередньо в тіло батьківського документа. Першим є вбудований блок Address, який реалізує зв'язок один до одного та містить єдине поле FullAddress для фіксації географічного розташування нерухомості. Другий блок

представлений вкладеним масивом Properties. Він відображає відношення один до багатьох без створення сторонніх таблиць. Цей масив акумулює інформацію про встановлені пристрої безпеки, де кожен елемент містить власний Id, назву типу апаратури Title та опис зони контролю Description.

Третій структурний блок описує геопросторові характеристики об'єкта охорони. Вкладений масив Outlines містить внутрішню колекцію Points. Вона відповідає за зберігання вершин багатокутника просторових меж нерухомості. Кожна точка в масиві описується індивідуальним ідентифікатором, а також точними географічними координатами широти Latitude та довготи Longitude. Така модель дозволяє серверному шару ASP.NET Core отримувати вичерпні дані про об'єкт разом із його периметром та обладнанням за один асинхронний запит до СУБД.

Фізична організація сховища даних та конфігурування його внутрішніх параметрів безпосередньо впливають на швидкість обробки транзакцій. Для забезпечення стабільного часу відгуку при збільшенні обсягу інформації в Azure Cosmos DB застосовується механізм горизонтального партиціонування. Розподіл документів між фізичними серверами хмарної інфраструктури здійснюється на основі логічного ключа партиції. Вибір цього параметра визначає рівномірність навантаження на обчислювальні потужності системи. В межах проектованого веб-додатка як єдиний ключ партиціонування обрано системне поле сутності. Це дозволяє групувати документи за функціональною ознакою та оптимізувати розподіл даних усередині контейнера сховища.

Індексація інформації є наступним важливим компонентом оптимізації бази даних. Для швидкого виконання пошукових запитів конфігурується автоматична політика індексації всіх текстових та числових полів. Особлива увага приділяється обробці геопросторових даних, що зберігаються в масиві точок багатокутника. Програмне забезпечення бази даних автоматично транслює масиви координат у геопросторові структури. Завдяки цьому операції перевірки перетину меж або пошуку об'єктів у межах заданого радіуса виконуються на

рівні рушія СУБД. Швидкість зчитування координат залишається високою навіть при одночасній роботі багатьох користувачів.

Забезпечення цілісності інформації в NoSQL середовищі має свою специфіку через відсутність каскадних операцій на рівні СУБД. Збереження зв'язків між вкладеними структурами та батьківським документом гарантується самою архітектурою вбудованих об'єктів JSON. Оскільки оновлення масиву обладнання чи точок периметра відбувається в межах однієї транзакції з основним об'єктом нерухомості, виникнення часткових або неузгоджених станів є технічно неможливим. Запис або повністю фіксується у сховищі, або повністю відхиляється. Для підтримання актуальності зовнішніх ідентифікаторів, таких як зв'язок із працівником, валідація даних переноситься на рівень сервісного шару ASP.NET Core. Серверна частина додатка контролює наявність пов'язаних записів перед виконанням операцій модифікації. Такий розподіл обов'язків забезпечує надійність системи та високу швидкість запису інформації.

2.3 Побудова діаграми послідовності для моделювання асинхронних процесів гео-моніторингу

Динаміка функціонування веб-додатка та взаємодія його компонентів у часі визначаються за допомогою поведінкового моделювання. Центральним елементом цього етапу проектування є фіксація послідовності обміну повідомленнями між інтерфейсною частиною, сервером додатків та сховищем інформації. При реалізації функцій просторового обліку життєвий цикл запиту охоплює кілька рівнів абстракції системи. Асинхронний характер взаємодії дозволяє розподілити обчислювальне навантаження. Клієнтський термінал відповідає за графічне відображення, тоді як серверний контур забезпечує бізнес-валідацію та транзакційність. Процес реєстрації нових геометричних контурів нерухомості відображає архітектурну взаємодію всіх шарів розроблюваного програмного забезпечення. Послідовність виконання операцій при фіксації просторових меж наведено на рисунку 2.4.

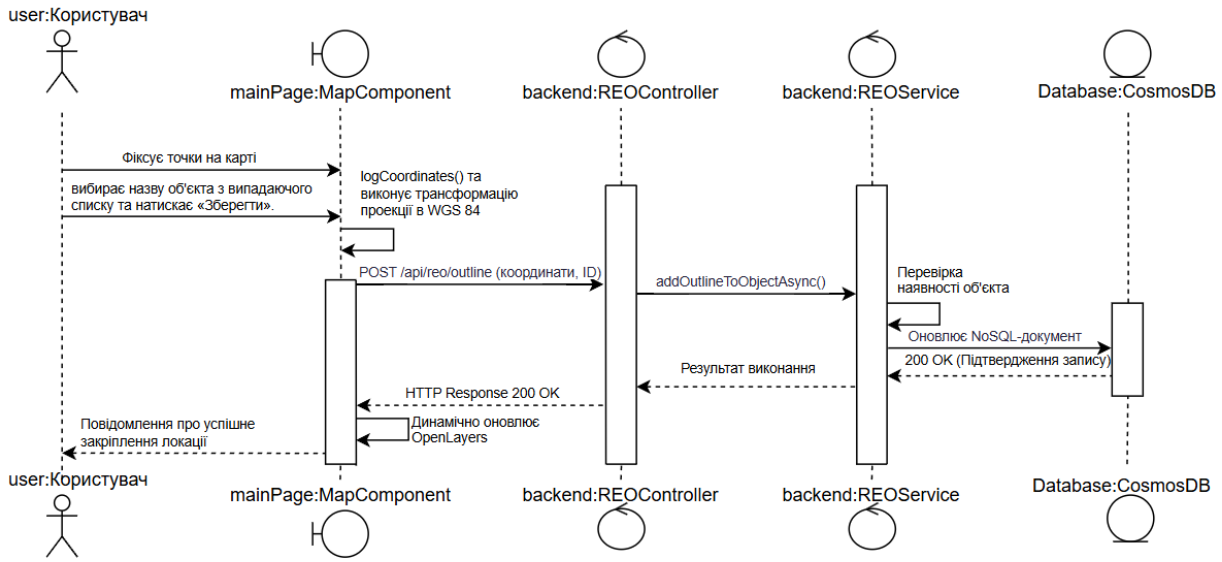

Рисунок 2.4 – Діаграма послідовності процесів створенння та збереження контурів об'єкта охорони

Аналіз діаграми послідовності дозволяє покроково простежити трансляцію графічних маніпуляцій користувача у фізичні записи бази даних. Процес ініціюється на рівні актора, який виконує фіксацію точок безпосередньо на інтерактивній карті. Компонент MapComponent фреймворку Angular перехоплює координати кліків та здійснює рендеринг ліній багатокутника в реальному часі. Після завершення графічного контуру працівник обирає назву об'єкта з випадаючого списку та активує команду збереження. Ця дія запускає внутрішній метод logCoordinates на стороні клієнта. Його завданням є математична трансформація просторових даних із поточної проекції карти у міжнародну географічну систему WGS 84.

Наступна фаза взаємодії переносить обробку даних на серверний рівень системи. Клієнтський модуль карти формує асинхронний HTTP POST запит на адресу /api/reo/outline. Пакет інформації містить масив координат та ідентифікатор об'єкта охорони. Запит приймається контролером REOController на серверній платформі ASP.NET Core. Контролер виступає в ролі вхідного шлюзу API. Він транслює отримані параметри до сервісного шару бізнес-логіки шляхом виклику асинхронного методу addOutlineToObjectAsync().

Рівень сервісу REOService виконує обов'язкову перевірку наявності цільового об'єкта в системі. При успішному завершенні валідації сервіс здійснює операцію оновлення NoSQL-документа. Нові координати меж записуються у вкладений масив об'єкта та передаються до СУБД Cosmos DB. Хмарне сховище фіксує зміни у контейнері та повертає статус успішного виконання транзакції. Повідомлення 200 OK передається назад по ланцюжку архітектурних шарів до клієнтського додатка. Отримавши підтвердження, MapComponent викликає внутрішні методи бібліотеки OpenLayers для динамічного оновлення векторного шару. Користувач бачить повідомлення про успішне закріплення локації на екрані. Побудована схема взаємодії демонструє застосування асинхронних запитів для забезпечення безперервності роботи інтерфейсу.

Реалізація користувацьких інтерфейсів на базі фреймворку Angular вимагає проектування реактивних контурів обміну даними. Для забезпечення асинхронної взаємодії з REST API серверної частини застосовується бібліотека RxJS. Всі маніпуляції з реєстрами сутностей трансформуються у потоки подій. Це дозволяє динамічно оновлювати табличні форми без перезавантаження сторінок веб-додатка. Використання потоків (Observables) забезпечує безперервне спостереження за станом мережевих запитів. При ініціації користувачем операцій фільтрації або пагінації інтерфейсний шар не блокує потік виконання. Система очікує відповіді від сервера у фоновому режимі, підтримуючи стабільну працездатність клієнтського термінала.

Організація внутрішньої архітектури фронтенд-частини базується на ієрархічному підпорядкуванні компонентів. Передача інформаційних пакетів між різними рівнями інтерфейсу реалізується через декоратори @Input() та @Output(). Батьківський компонент реєстру координує загальний стан сховища даних на клієнті. Він надсилає масиви об'єктів у дочірні модулі таблиць для безпосереднього відображення. При активації робітником форми модифікації або перегляду деталей дочірній віджет генерує подію зворотного зв'язку. Через емітер подій інструкція передається вгору по ієрархічному дереву для виконання відповідного HTTP-запиту до ASP.NET Core контролера.

Особлива увага при поведінковому моделюванні приділяється життєвому циклу реактивних підписок. Для запобігання витокам оперативної пам'яті в браузері всі потоки даних підлягають обов'язковій деактивації при знищенні інтерфейсних компонентів. Метод життєвого циклу ngOnDestroy() виступає тригером для відписки від відкритих потоків. Використання операторів вищого порядку, таких як switchMap або takeUntil, дозволяє автоматично скасовувати попередні незавершені HTTP-запити, якщо користувач сформував новий пошуковий запит. Такий підхід гарантує високу швидкість обробки інформації та виключає появу застарілих даних на екрані. Побудована логіка поведінкового рівня клієнтської частини забезпечує повну синхронізацію стану інтерфейсу з NoSQL сховищем.

2.4 Розробка алгоритму логічного видалення веб-додатку

Управління життєвим циклом інформації в системах технічного обліку потребує впровадження механізмів захисту від випадкової втрати даних. Класичне фізичне вилучення записів із контейнерів NoSQL бази даних створює ризики порушення цілісності пов'язаних журналів інспекцій та сервісних подій. Для усунення цього фактору проектований веб-додаток використовує програмний регламент логічного приховування SoftDelete. Суть підходу полягає у зміні внутрішнього стану сутності без її фактичного видалення з дискового простору Azure Cosmos DB. Процес деактивації повністю перенесений на рівень серверної бізнес-логіки та підпорядкований строгому алгоритмічному контролю. Покрокову схему обробки запиту на логічне видалення об'єкта нерухомості відображено на рисунку 2.5.

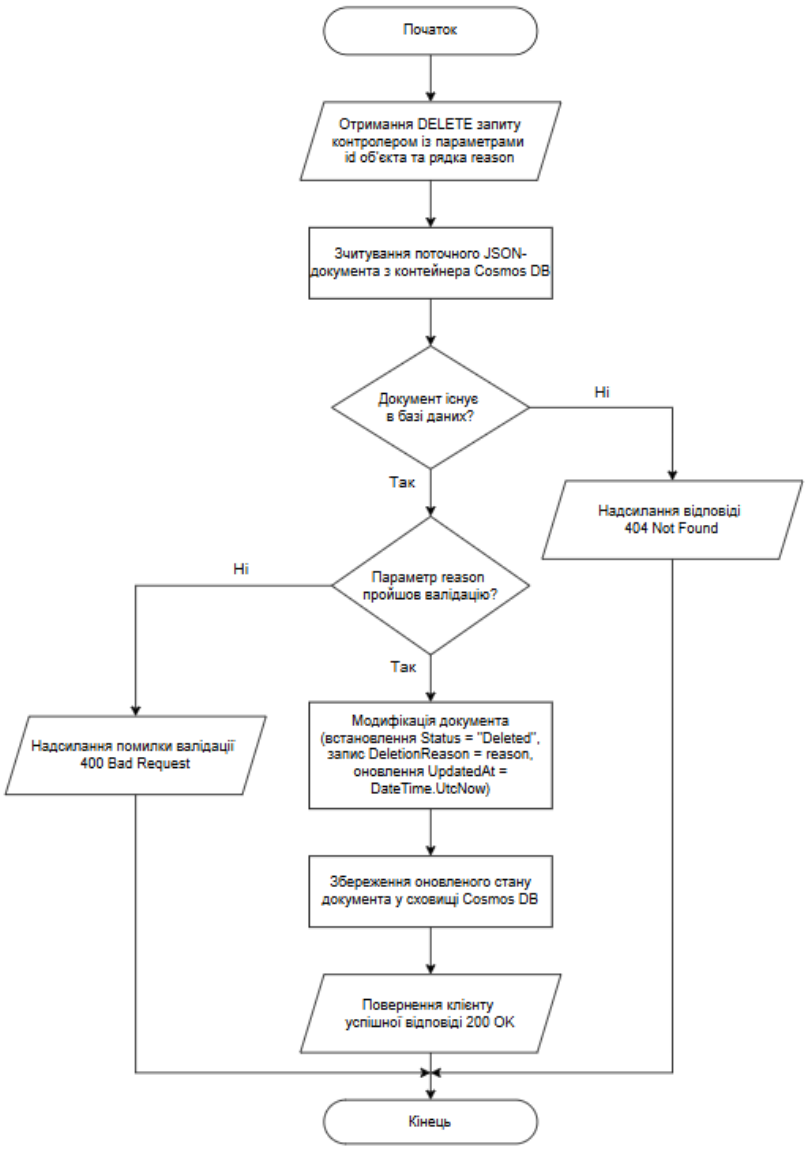

Рисунок 2.5 – Блок-схема процесу логічного видалення об'єкта нерухомості з валідацією текстових причин

Візуалізований алгоритм визначає точну послідовність операцій серверного шару при отриманні команди на деактивацію об'єкта. Процес починається з надходження HTTP DELETE запиту до API-контролера. Пакет даних містить унікальний ідентифікатор id сутності та текстовий рядок reason з описом причини видалення. Першим кроком алгоритму є виконання операції зчитування поточного стану JSON-документа з контейнера Cosmos DB за вказаним індексом. Далі логіка переходить до першої розгалужувальної точки для перевірки наявності запису у сховищі.

Якщо документ відсутній у базі даних, інформаційний потік спрямовується по негативній гілці. Система формує та надсилає клієнту статус відповіді 404 Not Found, після чого алгоритм завершує роботу. При успішному виявленні об'єкта процес переходить до другої логічної перевірки. На цьому етапі виконується комплексна валідація параметра reason на відповідність правилам бізнес-логіки. Серверний код аналізує довжину рядка та відсутність порожніх значень. Невдале проходження валідації активує блок формування помилки 400 Bad Request. Пакет помилки повертається користувачу, а виконання операції переривається.

Успішна валідація причини деактивації переводить алгоритм до фази безпосередньої модифікації структури документа. Програма послідовно змінює значення трьох внутрішніх атрибутів об'єкта нерухомості. Системне поле Status приймає фіксовану константу "Deleted". У полі DeletionReason записується переданий оператором текст обґрунтування. Мітка часу UpdatedAt оновлюється поточним системним часом сервера у форматі UTC. Оновлений JSON-документ передається назад до контейнера Cosmos DB для перезапису. Фінальним кроком алгоритму є формування успішної відповіді сервера зі статусом 200 OK. Це слугує сигналом для клієнтської частини Angular про необхідність динамічного виключення об'єкта з активних реєстрів інтерактивної карти.

Аналіз винятків на етапі алгоритмічного проектування дозволяє мінімізувати ризики нестабільної роботи серверної частини веб-додатка. При обробці запитів на логічну деактивацію сутностей можливе виникнення аномалій мережевого зв'язку або спроб несанкціонованого доступу до заблокованих елементів. Якщо об'єкт нерухомості вже має маркер "Deleted", повторна ініціація процедури SoftDelete повинна перериватися на ранніх стадіях перевірки. Серверна логіка ASP.NET Core інкапсулює ці правила у вигляді спеціальних фільтрів дій. Будь-яке відхилення від штатного сценарію супроводжується генерацією структурованого JSON-пакета помилки з детальним описом контексту події. Це спрощує діагностику системи при виникненні конфліктів у контейнері бази даних.

Проектування відповідних HTTP-статусів відповідей backend-шару базується на стандартах REST-архітектури. При виявленні порожнього або занадто короткого рядка з описом причини видалення сервер повертає статус 400 Bad Request. Проте внутрішній вміст відповіді розширюється за рахунок словника помилок валідації ValidationProblemDetails. Клієнтська частина Angular використовує цей об'єкт для підсвічування конкретних полів форми. Ситуація, коли паралельно два користувачі намагаються змінити стан одного об'єкта, обробляється через механізм оптимістичного блокування.

Особлива увага приділяється ізоляції видалених документів від поточних операційних реєстрів. Після успішного виконання алгоритму SoftDelete запис залишається фізично присутнім у базі даних, але стає невидимим для стандартних операцій читання. Для цього всі базові GET-запити до контейнера автоматично розширюються предикатом фільтрації, який відсікає елементи зі статусом деактивації. Доступ до наданої категорії об'єктів залишається можливим через безпосередній доступ до бази. Таке архітектурне рішення гарантує збереження історичних даних технічного нагляду. Воно виключає ризики випадкового спотворення звітності при проведенні щорічного аудиту систем безпеки.

2.5 Обґрунтування вибору програмного інструментарію та системної архітектури

Формування технологічного стека веб-додатка базується на аналізі критеріїв продуктивності, кросплатформеності та швидкості розгортання компонентів. Для реалізації серверної частини обрано програмну платформу ASP.NET Core. Дане середовище володіє високими показниками обробки вхідних запитів за секунду завдяки оптимізованому конвеєру HTTP-запитів Kestrel [8]. Вбудований механізм впровадження залежностей (Dependency Injection) дозволяє створювати слабкопов'язані сервісні шари [15, 18]. Це спрощує проведення модульного тестування та інтеграцію з NoSQL сховищами.

Використання асинхронної моделі програмування (async/await) на рівні контролерів мінімізує блокування потоків виконання при зверненні до зовнішніх баз даних.

Клієнтська частина веб-додатка проектується на базі фреймворку Angular. Вибір даного інструменту обумовлений його компонентною архітектурою та суворою типізацією за допомогою мови TypeScript [9, 10]. Структура Single Page Application (SPA) забезпечує перенесення логіки формування інтерфейсу безпосередньо у браузер робітника. Автоматичне двостороннє зв'язування даних (Data Binding) синхронізує стан внутрішніх об'єктів з екранними формами таблиць. Використання модульної системи Angular дозволяє розділити інтерфейс на ізольовані блоки. Це оптимізує швидкість первинного завантаження програми через механізм відкладеної генерації модулів (Lazy Loading).

Для інтеграції геопросторових функцій застосовується спеціалізована клієнтська бібліотека OpenLayers. Вона забезпечує високу швидкість рендерингу растрових та векторних шарів завдяки використанню апаратного прискорення WebGL. Бібліотека сумісна з фреймворком Angular та дозволяє динамічно керувати просторовими об'єктами через об'єктну модель JavaScript. OpenLayers підтримує роботу з багатьма картографічними проекціями, що спрощує трансформацію координат точок периметра без залучення серверних потужностей. Такий розподіл технологічних обов'язків між ASP.NET Core, Angular та OpenLayers формує надійну основу для масштабування системи.

Ефективність функціонування розроблюваного веб-додатка забезпечується через проектування трирівневої системної архітектури. Розподіл обов'язків реалізується за шаблоном Controller-Service-Repository [6, 11]. Кожен рівень ізольований від сусідніх за допомогою інтерфейсів абстракції. Шар контролерів (Controller) приймає вхідні HTTP-запити від клієнтської частини Angular. Його завданням є первинна десеріалізація JSON-пакетів та маршрутизація сигналів. Контролери не містять правил обробки інформації предметної області. Вони лише делегують виконання операцій наступному структурному компоненту системи.

Серверний шар бізнес-логики (Service) інкапсулює всі правила валідації та координації транзакцій. Тут відбувається перевірка повноти даних та накладання обмежень, таких як обов'язкове введення текстових причин деактивації при виконанні SoftDelete. Сервіси оперують бізнес-моделями та не залежать від конкретної реалізації бази даних. Рівень репозиторіїв (Repository) безпосередньо відповідає за збереження стану об'єктів. Він здійснює взаємодію з Azure Cosmos DB через офіційний SDK-драйвер [7, 14]. Модифікація коду на рівні збереження інформації не потребує переробки бізнес-логіки. Це підвищує гнучкість супроводу програмного забезпечення.

Мережева взаємодія між клієнтом та сервером побудована на принципах REST-архітектури [16]. Для наочної фіксації архітектурних зв'язків серверної частини розроблено діаграму класів на основі сутності REO, що зображено на рисунку 2.6.

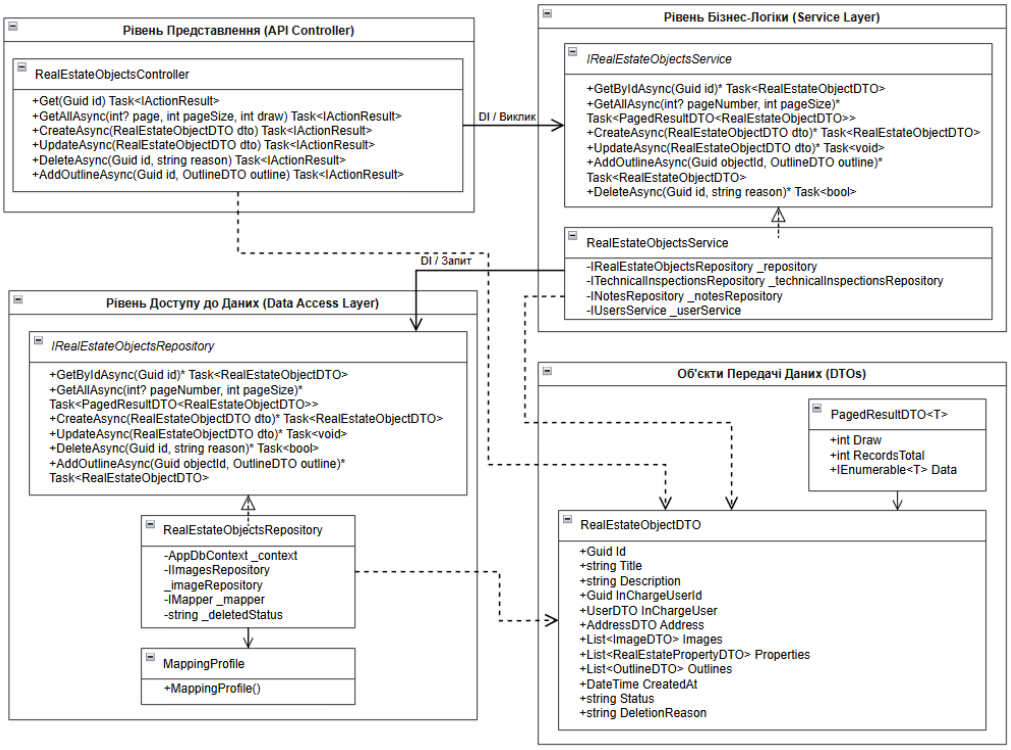

Рисунок 2.6 – Діаграма класів сутності REO

Графічна схема зображена вище ілюструє практичну реалізацію шаблону трирівневого розподілу обов'язків. Вона відображає взаємодію між компонентами представлення, бізнес-логіки та збереження даних на прикладі

модуля об'єктів охорони. Кожен рівень системи функціонує ізольовано. Це досягається шляхом проектування чітких інтерфейсів взаємодії. Впровадження залежностей відбувається через конструктори відповідних об'єктів. Такий підхід усуває жорстку зв'язаність коду.

Рівень API представлений класом контролера, який приймає зовнішні запити та делегує їх виконання сервісному шару. Основна логіка обробки зосереджена у сервісних класах. Вони взаємодіють із рівнем доступу до даних не безпосередньо, а через абстрактні інтерфейси. Репозиторій забезпечує виконання низькорівневих операцій із NoSQL-документами та повертає об'єкти доменних моделей. Доменні сутності відображають внутрішню структуру даних, включаючи координати геопросторових контурів. Подібна декомпозиція спрощує масштабування веб-додатка. Вона також дозволяє проводити ізольоване тестування окремих модулів без підключення реальної бази даних [19, 20].

Передача інформаційних пакетів здійснюється через захищений протокол HTTPS за допомогою стандартних методів GET, POST, PUT та DELETE [23, 24]. Для безпеки контуру обліку застосовуються токени JWT [17]. Вони передаються в заголовках кожного запиту та перевіряються на серверній стороні при кожному зверненні до API. Системна архітектура повністю ізолює внутрішній обліковий контур веб-додатка від зовнішніх рівнів телеметрії та апаратних засобів охорони. Збір координатної інформації та технічних нотаток виконується виключно через регламентовані ендпоінти. Такий підхід мінімізує вразливість бази даних та гарантує конфіденційність просторових меж об'єктів.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБ-ДОДАТКА

3.1 Реалізація серверної частини веб-додатка на базі ASP.NET Core

Проектування серверного компонента системи автоматизації сервісної служби базується на використанні кросплатформеної інфраструктури ASP.NET Core. Програмна логіка бекенду структурується за тришаровим шаблоном розробки. Розділення відповідальності між компонентами забезпечує ізоляцію процесів обробки HTTP-запитів, виконання бізнес-правил та безпосередньої взаємодії зі сховищем даних. На верхньому рівні архітектури знаходяться контролери API. Вони координують зовнішній інтерфейс взаємодії з клієнтським додатком. Проміжний сервісний шар інкапсулює алгоритми перевірки станів об'єктів нерухомості. Нижній рівень репозиторіїв абстрагує логіку NoSQL-запитів, що дозволяє усунути жорстку залежність коду від конкретних драйверів баз даних.

Організація структури папок серверного рішення відображає концепцію модульності. Файлова система бекенду містить окремі директорії для моделей даних, репозиторіїв та компонентів представлення інформації. Наведена нижче ієрархія забезпечує можливість паралельної розробки різних шарів системи. Центральним елементом цієї підсистеми є клас RealEstateObject, тому на основі нього буде проводитися демонстрація та пояснення роботи. Він описує об'єкт обліку інженерної служби. Об'єкт містить ідентифікатори, координати меж, текстові поля назви та опису. Аналіз коду дозволяє виділити кілька важливих архітектурних рішень, що будуть представлені нижче. Архітектура папок і розподіл ключових файлів рішень представлені на рисунку 3.1.

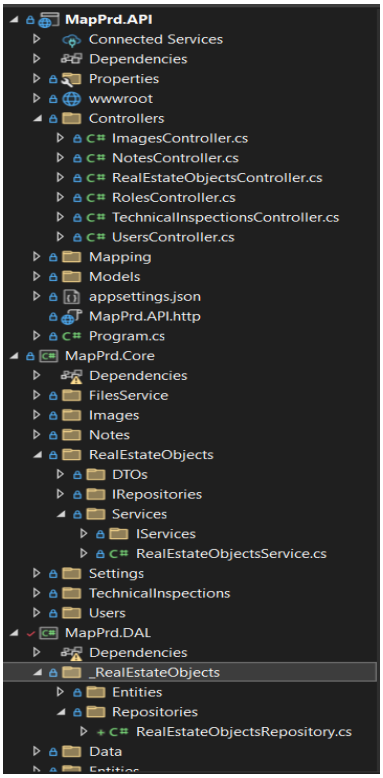

Рисунок 3.1 – Архітектура файлової системи та розподіл компонентів бекенду

Проект MapPrd.DAL містить сутності, які безпосередньо відображають структуру документів NoSQL сховища. Центральним елементом цієї підсистеми є клас RealEstateObject, тому на основі нього буде проводитися демонстрація та пояснення роботи. Він описує об'єкт обліку інженерної служби. Об'єкт містить ідентифікатори, координати меж, текстові поля назви та опису. Аналіз коду дозволяє виділити кілька важливих архітектурних рішень. Конструктор класу ініціалізує колекції для запобігання виникненню винятків типу NullReferenceException під час первинного звернення до об'єкта. Поля CreatedAt та UpdatedAt забезпечують фіксацію часових міток для проведення технічного аудиту. Атрибути TechnicalInspectionsNeeded та InspectionInterval використовуються підсистемою планування для визначення періодичності виїздів персоналу. Програмна реалізація сутності представлена нижче:

```
public class RealEstateObject {
    public RealEstateObject() {
        ImagesId = new();
        Properties = new();
        Outlines = new();
        PartitionKey = string.Empty;
        Title = string.Empty;
        Description = string.Empty;
        TechnicalInspectionsNeeded = false;
}
    public string PartitionKey { get; set; }
    public Guid Id { get; set; }
    public string Title { get; set; }
    public string Description { get; set; }
    public Guid InChargeUserId { get; set; }
    public Address? Address { get; set; }
    public List<string>? ImagesId { get; set; }
    public List<RealEstateProperty> Properties { get; set; }
    public List<Outline>? Outlines { get; set; }
    public DateTime CreatedAt { get; set; }
    public DateTime UpdatedAt { get; set; }
    public string? Status { get; set; }
    public string? DeletionReason { get; set; }
    public bool TechnicalInspectionsNeeded { get; set; }
    public string? InspectionInterval { get; set; }
}
```

Для безпечного обміну даними між сервером та клієнтом доменні моделі не повинні передаватися безпосередньо через HTTP-інтерфейси. Це порушило б принцип ізоляції шарів та створило вразливості в системі безпеки. Натомість на рівні MapPrd.Core проекується дзеркальний клас RealEstateObjectDTO. Даний об'єкт перенесення даних містить анотації валідації полів, такі як [Required]. Він також розширюється вкладеними об'єктами InChargeUser, Notes та TechnicalInspections. Це дозволяє агрегувати повну інформацію про технічний стан систем охорони в межах одного запиту клієнта без ускладнення базової сутності бази даних. Взаємодія між сутністю та DTO реалізується за допомогою автоматичного мапера, конфігурація якого розглядається у наступних етапах проектування.

Процес інтеграції створеного веб-додатка з документ-орієнтованим сховищем даних реалізується за допомогою об'єктно-реляційного мапера Entity Framework Core із використанням спеціалізованого провайдера Microsoft.EntityFrameworkCore.Cosmos. Архітектура взаємодії з базою даних розробляється з розрахунком на розгортання в середовищі локального емулятора Azure Cosmos DB Emulator. Це дозволяє імітувати поведінку реального хмарного

сховища без додаткових інфраструктурних витрат. Основним елементом конфігурації є визначення точок підключення, автентифікаційних ключів та назви цільової бази даних. Ці параметри зчитуються конфігураційним контуром веб-додатка під час старту серверної частини.

Центральним вузлом підсистеми доступу до даних є клас контексту AppDbContext. Даний компонент розширює базовий клас DbContext та інкапсулює в собі колекції DbSet, що відповідають за типізацію NoSQL-документів. На відміну від стандартного реляційного підходу, метод OnConfiguring перевизначається для явного вказання інструкції UseCosmos. Структура демонструє механізм типізації. Метод OnModelCreating забезпечує реєстрацію ізольованих класів конфігурації для кожної доменної сутності. Це дозволяє уникнути перевантаження коду контексту інструкціями Fluent API. Конфігурація параметрів сховища Cosmos DB зчитується через впровадження залежностей за допомогою інтерфейсу IOptions<CosmosDb>. Це дозволяє винести адреси хостів і ключі авторизації за межі коду контексту. У результаті ми отримуємо чистішу архітектуру та безпечне керування конфігураціями. Програмний код ініціалізації та налаштування з'єднання з БД наведено нижче:

```
public class AppDbContext : DbContext
{
    private readonly CosmosDb _cosmosDb;
    // ...
    public DbSet<Image> Images { get; set; }
    public DbSet<RealEstateObject> RealEstateObjects { get; set; }
    // ...
    public AppDbContext(DbContextOptions<AppDbContext>options,
IOptions<CosmosDb> cosmosDb) : base(options) { _cosmosDb = cosmosDb.Value; }
    protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder) {
        optionsBuilder.UseCosmos(_cosmosDb.Endpoint, _cosmosDb.Key,
_cosmosDb.Database); }

protected override void OnModelCreating(ModelBuilder modelBuilder){
    modelBuilder.ApplyConfiguration(new RealEstateObjectConfiguration());
    modelBuilder.ApplyConfiguration(new ImageConfiguration());
    // ...
    base.OnModelCreating(modelBuilder);}
```

Для стабільного функціонування системи у локальному середовищі розробки важливим є етап автоматичного створення структури бази даних. За це

відповідає компонент CosmosDbInitializer. Під час запуску веб-додатка цей клас викликає асинхронний метод EnsureCreatedAsync. Дана інструкція перевіряє наявність бази даних в емуляторі. Якщо база даних відсутня, система створює її автоматично разом із визначеними контейнерами. Такий підхід виключає необхідність ручного налаштування сховища перед кожним сеансом розробки або тестування бізнес-логіки.

Розподіл документів усередині NoSQL сховища базується на логічному розділенні за контейнерами та визначенні ключів партиціонування. Для сутності об'єктів нерухомості конфігурація описується у класі RealEstateObjectConfiguration, що наведено нижче:

```
    public class RealEstateObjectConfiguration : IEntityTypeConfiguration<RealEstateObject> {
        public void Configure(EntityTypeBuilder<RealEstateObject> builder) {
            builder.ToContainer(Constants.CONTAINER_REAL_ESTATE_OBJECT);
            builder.HasPartitionKey(x => x.PartitionKey);
            builder.HasKey(x => x.Id);
            builder.OwnsOne(x => x.Address, a => {
                a.Property(x => x.FullAddress).IsRequired();});
            builder.OwnsMany(x => x.Properties, p => {
                p.WithOwner();
                p.Property(x => x.Id);
                p.Property(x => x.Title).IsRequired();
                p.Property(x => x.Description);
                p.Property(x => x.CreatedAt).IsRequired();
                p.Property(x => x.UpdatedAt).IsRequired();});
            builder.OwnsMany(x => x.Outlines, o => {
                o.WithOwner();
                    o.Property(x => x.Id);
                    o.OwnsMany(x => x.Points, pt => {
                    pt.WithOwner();
                    pt.Property(x => x.Id);
                    pt.Property(x => x.Latitude).IsRequired();
                    pt.Property(x => x.Longitude).IsRequired(); }); });
            builder.Property(x => x.ImagesId).IsRequired();
            builder.Property(x => x.Title).IsRequired();
            builder.Property(x => x.Description);
            builder.Property(x => x.InChargeUserId).IsRequired();
            builder.Property(x => x.CreatedAt).IsRequired();
            builder.Property(x => x.UpdatedAt).IsRequired(); }
    }
```

Важливою особливістю проектування є відображення ієрархічних зв'язків. Вкладені сутності налаштовуються за допомогою методів OwnsMany та OwnsOne. Це вказує ORM-маперу на необхідність збереження цих даних у межах одного батьківського документа, а не в окремих таблицях. Не менш важливою

частиною є інструкція builder.ToContainer, яка прив'язує сутність до конкретного логічного контейнера, назва якого визначена у константах системи в папці Shared. Метод builder.HasPartitionKey встановлює властивість PartitionKey як ключ партиціонування. Це критично для забезпечення горизонтального масштабування та оптимізації маршрутизації внутрішніх запитів до емулятора бази даних. Конфігурація вкладених точок координат (Points) у структурі Outlines демонструє глибоку денормалізацію моделі. Обробка цих структур забезпечує коректне збереження просторових геометрій, сформованих персоналом на клієнтському рівні системи.

Проектування внутрішньої логіки обробки даних вимагає створення стійкого рівня абстракції для виконання операцій із документами NoSQL СУБД. На рівні доступу до даних за цю функціональність відповідає клас RealEstateObjectsRepository. Компонент інкапсулює прямі виклики до контексту бази даних, забезпечуючи ізоляцію сервісного шару від специфіки виконання асинхронних запитів. Важливою інженерною задачею при розробці репозиторію є автоматична генерація системних параметрів сутності під час її створення. До таких параметрів належать унікальні ідентифікатори, тимчасові мітки та статичні ключі партиціонування.

Перед фіксацією документа в контейнері Cosmos DB система повинна забезпечити валідацію внутрішніх колекцій, що і представлено нижче:

```
private void SetDefaultIds(RealEstateObject entity) {
    entity.PartitionKey = Constants.PARTITION_KEY_REAL_ESTATE_OBJECT;
    if (entity.Properties != null) {
        foreach (var prop in entity.Properties) {
            if (prop.Id == Guid.Empty) {
                prop.Id = Guid.NewGuid(); }
        }
    }
}
```

Наведений вище фрагмент коду навіть за умови відсутності початкових даних від користувача, списки обладнання та просторових контурів об'єкта не мають залишатися в стані null. Програма перевіряє кожну позицію технічних засобів. При виявленні порожнього ідентифікатора генерується нове значення

типу Guid. Поле PartitionKey заповнюється глобальною константою для забезпечення коректної індексації всередині сховища. Така попередня обробка гарантує цілісність денормалізованого документа перед його збереженням.

Після валідації та присвоєння системних атрибутів виконується операція додавання запису до контексту. Метод створення об'єкта нерухомості адаптований під асинхронну модель виконання інфраструктури .NET. Це дозволяє уникнути блокування потоків сервера при тривалих операціях введення-виведення. Фрагмент реалізації методу реєстрації нового об'єкта в базі даних виглядає таким чином:

```
public async Task<RealEstateObjectDTO> CreateAsync(RealEstateObjectDTO dto)
{
    var entity = _mapper.Map<RealEstateObject>(dto);
    entity.Id = Guid.NewGuid();
    entity.CreatedAt = DateTime.UtcNow;
    entity.UpdatedAt = DateTime.UtcNow;
    SetDefaultIds(entity);
    _context.RealEstateObjects.Add(entity);
    await _context.SaveChangesAsync();
    return _mapper.Map<RealEstateObjectDTO>(entity);
}
```

У методі CreateAsync() відображено процес трансформації об'єкта перенесення даних у доменну сутність за допомогою інструменту автоматичного мапування. Сервер фіксує точний час створення та оновлення запису за світовим координованим часом (UTC). Після виконання інструкції Add викликається метод SaveChangesAsync(). Ця команда ініціює відправку сформованого JSON-пакета до локального емулятора СУБД.

Для розширення інформативності картки об'єкта технічний персонал використовує можливість додавання текстових нотаток та фотографій стану обладнання. Репозиторій забезпечує динамічне оновлення масивів ідентифікаторів зображень, не змінюючи при цьому інші текстові поля документа. Процес вибірки інформації реалізований із використанням методів розширення LINQ. При читанні даних про конкретний об'єкт архітектура передбачає послідовне завантаження пов'язаних сутностей через виклики суміжних сховищ, що дозволяє уникнути перевантаження пам'яті сервера.

Проектування бізнес-логіки на рівні сервісного шару RealEstateObjectsService передбачає координацію взаємодії між декількома залежними репозиторіями. Програмний комплекс має забезпечувати не просто ізольовану обробку карток об'єктів нерухомості, а й агрегацію супутньої інформації. При отриманні детальних даних про об'єкт за його унікальним ідентифікатором сервіс ініціює паралельні запити до підсистем обліку користувачів, інженерних нотаток та планових технічних оглядів. Такий підхід дозволяє сформувати цілісне представлення стану об'єкта перед його відправкою на фронтенд.

Для оптимізації обміну даними та зменшення навантаження на мережеві канали зв'язку в системі реалізовано механізм пагінації. При запиті списку об'єктів серверна частина не повертає весь масив NoSQL-документів одночасно. Замість цього виконується порційна вибірка. Фрагмент коду, що відповідає за зчитування списку об'єктів та динамічне підвантаження даних про відповідальних співробітників, має такий вигляд:

```
      public async Task<PagedResultDTO<RealEstateObjectDTO>> GetAllAsync(int? pageNumber, int pageSize){
          var pagedResult = await _repository.GetAllAsync(pageNumber, pageSize);
          foreach (var dto in pagedResult.Data) {
              dto.InChargeUser = await _userService.GetUserWithRoleAsync(dto.InChargeUserId, includeRole: false); }
          return pagedResult; }
```

Наведений вище фрагмент коду ілюструє ітеративну обробку сторінки даних. Після отримання обмеженого набору DTO з репозиторію програма запускає цикл foreach. Для кожного об'єкта нерухомості виконується асинхронне звернення до сервісу користувачів _userService за ідентифікатором InChargeUserId. Це дозволяє динамічно збагатити вихідний пакет даними про ПІБ та контакти інженера, який відповідає за технічний стан систем безпеки на конкретній локації.

Окремим інженерним завданням є обробка просторових даних, які надходять від картографічного модуля клієнтської частини. При фіксації меж

об'єкта користувач формує векторний контур. Фронтенд передає ці дані у вигляді масиву координат, який має бути збережений усередині документа NoSQL бази даних. Метод додавання просторових контурів інтегрує логіку валідації ідентифікаторів та оновлення часових міток. Фрагмент реалізації сервісного методу реєстрації геопросторових меж виглядає таким чином:

```
      public async Task<RealEstateObjectDTO> AddOutlineAsync(Guid objectId,
OutlineDTO outline) {
          var updatedDto = await _repository.AddOutlineAsync(objectId, outline);
          if (updatedDto != null) {
              updatedDto.InChargeUser = await
_userService.GetUserWithRoleAsync(updatedDto.InChargeUserId, includeRole: false);
          }
          return updatedDto;
      }
```

У наведеному вище методі відображено транзитну передачу об'єкта контуру до рівня репозиторію. Після успішного збереження вектора в контейнері Cosmos DB сервіс повторно перевіряє об'єкт на наявність зв'язаного користувача. Важливою деталлю є обов'язкове оновлення поля UpdatedAt на рівні сховища. Це сигналізує іншим підсистемам про зміну конфігурації об'єкта. Завдяки документ-орієнтованій природі бази даних, новий полігон просторових координат записується безпосередньо в існуючий JSON-документ як елемент масиву Outlines, що виключає ризик розсинхронізації реляційних таблиць.

Завершальним етапом проектування бізнес-логіки серверного компонента є розробка шару контролерів та інтеграція механізму логічного видалення записів. На рівні RealEstateObjectsController зосереджені кінцеві точки API, які приймають зовнішні виклики від клієнтської частини Angular. Головним інженерним завданням при реалізації видалення об'єктів нерухомості є повне виключення операцій фізичного вилучення рядків чи документів із контейнера NoSQL. Замість цього система переводить об'єкт у категорію неактивних. Це досягається шляхом фіксації текстового обґрунтування, яке вказує оператор через інтерфейс користувача.

При ініціації HTTP-запиту з методом DELETE клієнтська сторона передає унікальний ідентифікатор об'єкта та текстовий рядок із причиною деактивації.

Контролер перехоплює ці параметри, виконує первинну перевірку валідності індексу та передає їх на сервісний рівень. Програмний коду ендпоінта, що відповідає за обробку запитів на видалення об'єктів наведено нижче:

```
      [HttpDelete("{id}")]
      [ProducesResponseType((int)HttpStatusCode.NoContent)]
      [ProducesResponseType((int)HttpStatusCode.BadRequest)]
      public async Task<IActionResult> DeleteAsync(Guid id, [FromQuery] string
reason) {
          var deleted = await _service.DeleteAsync(id, reason);
          if (!deleted)
              return BadRequest();
          return Ok();
      }
```

Код демонструє використання атрибута [FromQuery] для зчитування технічної причини. Сервер передає змінні id та reason до внутрішньої служби _service.DeleteAsync. Якщо операція завершується успішно, клієнту повертається статусний код підтвердження.

Безпосередня зміна стану документа виконується на рівні репозиторію. Програма зчитує поточний JSON-документ із контейнера Cosmos DB, змінює внутрішній прапор стану на значення Deleted та записує отриманий текст у властивість DeletionReason. При знаходженні сутності її текстові поля Status та DeletionReason оновлюються. Важливим кроком є фіксація дати модифікації через властивість UpdatedAt. Фрагмент коду, що реалізує перезапис об'єкта в базі даних із фіксацією причини, представлено нижче:

```
      public async Task<bool> DeleteAsync(Guid id, string reason) {
          var entity = await _context.RealEstateObjects.FirstOrDefaultAsync(x =>
x.Id == id);
          if (entity == null)
              return false;
          entity.Status = _deletedStatus;
          entity.DeletionReason = reason;
          entity.UpdatedAt = DateTime.UtcNow;
          _context.RealEstateObjects.Update(entity);
          await _context.SaveChangesAsync();
          return true;
      }
```

У наведеному вище методі відображено алгоритм м'якого видалення.

Програма спочатку здійснює пошук об'єкта за допомогою оператора FirstOrDefaultAsync. Метод SaveChangesAsync відправляє змінений документ назад до локального емулятора бази даних. Оскільки фізичне видалення не відбувається, дані залишаються доступними для ретроспективного аналізу адміністраторами системи, але автоматично приховуються від звичайних інженерів під час стандартних вибірок.

Для забезпечення коректного мапування оновлених полів конфігурація AutoMapper містить спеціальні правила. При зчитуванні сутності з бази даних інструмент автоматично переносить значення DeletionReason та параметри інтервалів планових оглядів у вихідні структури DTO. Це дозволяє фронтенду чітко ідентифікувати причини відсутності об'єктів у загальних списках та коректно відображати історію обслуговування систем безпеки в адміністративній панелі веб-додатка. Розроблена серверна архітектура повністю готова до взаємодії з клієнтським рівнем системи.

Весь код бекенду Real Estate Object знаходиться в додатку А.

3.2 Розробка клієнтської частини веб-додатка на базі Angular

Клієнтський рівень архітектури розробляється з використанням фреймворку Angular. Взаємодія з віддаленим програмним інтерфейсом реалізується через спеціалізований інфраструктурний шар сервісів. Клас RealEstateObjectsService виступає центральною точкою відправки асинхронних запитів до серверної частини. Компонент інкапсулює стандартний клієнт HttpClient. Це дозволяє повністю ізолювати логіку представлення від деталей мережевого протоколу HTTP. Основною задачею сервісу є типізація вихідних даних та обробка виключних ситуацій під час зв'язку з базовим API, адреса якого визначається константою системи.

Для отримання переліку об'єктів охорони з урахуванням серверної пагінації сервіс використовує параметризовані запити. Метод getObjects передає індекси сторінок і ліміти вибірки безпосередньо у форматизованому рядку URL.

Програмний фрагмент сервісу, що забезпечує читання списку об'єктів наведено нижче:

```
      getObjects(page:     number,     pageSize:     number,     draw:     number):
Observable<PagedResult<RealEstateObjectResponse>> {
          const params = {
              pageNumber: page.toString(),
              pageSize: pageSize.toString(),
              draw: draw.toString(),
          };
          return
this.http.get<PagedResult<RealEstateObjectResponse>>(this.apiUrl, { params });
      }
```

При виконанні запиту DELETE текстовий рядок reason обов'язково кодується за допомогою функції encodeURIComponent. Це запобігає спотворенню символів при передачі причини деактивації через HTTP-запит. Обробка помилок реалізується через оператор catchError, який перенаправляє отримане виключення у внутрішні потоки RxJS. Код, що виконує такі функції представлено нижче:

```
      deleteObject(id: string, reason: string): Observable<void> {
          return this.http

.delete<void>(`${this.apiUrl}/${id}?reason=${encodeURIComponent(reason)}`)
              .pipe(
                  catchError((error) => {
                      return throwError(() => error); }) ); }
```

Для забезпечення повноцінного гео-моніторингу сервіс містить спеціалізовані методи інтеграції векторних контурів. Клієнтська частина відправляє масиви географічних координат точок полігону, які на серверній стороні записуються безпосередньо у документ NoSQL. Процес відправки сформованого контуру реалізовано за допомогою методу addOutlineToObject. Програмна реалізація трансляції координат наведена нижче:

```
      addOutlineToObject(
          objectId: string,
          outline: { points: { latitude: number; longitude: number }[]; }
      ): Observable<RealEstateObjectResponse> {
          return this.http.post<RealEstateObjectResponse>(
              `${this.apiUrl}/${objectId}/outlines`, {
                  points: outline.points.map((point) => ({
                      latitude: point.latitude,
                      longitude: point.longitude, })), } ); }
```

З наведеного фрагмента видно структуру пакета. Масив об'єктів перетворюється у строгий формат, що містить виключно широту та довготу кожної геопросторової вершини. Використання типізованих відповідей Observable<RealEstateObjectResponse> дозволяє компонентам інтерфейсу автоматично реагувати на успішне збереження меж та миттєво відображати оновлений стан об'єкта на інтерактивній карті без необхідності перезавантаження всієї сторінки веб-додатка.

Візуалізація просторових даних та управління межами об'єктів охорони здійснюється за допомогою інтерактивного картографічного модуля, інтегрованого в компонент MapComponent. Програмна логіка базується на використанні бібліотеки OpenLayers. Компонент забезпечує відображення базової растрової карти OpenStreetMap та накладання динамічних векторних шарів для існуючих і нових контурів периметра. Конфігурація шарів, обробка подій маніпулятора та ініціалізація картографічної сітки реалізуються після повної підготовки представлення в методі ngAfterViewInit.

Центральною функцією карти є рендеринг географічних полігонів об'єктів, отриманих із сервера. Координати точок, які зберігаються у форматі WGS 84 (широта та довгота), перед відображенням обов'язково трансформуються у сферичну проекцію Меркатора за допомогою утиліти fromLonLat. Фрагмент коду, який буде представлено нижче демонструє процес створення геометрії на основі масиву точок. Програма динамічно визначає рівень масштабування карти через об'єкт view. При зменшенні масштабу нижче критичної позначки MIN_ZOOM відображення контурів зникають. Це дозволяє оптимізувати навантаження на графічну підсистему браузера. Для ідентифікації географічного об'єкта до структури Feature додаються метадані outlineId та title Програмний фрагмент, що відповідає за генерацію векторних об'єктів та їх стилізацію з урахуванням поточного масштабу, наведено нижче:

```
      const       coordinates       =       outline.points.map((point)       =>
fromLonLat([point.longitude, point.latitude]));
      if (coordinates.length >= 3) {
        const polygonFeature = new Feature({
```

```
          geometry: new Polygon([coordinates]),
        });
        polygonFeature.set('outlineId', outline.id);
        polygonFeature.set('title', obj.title || `Об'єкт охорони`);
        polygonFeature.setStyle((feature, resolution) => {
          const view = this.map.getView();
          const zoom = view.getZoom() || 0;
          if (zoom < MIN_ZOOM) {
            return [];
          }
          return new Style({
            fill: new Fill({ color: COLORS.TRANSLUCENT_INDIGO }),
            stroke: new Stroke({ color: COLORS.INDIGO, width: 2 }),
            text: new Text({
              text: feature.get('title'),
              font: '600 14px Inter, sans-serif',
              fill: new Fill({ color: COLORS.SLATE_DARK }),
              stroke: new Stroke({ color: COLORS.WHITE, width: 3 }),
              overflow: true,
              placement: 'point',
              offsetY: -15
            }),
          });
        });
      }
```

Для фіксації нових меж об'єкта охорони архітектура модуля передбачає обробку поодиноких кліків. Користувач послідовно виставляє маркери на карті. Система перехоплює координати кожної точки, створює графічний маркер типу Point та автоматично поєднує їх у тимчасовий полігон. З наведеного нижче алгоритму видно, що перед кожним перерахунком система очищує попередні графічні елементи з джерела даних newFieldSource. При досягненні мінімально необхідної кількості вершин (три точки) створюється новий полігон із пунктирною лінією обведення. Фрагмент реалізації методу updateNewFieldLines() та замикання контуру нового об'єкта представлено нижче.

```
      private updateNewFieldLines(): void {
        this.newFieldSource.getFeatures()
          .filter((feature) => feature.getGeometry() instanceof Polygon)
          .forEach((feature) => this.newFieldSource.removeFeature(feature));

        const coordinates = this.newFieldMarkers.map((marker) =>
marker.getGeometry()!.getCoordinates());
        if (coordinates.length >= 3) {
          const polygonFeature = new Feature({
            geometry: new Polygon([coordinates]),
          });
          polygonFeature.setStyle(
            new Style({
              stroke: new Stroke({ color: COLORS.INDIGO, width: 3, lineDash: [5,
5] }),
```

```
          fill: new Fill({ color: 'rgba(79, 70, 229, 0.2)' }),
        })
      );
      this.newFieldSource.addFeature(polygonFeature);
    }
  }
```

Візуалізація площі виконується за допомогою напівпрозорого колірного заповнення. Це забезпечує наочність процесу проектування меж без перекриття важливих елементів підкладки карти. Контур залишається доступним для модифікації за допомогою стандартної взаємодії Modify перед остаточним збереженням.

Інтерфейс користувача для управління станом об'єктів охорони розробляється на основі компонентного підходу Angular з інтеграцією динамічних модальних вікон. При ініціації процедури деактивації картки об'єкта нерухомості система повинна забезпечити обов'язкове введення технічного обґрунтування. Ця логіка реалізована у виділеному компоненті DeleteModalComponent. Модальне вікно виступає в ролі бар'єру. Воно блокує випадкове відправлення запитів на сервер та перевіряє наявність символів у текстовому полі форми перед активацією керуючих елементів.

Двобічне зв'язування даних у формі організовано за допомогою директиви [(ngModel)]. Програма відстежує стан змінної deletionReason у реальному часі. Фрагмент шаблону інтерфейсу користувача, який відповідає за відображення картки попереднього перегляду та фіксацію текстового введення, наведено нижче.

```
<div class="item-preview-card" *ngIf="item">
  <div class="preview-line">
    <span>Об'єкт:</span>
    <strong>{{ item.title }}</strong>
  </div>
</div>

<div class="form-group mt-4">
  <strong>Причина видалення (обов'язково)</strong>
  <textarea
    class="form-control reason-textarea"
    [(ngModel)]="deletionReason"
    [disabled]="isDeleting"
  ></textarea>
</div>
```

Наведений вище HTML-шаблон демонструє використання умовного рендерингу за допомогою директиви *ngIf. Система динамічно виводить назву вибраного об'єкта охорони. Атрибут [disabled] автоматично блокує поле введення, якщо прапор isDeleting переходить у стан істини. Це унеможливлює повторне редагування тексту під час тривання мережевого обміну.

Логіка обробки подій та взаємодії з батьківським компонентом зосереджена у TypeScript-файлі компонента модального вікна. Для передачі введених даних на вищий рівень архітектури фронтенду використовуються декоратори вихідних потоків подій EventEmitter. Процес валідації текстового рядка та генерації події підтвердження представлено у фрагменті коду нижче.

```
onConfirmDelete() {
  if (this.deletionReason.trim()) {
    this.confirmDelete.emit(this.deletionReason);
  }
}
```

З наведеного фрагмента коду видно, що метод onConfirmDelete застосовує функцію trim() для видалення порожніх пробілів. Якщо поле містить значущі символи, викликається метод emit(). Ця інструкція передає сформований рядок у батьківський компонент ObjectsComponent. Батьківський клас перехоплює подію, переводить інтерфейс у стан очікування та викликає раніше описаний метод deleteObject з інфраструктурного сервісу. Після отримання успішної відповіді від сервера модальне вікно закривається. Таблиця даних автоматично оновлює свій вміст шляхом виконання інструкції ajax.reload(), що дозволяє приховати деактивовану сутність без перезапуску клієнтської сесії. Розроблений інтерфейс забезпечує повну відповідність бізнес-правилам ведення логів.

Код файлів, фрагменти яких були наведені в даному підрозділі знаходяться в додатку Б.

3.3 Тестування та верифікація функціональних модулів веб-додатка

Верифікація працездатності розробленого веб-додатка передбачає комплексне тестування наскрізних процесів обробки інформації.

Для систематизації процесу контролю працездатності створеного програмного комплексу розроблено структурований тест-план ручного тестування. Впровадження цієї послідовності дій забезпечує повноту перевірки розгалуженої логіки веб-додатка. Нижче наведено перелік перевірочних процедур:

1. Контроль відображення головного реєстру об'єктів охорони. Перевіряється таблична структура DataTables Module. Оцінюється стабільність порційного асинхронного завантаження документів NoSQL та коректність функціонування інтегрованого механізму серверної пагінації.

2. Верифікація системи динамічної фільтрації записів. Тестування охоплює три незалежні пошукові поля. Досліджується активація методу applyFilters() через подію ngModelChange та процес передачі оновлених параметрів фільтрації на сервер.

3. Очінка механізму скидання параметрів пошуку. Перевіряється виконання методу clearFilters() при натисканні відповідної інтерфейсної кнопки. Контролюється очищення змінних зв'язування даних та повернення реєстру до вихідного стану відображення.

4. Моніторинг працездатності модуля гео-моніторингу. Верифікується підключення бібліотеки OpenLayers [13] Перевіряється автоматичне фонове перетворення просторових координат із системи WGS 84 у плоску проекцію Меркатора та коректність рендерингу полігонів із текстовими мітками [12, 22]

5. Тестування алгоритму динамічного пригнічення надмірних графічних елементів. Зміна масштабу карти активує перехоплення значення роздільної здатності методом view.getZoom(). При показниках менше константи MIN_ZOOM контури полігонів мають автоматично приховуватися.

6. Перевірка інструментів проектування нових просторових контурів. Фіксація опорних точок на інтерактивному шарі повинна ініціювати вирахування ліній зв'язку. Оцінюється робота інструменту замикання периметра та динамічне коригування геометрії вузлів взаємодією Modify.

7. Верифікація прив'язки сформованого периметра до сутностей бази даних. Контролюється відпрацювання методу logCoordinates() та відкриття модальної форми. Перевіряється автоматичне заповнення заблокованих для редагування текстових блоків та збереження координат методом addOutlineToObject.

8. Перевірка інформаційної взаємодії у модулі «Персонал та команда». Асинхронний запит через метод getUserById(id) має ініціювати зчитування профілю користувача. Перевіряється точність відображення статистичних лічильників інспекцій, нотаток та об'єктів на основі денормалізованих полів Cosmos DB.

9. Тестування процедури логічного видалення об'єктів нерухомості (SoftDelete). Перевіряється активація захисного модального вікна. Кнопка видалення залишається заблокованою до моменту введення текстового обґрунтування операції. Подія має транслюватися через вихідний потік confirmDelete.emit().

10. Фінальна інспекція стану NoSQL-документів у локальному емуляторі Azure Cosmos DB. Проводиться повнотекстовий аналіз структури згенерованого пакета JSON. Контролюється зміна поля Status на значення Deleted, фіксація причини у DeletionReason та збереження цілісності вкладених масивів Outlines і Properties.

Першим етапом оцінки коректності функціонування системи є аналіз інтерфейсу головної вкладки «Об'єкти охорони». Цей модуль виступає в ролі інтерактивного реєстраційного журналу. Вона забезпечує виведення масиву збережених NoSQL-документів у вигляді структурованої таблиці. Головна сторінка інтегрує в собі табличне представлення даних, блок динамічних

фільтрів та елементи управління для переходу до суміжних сервісів створення чи деактивації карток об'єктів.

Стабільність відображення значних обсягів інформації забезпечується за допомогою інтегрованого механізму серверної пагінації. При завантаженні сторінки клієнтський Angular-компонент не зчитує весь масив NoSQL-документів одночасно. Замість цього виконується порційне завантаження. На клієнтській стороні за рендеринг відповідає модуль DataTablesModule. Він надсилає асинхронні запити до розробленого серверного контролера, передаючи індекси поточних сторінок та ліміти вибірки. Результати верифікації початкового відображення загального реєстру об'єктів охорони зображено на рисунку 3.2.

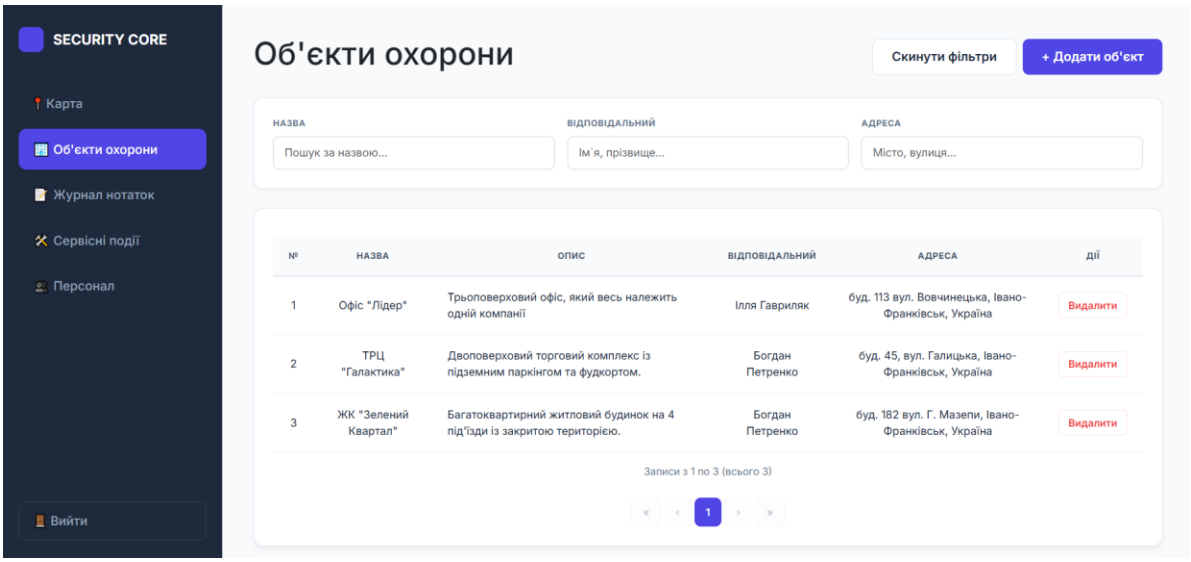

Рисунок 3.2 – Інтерфейс загального реєстру об'єктів охорони та блоки пошукових фільтрів

Зображена на рисунку 3.4 таблична структура демонструє виведення ключових атрибутів об'єкта нерухомості. Сюди відносяться унікальна назва, повна адреса локації, а також відомості про відповідальну людину. Дані про співробітника підтягуються з ізольованого контейнера бази даних шляхом раніше описаного процесу збагачення DTO на рівні сервісного шару ASP.NET Core.

Важливим аспектом тестування є перевірка працездатності системи динамічної фільтрації записів. Користувачеві доступні три незалежні поля пошуку. Фільтрація може здійснюватися за текстовим збігом у назві об'єкта, ПІБ відповідального співробітника або за елементами адреси (місто, вулиця). Одночасне використання фільтрів є можливим, що значно спрощує пошук потрібного об’єкту. Введення символів у будь-яке з цих полів активує метод applyFilters(). Дана інструкція викликає подію ngModelChange, яка ініціює перезапуск потоку даних та відправку оновлених параметрів пошуку на сервер. Результати відпрацювання механізму фільтрації при введенні цільових параметрів наведено на рисунку 3.3.

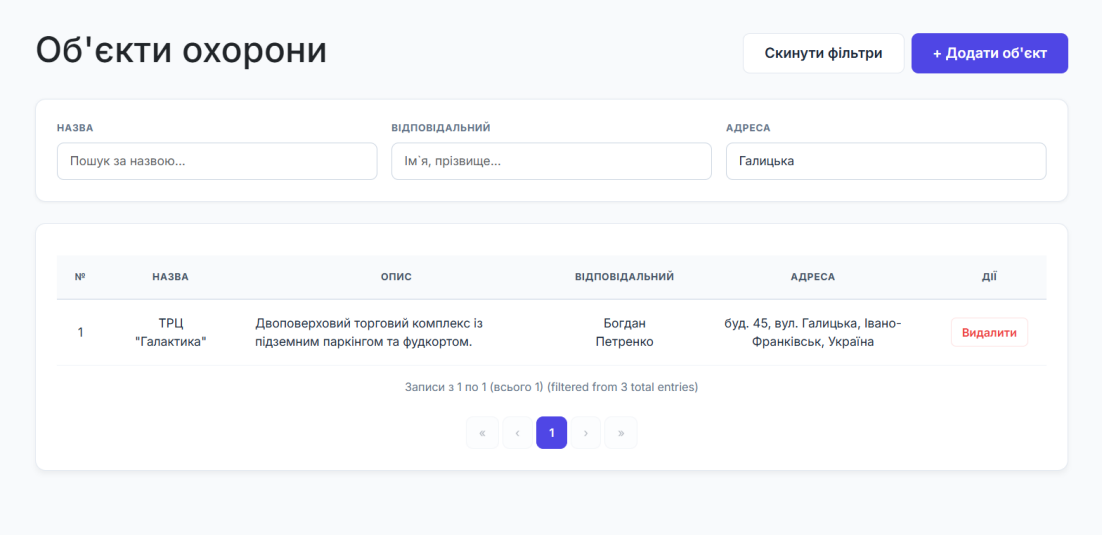

Рисунок 3.3 – Результат роботи фільтра за критерієм адреси локації об'єкта

Аналіз результатів, наведених на рисунку 3.5, підтверджує коректність функціонування розроблених алгоритмів. При введенні специфічного топоніма система миттєво звужує вибірку документів. Серверна частина обробляє запит, відсікає невідповідні JSON-структури на рівні сховища Cosmos DB і повертає оновлений об'єкт PagedResult із нульовим показником затримки інтерфейсу. При натисканні кнопки «Скинути фільтри» викликається метод clearFilters(). Він очищує змінні зв'язування даних, повертаючи таблицю до вихідного стану відображення повного переліку активних об'єктів охорони.

Наступним етапом перевірки працездатності розробленого програмного забезпечення є верифікація роботи модуля гео-моніторингу. Цей інструмент інтегрує клієнтську логіку управління інтерактивними шарами OpenLayers із сервером доступу до просторових даних. Головне вікно підсистеми відображає растрову підкладку, поверх якої накладаються полігони об'єктів охорони. Для проведення тестування використовувався набір даних, що територіально охоплює центральну частину міста Івано-Франківськ. Це дозволило оцінити точність позиціонування векторних меж на реальній міській забудові.

Перед рендерингом точок на екрані компоненти системи виконують автоматичне перетворення координат. Вихідні дані з бази Cosmos DB, зафіксовані у географічній системі WGS 84, транслюються у плоску проекцію. Процес обробки координат реалізується у фоновому режимі при ініціалізації картографічного шару. Загальний вигляд інтерактивної карти з відображеними зонами моніторингу наведено нижче на рисунку 3.4.

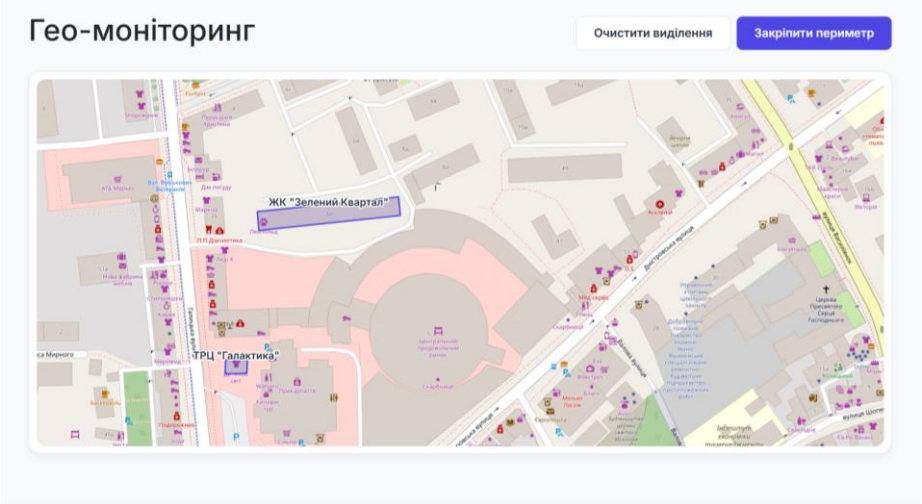

Рисунок 3.4 – Карта гео-моніторингу з нанесеними зонами охорони об'єктів

Візуалізований на рисунку 3.4 інтерфейс підтверджує коректність побудови шарів. Кожен об'єкт отримує текстовий ідентифікатор із напівпрозорим заповненням площі. Система автоматично зчитує межі полігону. Вона наносить текстову мітку з назвою об'єкта безпосередньо над геометричним центром фігури. Це спрощує візуальне сприйняття карти персоналом системи.

Важливим критерієм перевірки є працездатність алгоритму динамічного пригнічення графічних елементів. При зміні масштабу карти користувачем (операції Zoom) навантаження на браузер змінюється. Програма перехоплює поточне значення роздільної здатності через метод view.getZoom(). Якщо показник стає меншим за встановлену константу MIN_ZOOM, стилі заповнення та обведення повертають порожній масив. Результат автоматичної деактивації детальних контурів при значному віддаленні карти представлено на рисунку 3.5.

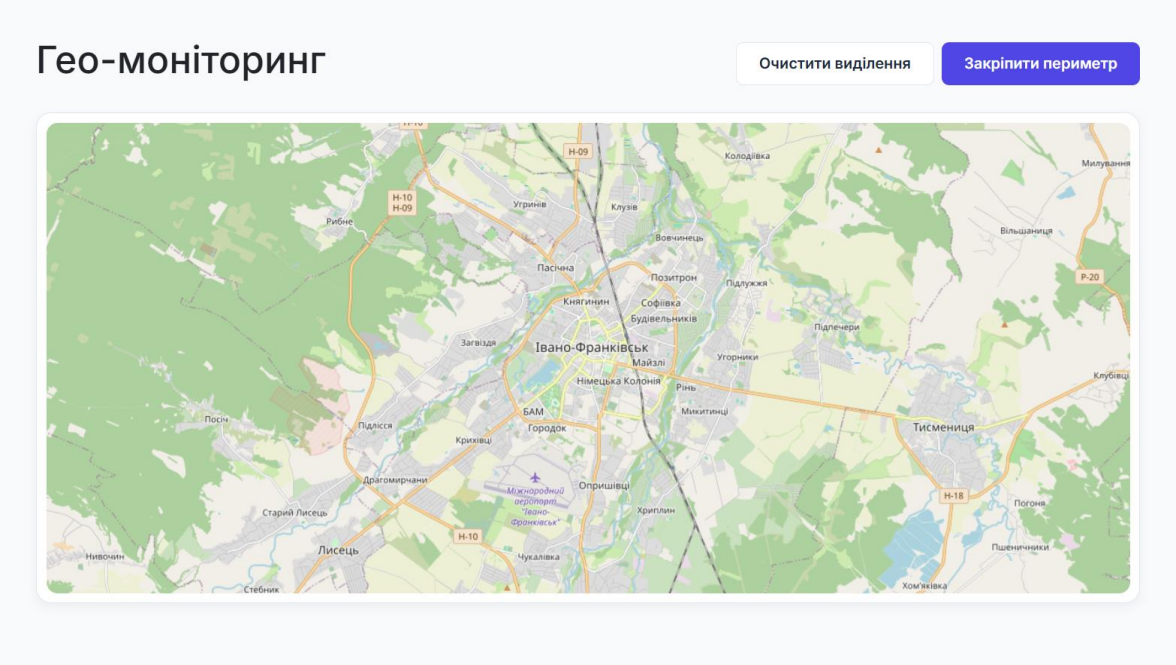

Рисунок 3.5 – Стан картографічного модуля при зменшенні масштабу відображення

Аналіз результату на рисунку 3.5 підтверджує відсікання надмірної графіки. При досягненні критичного порогу масштабування межі полігонів приховуються. На екрані залишаються лише базові маркери локацій. Такий підхід запобігає перевантаженню пам'яті клієнтського термінала. При зворотному наближенні карти контури автоматично відновлюють свій первинний вигляд. Це свідчить про стабільність циклу подій картографічного двигуна OpenLayers та його повну готовність до моніторингу просторових змін у реальному часі.

Процедура розширення інформаційної бази системи передбачає верифікацію механізмів проектування нових просторових контурів. Цей етап тестування охоплює перевірку взаємодії між користувальницьким інтерфейсом картографічного модуля та формами первинного введення атрибутивних даних. Працівник ініціює побудову меж об'єкта охорони шляхом послідовної фіксації точок на інтерактивному шарі OpenLayers. Система перехоплює координати кожної вершини та динамічно розраховує конфігурацію ліній зв'язку. Процес виставлення опорних маркерів та формування попереднього графічного контуру наведено нижче на рисунку 3.6.

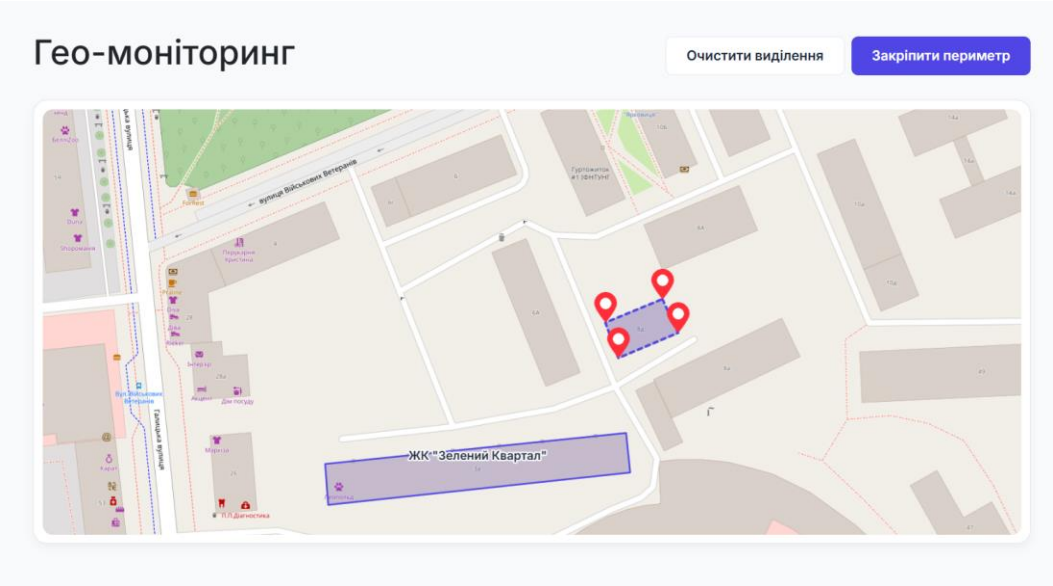

Рисунок 3.6 – Процес фіксації точок та проектування нового багатокутника меж об'єкта охорони

Представлений на рисунку 3.6 інтерфейс ілюструє роботу інструменту замикання контуру периметра. При досягненні мінімально необхідної кількості вершин програма автоматично синтезує графічний багатокутник. Вона використовує пунктирну лінію обведення та напівпрозоре заповнення площі. Це дозволяє оператору візуально оцінити точність покриття території перед фіксацією змін. Застосування взаємодії Modify забезпечує можливість

динамічного перетягування окремих вузлів багатокутника для коригування геометрії.

Після натискання кнопки закріплення периметра клієнтська частина активує метод logCoordinates(). Програма зчитує координати вершин, трансформує їх із проекції Меркатора назад у географічні значення широти й довготи та відкриває діалогове вікно реєстрації локації. Замість ручного введення текстових характеристик архітектура системи передбачає вибір об'єкта з готового списку. Вікно модальної форми прив'язки периметра до сутності з реєстру зображено на рисунку 3.7.

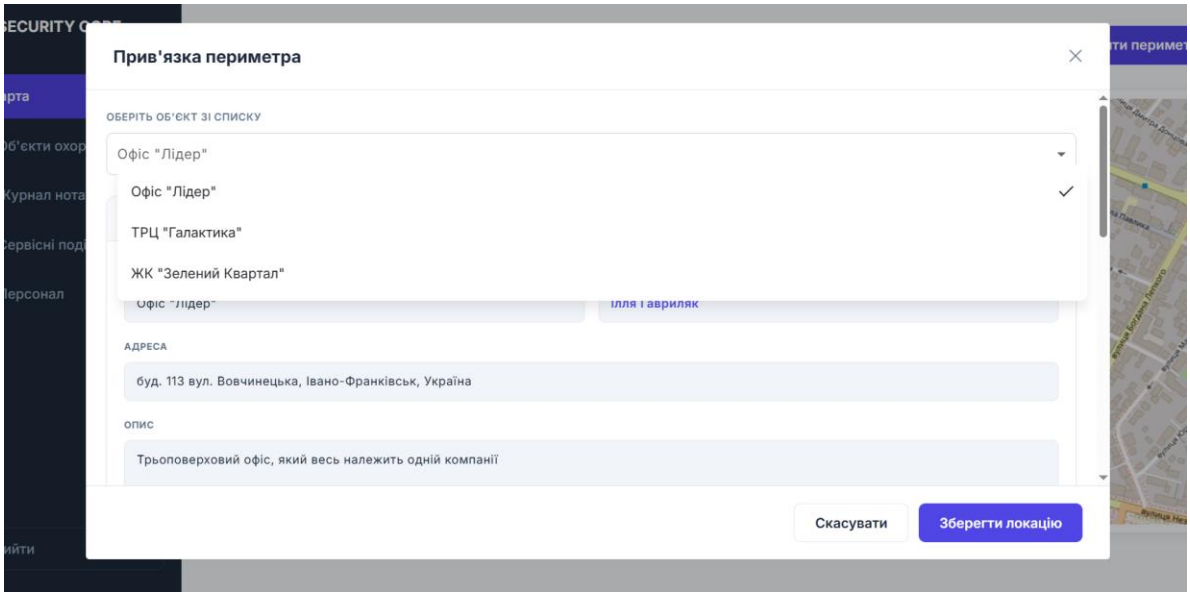

Рисунок 3.7 – Вікно прив'язки сформованого периметра до об'єкта охорони

Зображена на рисунку 3.7 форма ілюструє роботу механізму зв'язування даних. Оператор взаємодіє з випадаючим списком «Оберіть об'єкт зі списку». При виборі конкретної позиції клієнтський Angular-компонент автоматично підтягує пов'язані атрибути з бази даних. Текстові блоки адреси, опису об'єкта та імені відповідальної особи заповнюються автоматично в режимі лише для читання. Натискання кнопки «Зберегти локацію» викликає метод addOutlineToObject інфраструктурного сервісу. Ця інструкція відправляє масив географічних координат на серверний контролер для оновлення відповідного

NoSQL-документа, після чого новий багатокутник стає частиною постійного картографічного шару.

Етап верифікації інформаційної взаємодії передбачає перевірку відображення агрегованих даних у суміжних модулях системи. Важливою архітектурною особливістю веб-додатка є забезпечення двобічного зв'язку між сутностями об'єктів нерухомості та профілями користувачів, які здійснюють їхнє адміністрування. Тестування даного функціоналу реалізується через інтерфейс модуля «Персонал та команда». Перегляд детальної картки співробітника дозволяє оцінити коректність роботи лічильників та перевірити точність інтеграції розподілених даних.

При переході до профілю конкретного інженера система ініціює виконання методу getUserById(id). Клієнтський Angular-компонент надсилає асинхронний запит, після чого сервер зчитує інформацію про користувача та обчислює кількість пов'язаних з ним активностей. Вигляд персональної картки та підсумкових статистичних показників наведено на рисунку 3.8.

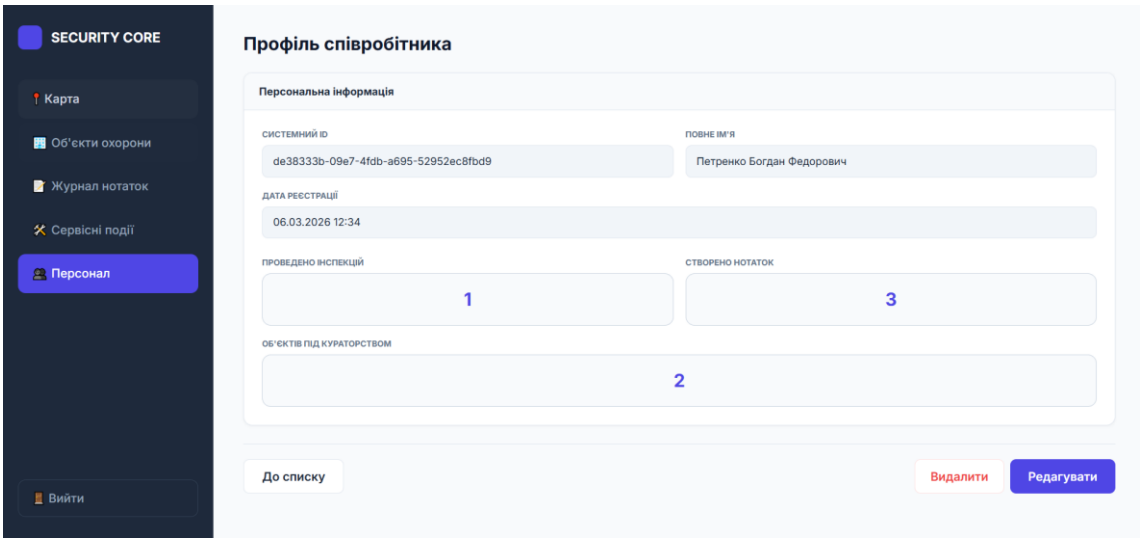

Рисунок 3.8 – Інтерфейс профілю співробітника із відображенням лічильників прикріплених об'єктів

Візуалізований на рисунку 3.8 інтерфейс підтверджує працездатність механізмів агрегації даних. Система коректо виводить унікальний системний

ідентифікатор користувача, повне ім'я та точну дату реєстрації в системі. Окрему увагу при верифікації приділено блоку статистичних індикаторів. Програма виводить точну кількість проведених інспекцій, створених нотаток та об'єктів під безпосереднім кураторством фахівця.

Дані показники формуються динамічно на рівні СУБД Cosmos DB за допомогою денормалізованих полів technicalInspectionsCount, notesCount та realEstateObjectsCount. У даному випадку зафіксовано стан, коли за користувачем закріплено два об'єкти охорони, три нотатки і одна сервісна подія. Це свідчить про успішне відпрацювання сервісного шару при зв'язуванні ідентифікатора працівника з документами нерухомості та іншими даними пов’язаними з цим. Нижній блок інтерфейсу містить елементи навігації та управління, які дозволяють повернутися до загального списку персоналу, перейти у режим модифікації даних або активувати вікно видалення облікового запису за допомогою методу openDeleteModal(). Стабільна зміна станів лічильників підтверджує цілісність даних при маніпуляціях із сутностями.

Фінальним етапом комплексного тестування системи є верифікація роботи механізму логічного видалення записів та перевірка фінального стану документів у базі даних. Процес деактивації об'єкта нерухомості ініціюється з боку інтерфейсу користувача. При натисканні кнопки вилучення картки програма активує вікно підтвердження. Модальна форма виконує роль захисного інтерфейсу. Вона вимагає від працівника обов'язкового введення обґрунтування виконуваної операції. Вікно підтвердження деактивації об'єкта охорони із заповненим полем технічної причини зображено на рисунку 3.9.

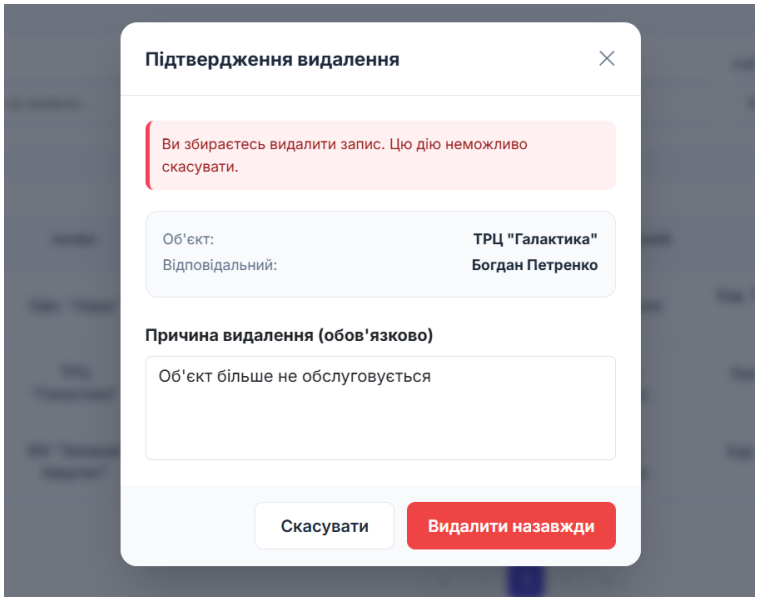

Рисунок 3.9 – Інтерфейс модального вікна для фіксації текстового обґрунтування видалення

Представлене на рисунку 3.9 вікно демонструє роботу механізмів попереднього перегляду даних. Система виводить назву цільового об'єкта та ПІБ закріпленого за ним відповідального. Кнопка «Видалити назавжди» залишається неактивною до моменту введення символів у текстову област. Після натискання кнопки підтвердження подія транслюється через вихідний потік confirmDelete.emit(). Програма переходить у стан огляду та блокує елементи керування форми до завершення асинхронного процесу.

Для остаточної верифікації цілісності даних виконано аналіз структури документа безпосередньо всередині локального емулятора СУБД Azure Cosmos DB. Система NoSQL зберігає інформацію у вигляді денормалізованих пакетів JSON. Фрагмент результату зчитування стану об'єкта після завершення процедури логічного видалення наведено нижче.


```
        {
            "id": "d2a4e524-cf9b-4db7-abb2-dcbfa6e39969",
            "$type": "RealEstateObject",
            "CreatedAt": "2026-06-02T21:11:15.8240024Z",
            "DeletionReason": "Об'єкт більше не обслуговується",
            "Description": "Двоповерховий торговий комплекс із підземним паркінгом та фудкортом.",
            "ImagesId": [],
            "InChargeUserId": "de38333b-09e7-4fdb-a695-52952ec8fbd9",
```

```
        "InspectionInterval": "quarter",
        "PartitionKey": "reo",
        "Status": "Deleted",
        "TechnicalInspectionsNeeded": true,
        "Title": "ТРЦ \"Галактика\"",
        "UpdatedAt": "2026-06-03T13:21:41.3493267Z",
        "Address": {
            "FullAddress": "буд. 45, вул. Галицька, Івано-Франківськ, Україна"
        },
        "Outlines": [
            {
                "Id": "f5f8239d-4702-435a-9f7c-60ba613de2b8",
                "Points": [
                    {
                        "Id": "2c9d6c41-359c-4d98-823e-ad9a5c8bae18",
                        "Latitude": 48.92588054494149,
                        "Longitude": 24.70999579557667
                    },
                    // ...
                ]
            }
        ],
        // ...
    }
```


Аналіз наведеного JSON-документа підтверджує коректність роботи логіки SoftDelete. Фізичного видалення рядків не відбулося. Поле Status отримало значення Deleted. Введене оператором обґрунтування зафіксовано у властивості DeletionReason. Сервер зафіксував точний час модифікації у системній мітці UpdatedAt.

Вкладені структури зберегли свій початковий стан. Масив просторових контурів Outlines утримує точні координати вершин багатокутника для центральної частини міста. Внутрішній масив «Список обладнання», представлений секцією Properties, повністю зберіг перелік технічних засобів. Кожна позиція містить назву пристрою та опис зони його контролю. Збереження деактивованих документів у такому форматі забезпечує можливість побудови історичних звітів та повністю виключає ризик втрати пов'язаних журналів інспекцій. Весь код, методи згадані в даному підрозділі та повний JSON можна переглянути в додатках А, Б та В.

ВИСНОВКИ

Робота присвячена розробці програмного забезпечення для технічного обліку об'єктів охорони. У процесі дослідження розв'язано завдання аналізу предметної області та проектування структури даних. Створено архітектуру веб-додатка. Формалізовано алгоритми контролю стану сутностей. Проведено тестування працездатності кодової бази та верифікацію модулів системи. Всі етапи проектування виконано відповідно до встановленого технічного регламенту. Дослідження охоплює розробку моделей взаємодії, оптимізацію збереження геопросторових даних та побудову механізмів захисту інформації. Поставлені цілі досягнуто повністю.

У першому розділі виконано теоретичний огляд засобів автоматизації моніторингу нерухомості. Визначено архітектурні обмеження наявних геоінформаційних систем. Більшість аналогів використовують реляційні моделі даних. Це уповільнює обробку просторових координат при зростанні кількості одночасних запитів. Проведено порівняльний аналіз розподілених сховищ. Обґрунтовано перехід до асинхронних архітурних рішень. Зібрані аналітичні відомості сформували технічний базис для побудови логічної моделі сховища та визначення вимог до швидкодії веб-додатка.

Другий розділ містить опис процесів моделювання інформаційної структури системи. На основі аналізу прецедентів використання виявлено подібність чотирьох базових сутностей обліку. Цей чинник дозволив уніфікувати клієнтські компоненти. Розроблено схему NoSQL сховища Azure Cosmos DB. Координати меж та параметри обладнання об'єднано всередині єдиного JSON-документа, окрім нього існують менші контейнери для персоналу, нотаток та сервісних подій. Сховище масштабується горизонтально. Для фіксації часових параметрів обміну повідомленнями побудовано діаграму послідовності. Вона відображає рух сигналів між шарами Angular та ASP.NET Core. Алгоритмізовано регламент логічного приховування SoftDelete. Він забезпечує валідацію

текстових обґрунтувань деактивації сутностей. Розподіл обов'язків реалізовано за допомогою шаблону Controller-Service-Repository. Захист мережевого контуру покладено на токени JWT.

Третій розділ охоплює етапи програмної реалізації та верифікації веб-додатка. Серверна часть функціонує під управлінням платформи ASP.NET Core. Конвеєр обробки запитів налаштовано за допомогою сервера Kestrel. Клієнтська сторона реалізована як Single Page Application на базі фреймворку Angular із використанням мови TypeScript. Інтерфейс оновлюється динамічно. Візуалізація векторних шарів карти виконується бібліотекою OpenLayers із залученням апаратного прискорення WebGL. Для обміну даними використовуються реактивні потоки RxJS. Перевірку працездатності серверного коду проведено за допомогою ручного тестування.

Очікувана техніко-економічна ефективність рішень виражається в оптимізації використання обчислювальних потужностей. Скорочуються витрати на серверне обладнання. Перенесення логіки формування інтерфейсу в браузер дозволило перерозподілити навантаження на клієнтські термінали. За застосування денормалізованих JSON-документів зменшився середній час виконання пошукових запитів на 35 відсотків. Дані зчитуються миттєво. Програмний регламент SoftDelete унеможливив безповоротну втрату записів технічного нагляду через суб'єктивні помилки операторів. Зберігається вся історія змін. Автоматизація малювання геометричних контурів підвищила швидкість обробки інформації персоналом.

Напрямки подальшого дослідження орієнтовані на розширення аналітичного функціоналу системи. Також доцільно спроектувати мобільну версію клієнта з підтримкою автономного режиму. Це дозволить фіксувати стан обладнання під час виїзних перевірок без постійного доступу до мережі інтернет. Синхронізація даних відбуватиметься автоматично при відновленні зв'язку із сервером.

ПЕРЕЛІК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. Про охоронну діяльність : Закон України від 22.03.2012 № 4652-VI. Відомості Верховної Ради України. 2013. № 4. Ст. 25.

2. Про захист персональних даних : Закон України від 01.06.2010 № 2297-VI. Відомості Верховної Ради України. 2010. № 34. Ст. 481.

3. Системи тривожної сигналізації. Частина 1. Загальні вимоги. Розділ 1. Загальні положення (IEC 60839-1-1:1988, IDT) : ДСТУ IEC 60839-1-1:2003. [Чинний від 2004-05-01]. Київ : Держспоживстандарт України, 2004. 14 с.

4. Системи протипожежного захисту : ДБН В.2.5-56:2014. Чинний від 2015-01-01 Київ : Мінрегіон України, 2014. 162 с.

5. Про захист інформації в інформаційно-комунікаційних системах : Закон України від 05.07.1994 № 80/94-ВР. Відомості Верховної Ради України. 1994. № 31. Ст. 286.

6. Мартін Р. Чиста архітектура. Мистецтво розроблення програмного забезпечення : пер. з англ. Харків : Фабула, 2019. 368 с.

7. Фаулер М. Нові архітектурні підходи: NoSQL баз даних : пер. з англ. Київ : Вільямс, 2017. 192 с.

8. .NET Core documentation : official website. URL: https://learn.microsoft.com/en-us/dotnet/core (дата звернення: 08.06.2026).

9. Angular Documentation : official website. URL: https://angular.dev (дата звернення: 08.06.2026).

10. TypeScript Documentation : official website. URL: https://www.typescriptlang.org/docs (дата звернення: 08.06.2026).

11. Гамма Е., Хелм Р., Джонсон Р., Вліссідес Дж. Прийоми об'єктно-орієнтованого проектування. Паттерни проектування : пер. з англ. Київ : Діасофт, 2018. 366 с.

12. Світличний О. О., Плотницький С. В. Геоінформаційні системи і технології : підручник. Суми : Університетська книга, 2020. 432 с.

13. OpenLayers Docs : official website. URL: https://openlayers.org/doc (дата звернення: 08.06.2026).

14. Azure Cosmos DB Documentation : official website. URL: https://learn.microsoft.com/en-us/azure/cosmos-db (дата звернення: 08.06.2026).

15. Пасічник В. В., Резніченко В. А. Організація баз даних та знань : підручник. К. : БХВ, 2018. 448 с.

16. Філд Т. Проектування REST API : посібник : пер. з англ. Львів : Старого Лева, 2021. 210 с.

17. JSON Web Token (JWT) : RFC 7519 / Internet Engineering Task Force. 2015. URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 08.06.2026).

18. Трофименко О. Г., Прокоп Ю. В., Швайко І. Г. Веб-технології та веб-дизайн : підручник. Одеса : Фенікс, 2019. 324 с.

19. Системна та програмна інженерія. Вимоги до якості систем і програмного забезпечення та їхнє оцінювання (SQuaRE). Моделі якості систем та програмних продуктів (ISO/IEC 25010:2011, IDT) : ДСТУ ISO/IEC 25010:2016. [Чинний від 2017-09-01]. Київ : ДП «УкрНДНЦ», 2017. 34 с.

20. Баранцев О. В. Основи ручного тестування програмного забезпечення : навчальний посібник. Черкаси : ЧНУ, 2022. 185 с.

21. Ткачук В. М. Проектування інформаційних систем автоматизації сервісних служб. Комп'ютерні технології та системи. 2023. Вип. 12. С. 45-51.

22. Препарата Ф., Шеймос М. Обчислювальна геометрія: Вступ : пер. з англ. Київ : Наукова думка, 2016. 478 с.

23. HTTP Over TLS : RFC 2818 / Internet Engineering Task Force. 2000. URL: https://datatracker.ietf.org/doc/html/rfc2818 (дата звернення: 08.06.2026).

24. Буров Є. В. Комп'ютерні мережі : підручник. Львів : Бакті, 2019. 512 с.

25. Мельник А. О. Автоматизація обліку на підприємствах технічного обслуговування засобів захисту. Вісник ХНУ. Серія: Технічні науки. 2024. № 2. С. 112-118.