Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)
Навчально-науковий інститут комп'ютерних наук та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))
Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Магістерська кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему: «Розроблення вебплатформи для систематизації та ефективного управління корпоративними даними в приватних компаніях»

Виконав: студент 6 курсу групи КН-62м
спеціальності
122 "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)

Кіндрат В. О.
(прізвище та ініціали)

Керівник Думанський О. І.
(прізвище та ініціали)

Рецензент Карачинський В. А.
(прізвище та ініціали)

Львів – 2025

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра комп'ютерних наук
Рівень вищої освіти другий (магістерський)
Спеціальність 122 "Комп'ютерні науки"
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
_____ Борецька І.Б.
"10" грудня 2025 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Кіндрату Віталію Олеговичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення вебплатформи для систематизації та ефективного управління корпоративними даними в приватних компаніях

керівник роботи Думанський О. І., к.т.н, доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "29" квітня 2025 року № С-288
2. Термін подання студентом роботи 10 грудня 2025 р.
3. Вихідні дані до роботи Аналіз шляхів вирішення задачі, організаційна структура застосунку
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Розділ 1. Стан проблемної області
Розділ 2. Інформаційне забезпечення
Розділ 3. Математичне забезпечення
Розділ 4. Програмне забезпечення
Розділ 5. Розроблення стартап-проєкту
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Слайди для доповіді (підготовка матеріалу для доповіді загальним обсягом 10-12 слайдів)
6. Дата видачі завдання 01.05.2025

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Ознайомлення із предметною областю	01.05.2025-14.09.2025	Виконано
2	Опис функціоналу програмного продукту	15.09.2025-21.09.2025	Виконано
3	Прокетування та опис архітектури додатку	22.09.2025-03.10.2025	Виконано
4	Аналіз математичної моделі застосунку	04.10.2025-17.10.2025	Виконано
5	Розробка програмного продукту	18.10.2025-14.11.2025	Виконано
6	Тестування продукту	15.11.2025-23.11.2025	Виконано
7	Оформлення пояснювальної записки	24.11.2025-05.12.2025	Виконано

Студент _____ Кіндрат В. О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Думанський О. І.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Магістерська робота містить 81 сторінок пояснювальної записки, 23 рисунки, 5 UML діаграм, 6 рисунків прототипів інтерфейсу користувача 3 таблиці, 6 додатків, 18 джерел.

У роботі розроблено вебсистему, що поєднує динамічне формування структурованих даних із картографічним відображенням, забезпечуючи створення сутностей різного типу та їх прив’язку до геопросторових об’єктів у інтерактивному середовищі, реалізованому за допомогою технологій Phoenix/Elixir, React/TypeScript та PostgreSQL із підтримкою PostGIS. Запропонований вебзастосунок орієнтований на потреби приватних компаній, яким необхідні універсальні інструменти для організації, обліку та аналізу інформації в контексті просторових бізнес-процесів, і може бути адаптований до різних предметних областей, сприяючи підвищенню ефективності управління даними.

Ключові слова: вебсистема, цифровізація бізнес-процесів, управління даними, динамічні форми, кластеризація, Elixir, PostgreSQL.

ABSTRACT

The master's thesis consists of 81 pages of explanatory text, 23 figures, 5 UML diagrams, 6 user interface prototype images, 3 tables, 6 appendixes, and 18 references.

The study presents the development of a web-based system that integrates dynamic generation of structured data with cartographic visualization, enabling the creation of various types of entities and their linkage to geospatial objects within an interactive environment implemented using Phoenix/Elixir, React/TypeScript, and PostgreSQL with PostGIS support. The proposed application is designed to meet the needs of private companies requiring a universal tool for organizing, managing, and analyzing information within spatially oriented business processes, and can be adapted to diverse domains, thereby contributing to improved efficiency in data management.

Keywords: web system, business process digitalization, data management, dynamic forms, clustering, Elixir, PostgreSQL.

ТЕХНІЧНЕ ЗАВДАННЯ

Необхідно розробити програмне та алгоритмічне забезпечення вебсервісу для менеджменту та обліку даних приватних компаній, а саме:

1. Реалізувати модуль створення динамічних форм для збору даних з урахуванням специфіки продукту.
2. Розробити засоби роботи з геоданими, зокрема створення форм із полями для локацій та можливість прив’язки даних до конкретних географічних об’єктів.
3. Забезпечити відображення геоданих на інтерактивній карті світу, з підтримкою масштабування, пошуку та вибору об’єктів.
4. Розробити календарний модуль, що дозволяє створювати записи, пов’язані з певними локаціями та формами (події, завдання, нагадування).
5. Забезпечити механізми керування доступом користувачів, включно з аутентифікацією, авторизацією та розмежуванням прав відповідно до ролей у системі
6. Застосувати математичні алгоритми для обробки даних та оптимізації роботи сервісу.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

ERP – Enterprise Resource Planning – система планування ресурсів підприємства.

GPS – Global Positioning System – глобальна система супутникового позиціонування.

EPSG – European Petroleum Survey Group – система ідентифікаторів систем координат для геодезичних та картографічних даних.

KDE – Kernel Density Estimation – оцінка щільності розподілу за допомогою ядрових функцій.

DBSCAN – Density-Based Spatial Clustering of Applications with Noise – метод кластеризації на основі щільності з урахуванням шумів.

LOF – Local Outlier Factor – метод локального виявлення аномалій.

JSON – JavaScript Object Notation – текстовий формат обміну структурованими даними.

JSONB – JavaScript Object Notation Binary – бінарний формат зберігання JSON у базі даних PostgreSQL.

API – Application Programming Interface – інтерфейс взаємодії між програмними компонентами.

REST – Representational State Transfer – архітектурний стиль побудови вебслужб.

SPA – Single Page Application – односторінковий вебзастосунок, що працює без перезавантаження сторінки.

SaaS – Software as a Service – модель постачання програмного забезпечення як сервісу за підпискою.

ВСТУП

Швидкий розвиток інформаційних технологій призвів до збільшення обсягів даних та високих вимог до їх обробки, структурування та представлення. Все більше приватних фірм впроваджують вебсистеми для полегшення систематичної роботи з інформацією та підвищення ефективності своїх внутрішніх процесів. Хоча існує багато програмних систем, більшість з них або все ще є вузькоспеціалізованими, або недостатньо гнучкими, що ускладнює їх адаптацію до різних реальних сценаріїв. Саме це зумовлює актуальність дослідження, спрямованого на створення вебзастосунку, здатного поєднувати формування структурованих даних із можливістю їх просторової прив’язки.

Об’єктом дослідження у роботі є процеси управління та обліку даних у приватних компаніях, а предметом – методи і програмні засоби розроблення вебсистем для динамічного формування, зберігання та подальшої обробки даних різних типів у поєднанні з просторовими компонентами. Таке окреслення меж дослідження дає змогу чітко визначити наукову проблему, що полягає у відсутності універсальних інструментів, які забезпечували б інтеграцію атрибутних і геопросторових даних у межах єдиної програмної платформи.

Метою запропонованої роботи є розробка вебдодатку із гнучким формування шаблонів та об’єктів, що зв’язуються із фізичними сутностями на карті, дозволяючи користувачам використовувати вебсайт для ефективного управління даними. Для досягнення цієї мети супроводжуватиметься описанням вимог до програмного продукту, обґрунтуванням вибору технологій, проектуванням архітектури системи, реалізуванням компонент для сервера та клієнта, інтегрування алгоритмів обробки даних для оптимізації продуктивності сервісу та перевіркою отриманого рішення. Такий підхід дає змогу забезпечити комплексність і завершеність дослідження.

Наукова новизна полягає в розробці концепції вебдодатку, яка передбачає поєднання інструментів для динамічного формування організованих даних з модулем картографічного відображення – підхід, який не часто застосовується в існуючих вебсистемах, які зазвичай зосереджуються лише на одному з цих компонентів.

Інтеграція таких атрибутивних та геопросторових моделей формує новий спосіб організації інформаційних процесів у приватних компаніях.

Практична значущість результатів роботи визначається можливістю впровадження створеної системи в підприємства різних сфер діяльності. Вебдодаток для обліку, аналізу та візуалізації даних значно покращить управління бізнесом. Це підвищує ефективність бізнес-операцій та скорочує операційні видатки підприємства.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Роль ІТ у розвитку бізнесу

На сьогоднішній день уявити бізнес, який не використовує інформаційні технології для його ведення практично неможливо, при чому це актуально для бізнесу будь-якого рівня. Згідно з останніми дослідженнями величина інвестицій у цифрову трансформацію бізнесу зростає невпинно щороку і уже досягли показника 2.9 трильйонів доларів [1]. В свою чергу не інвестуючи у інформаційні технології, бізнес буде втрачати кошти, адже ми живемо у епоху інтернету та новітніх технологій, і саме інвестиції в такі технології допоможуть бізнесу завжди залишатись на плаву. Одним із головних рушіїв діджиталізації бізнесу у світі є власне малий та середній бізнес, так, за даними Європейської комісії, діджиталізація малого бізнесу може створити до 1.5 млн нових робочих місць у ЄС [2], а інвестиції у своє ПЗ в кінцевому результаті збільшує прибутки [3].

Попри очевидні переваги цифрової трансформації, повноцінна розробка та впровадження спеціалізованих програмних систем часто є фінансово недоступними для значної частини малих і середніх підприємств. Наукові дослідження засвідчують, що впровадження комплексних рішень, зокрема ERP-систем, є дорогим і ризикованим навіть для середніх компаній, тоді як для малих підприємств такі проєкти стають надмірно ресурсомісткими через обмежені бюджети та високу чутливість до витрат [4]. Аналітичні огляди ринку програмної розробки також підтверджують, що вартість створення індивідуальних вебзастосунків може варіюватися від кількох тисяч до сотень тисяч доларів залежно від їхньої складності та масштабу, що формує суттєвий фінансовий бар’єр для більшості бізнесів [5]. Одним із варіантів подолання цієї проблеми є створення універсальних програмних рішень, які здатні охоплювати різні предметні області та забезпечувати гнучке налаштування під потреби конкретного бізнесу.

1.2 Особливості систем обліку даних

Важливим аспектом облікових послуг у приватних корпораціях є робота з комерційно чутливою або конфіденційною інформацією. Несанкціонований доступ, модифікація або витік таких даних можуть призвести до реальних фінансових та репутаційних збитків, а також до порушення договірних зобов'язань. Це вимагає встановлення політик інформаційної безпеки повного спектру, таких як контроль доступу, шифрування, аудити та регулярний моніторинг активності користувачів.

Це передбачає необхідність гнучкості та масштабованості таких вебсистем. Ринкові умови, регуляторні зміни та технологічні розробки призводять до постійної трансформації бізнес-процесів. Таким чином, інформаційні системи також повинні дозволяти швидко розширювати функціональність, змінювати структури даних та адаптувати нову інформацію без значних витрат на модернізацію. Особливо важливо для компаній, що працюють у динамічних сферах, де попит на дані змінюється майже миттєво, є гнучкість архітектури.

Крім того, на систему також впливають точність, надійність та ефективність роботи. Компанія повинна мати надійні дані, своєчасно забезпечувати прийняття правильних управлінських рішень, а також планування та прогнозування діяльності компанії. Відповідно, система повинна вирішувати певні питання: правильність обробки, відсутність дублювання інформації в даних, цілісність даних у багатокористувацькому середовищі та запобігання помилкам, викликаним людським фактором.

Іншим ключовим питанням є дотримання регуляторних вимог та стандартів обробки інформації. Існують суворі регламенти щодо зберігання та передачі даних у різних галузях промисловості, включаючи фінансові ринки, страхові компанії або сфери особистих даних. Ці регламенти повинні враховуватися системою обліку даних, щоб забезпечити юридичну правильність операцій компанії.

У сукупності ці чинники формують комплекс вимог, яким мають відповідати сучасні вебсистеми обліку даних, що використовуються у приватних компаніях.

1.3 Порівняльний аналіз вебсистем для обліку корпоративних даних

На ринку сьогодні представлено не так багато систем, які б містили в собі динамічну генерацію форм та масштабовану карту світу, на якій ми можемо ці форми відображати у вигляді точок, ліній чи полігонів. Водночас, існують цікаві самостійні рішення, що забезпечують окрему генерацію форм та незалежний засіб для менеджменту мапами.

1.3.1 Wu Foo

WuFoo – це онлайн-платформа для створення форм для вебдодатків, яка дозволяє користувачам легко створювати власні форми, без витрат часу, а також використовувати шаблони, надані вбудованими функціями. Основні переваги системи – це простота використання та додаткові функції, такі як інструменти для створення звітів та опитувань. Недоліком цієї платформи є те, що вона має дуже старий інтерфейс, що робить систему менш сучасною, а це у свою чергу створює поганий досвід для сучасних користувачів.

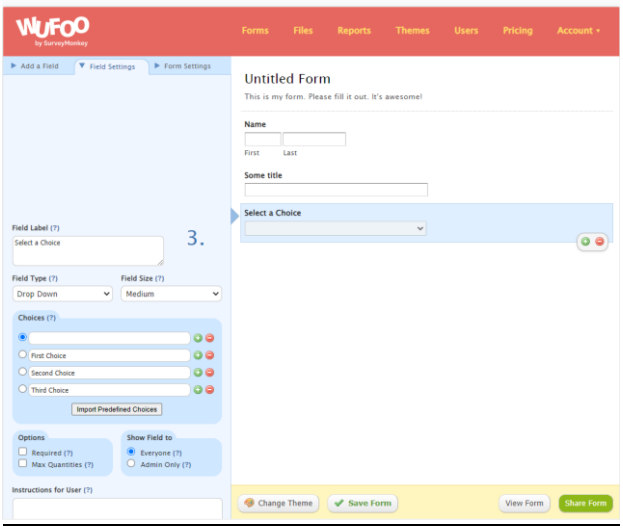

Рисунок 1.1 – Інтерфейс конструктора форм у WuFoo

1.3.2 Cognito Forms

Cognito Forms – це вебсервіс для створення форм та структурованих опитувань. Найбільша перевага системи полягає в тому, що вона дозволяє зручно та просто створювати форми, а також має інтеграції з платіжними сервісами, корпоративними

платформами, календарними системами та іншими зовнішніми сервісами. Водночас до недоліків платформи можна віднести обмеженість сфери її застосування, оскільки Cognito Forms передусім функціонує як конструктор форм і не забезпечує ширшої функціональності, необхідної для комплексного управління даними.

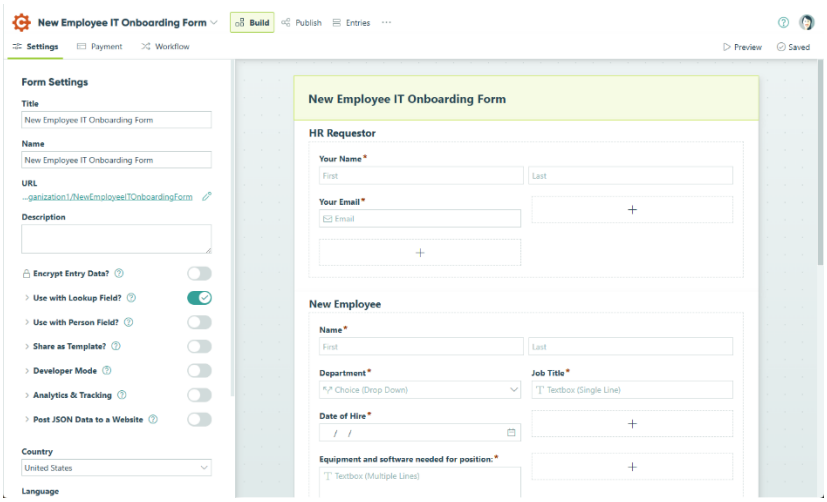

Рисунок 1.2 – Інтерфейс конструктора форм у Cognito Forms

1.3.3 ArcGis

ArcGIS – це вебдодаток, розроблений для створення, перегляду та аналізу картографічних даних, що керується компанією Esri. Ця служба в основному призначена для користувачів, які зацікавлені в доступі до географічної інформації в режимі реального часу. Її просторові дані надають можливість для різних операцій, включаючи візуалізацію, редагування, накладання шарів, геокодування та геостатистичний аналіз. Таким чином, ArcGIS широко застосовується в державному управлінні, екологічному моніторингу, логістиці, міському плануванні та багатьох інших сферах, де геопросторові дані є важливими.

До основних переваг ArcGIS належить багатий набір інструментів для роботи з картами та географічною інформацією, наявність вбудованих аналітичних модулів, а також підтримка інтеграції з різними зовнішніми геоданими та сервісами. Крім того, система має розвинену інфраструктуру для командної роботи, що дозволяє декільком користувачам одночасно взаємодіяти з просторовими даними, створювати спільні проєкти та публікувати карти у вебсередовищі.

Проте, у нього також є ряд недоліків, таких як високі витрати на придбання та ліцензування, що заважає йому бути більш доступним для малих і середніх підприємств. Ще однією проблемою є його складність: велика кількість доступних функціональних можливостей означає, що без відповідних навчальних програм система досить важка для освоєння недосвідченим користувачем. Це вимагає спеціалізованого навчання або участі, що збільшує витрати на впровадження.

Незважаючи на усі ці обмеження, ArcGIS продовжує бути одним з найпотужніших інструментів для роботи з геопросторовими даними. Однак для багатьох підприємств система стає доцільною лише за умови достатнього балансу між тим, скільки функціоналу сервісу може знадобитися для роботи, і скільки ресурсів компанія планує витратити на впровадження та підтримку системи.

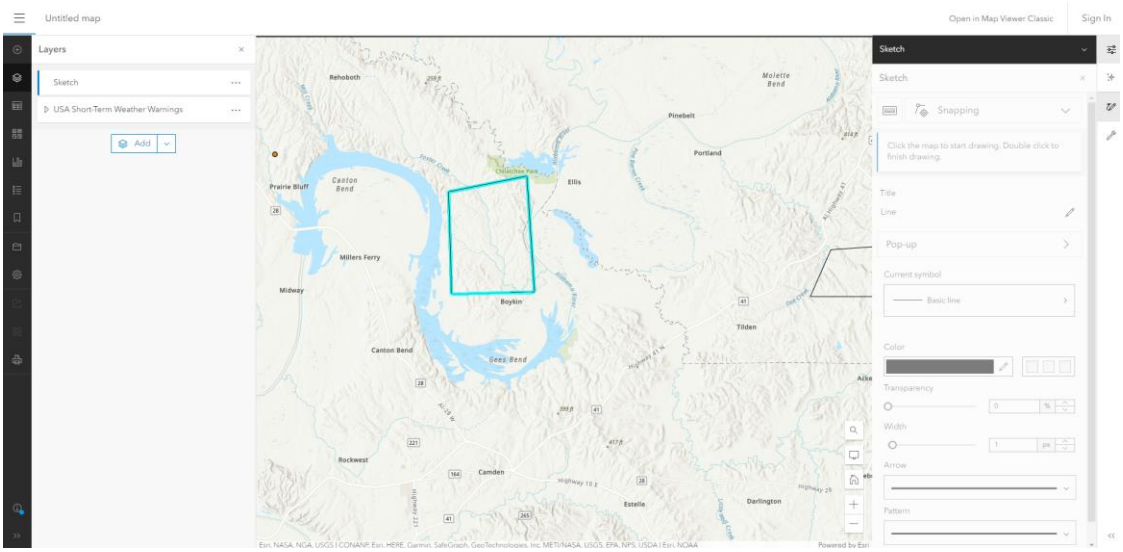

Рисунок 1.3 – Інтерфейс картографічного модуля ArcGis

Висновки до розділу

У межах оглядового розділу було проаналізовано загальні передумови цифрової трансформації бізнесу та особливості використання вебсистем для обліку даних. У ході дослідження ідентифіковано низку наявних вебрішень, призначених для динамічного створення форм та роботи з картографічною інформацією. Проте жодна з розглянутих систем не забезпечує повноцінної інтеграції цих двох функціональних складових. Таким чином, аналогів або прямих конкурентів розроблюваної системи виявлено не було.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Постановка завдання

У межах створення вебсистеми для менеджменту та обліку даних приватних компаній необхідно сформувати цілісну інформаційну модель, що забезпечуватиме правильну організацію даних, коректність взаємодії між користувачем та системою, а також узгодженість усіх бізнес-процесів. Це у свою чергу передбачає виділення об’єктів дослідження, визначення набору вхідних та вихідних даних, а також формалізацію їх взаємозв’язків з метою подальшої побудови концептуальної моделі.

У системі визначено перелік базових сутностей, які описують структуру предметної області та забезпечують реалізацію основних функціональних можливостей. До їх складу належать як бізнес-орієнтовані об’єкти (локації, форми, події), так і службові елементи (користувачі, організація, типи даних), що забезпечують логічну цілісність усієї платформи. У процесі дослідження виділено такі ключові об’єкти:

Організація (Organization) – центральний об’єкт, у межах якого існують усі інші сутності та формується доступ користувачів. Саме на рівні організації визначаються межі даних, внутрішні політики та структура бізнес-процесів.

Користувач (User) – суб’єкт взаємодії з системою, що має певні права доступу та виконує операції із даними відповідно до виділеної ролі.

Локація (Location) – геопросторовий об’єкт, який є прототипом фізичних об’єктів підприємства та виступає контейнером для форм, контактів і записів календаря.

Тип форми (FormType) – опис структури даних, що визначає набір полів, їх типи та обмеження для майбутніх екземплярів форм.

Екземпляр форми (FormInstance) – конкретний набір інформації, заповнений користувачем відповідно до певного типу форми та прив’язаний до локації.

Контакт (Contact) – інформація про працівника чи відповідальну особу, яка може бути закріплена за локацією або певним бізнес-процесом.

Подія календаря (Event) – подія, що обмежена часовими рамками та може бути пов’язана з локаціями або формами.

Кожен із цих об’єктів виконує певну роль у загальній архітектурі системи, формуючи взаємопов’язану мережу сутностей. Їхня структура і зв’язки відображаються у UML-діаграмах класів та ER-діаграмах, що дозволяє візуально окреслити логіку функціонування системи та визначити найважливіші взаємодії. У додатку А на діаграмі прецедентів наведено усі можливі варіанти використання системи.

Вхідні дані системи охоплюють усю інформацію, що надходить від користувача або генерується через інтерфейси взаємодії, зокрема формується в процесі заповнення динамічних форм, створення локацій чи подій, а також під час роботи з інтерактивною картою. До основних типів вхідних даних належать:

- текстові значення – назви локацій, описи об'єктів, імена співробітників, назви полів форм;
- числові значення – ідентифікатори, параметри полігонів, значення числових полів форм;
- датово-часові значення – дані для подій календаря, час створення сутностей;
- булеві параметри – статуси активності, параметри видимості тощо;
- геопросторові дані – координати точок або полігонів, сформовані за допомогою інтерактивної карти;
- структуровані дані динамічних форм – набір значень, що відповідають визначеній структурі типу форми.

Для забезпечення коректності та узгодженості даних розроблено систему обмежень, що включає:

- обов’язковість заповнення визначених полів, необхідних для створення сутності;
- унікальність певних атрибутів у межах організації (електронна адреса, логін, назва сутності);

- валідацію формату даних, включно з перевіркою типів значень, довжини тексту, діапазонів числових даних;
- цілісність посилань, яка гарантує, що залежні сутності не можуть існувати без базових (наприклад, форма не створюється без прив’язки до типу форми).

Таким чином, система вхідних даних забезпечує основу для формування усіх бізнес-об’єктів і гарантує, що інформація буде введена коректно та відповідно до логіки предметної області.

Вихідні дані системи являють собою результати обробки вхідної інформації та відображають її у форматах, зручних для подальшої роботи користувачів і зовнішніх систем. Вони доступні як у формі графічного представлення, так і у вигляді структурованих даних, придатних до інтеграції або експорту. До вихідних даних належать:

- інтерактивне картографічне відображення локацій та пов’язаних з ними об’єктів;
- табличні представлення – списки локацій, форм, контактів і подій;
- деталізовані сторінки сутностей, що містять повну інформацію про вибраний об’єкт та його залежності;
- експортовані файли у форматі CSV, що дозволяють аналізувати дані поза межами системи;
- структуровані JSON-відповіді, які повертає API для клієнтської частини або зовнішніх сервісів.

Вихідні дані дозволяють відобразити актуальний стан бізнес-об’єктів у системі, підтримують інформаційні потоки між клієнтом і сервером та забезпечують основні інструменти для прийняття управлінських рішень.

2.2 Проєктування архітектури програмного забезпечення

Вебсистема, призначена для управління інформаційними ресурсами приватних компаній, функціонує на засадах клієнт-серверної архітектури. Система складається з двох основних модулів: клієнтського та серверного. Клієнтський модуль (Front-end), з яким безпосередньо взаємодіє користувач, розгорнутий як вебсервіс у стандартному веббраузері. Ця частина відповідає за генерацію запитів із використанням протоколів HTTPS та їх подальшу передачу до серверної частини. На противагу цьому, серверний модуль (Back-end), що являє собою програмний додаток, розміщений на сервері та виконує ключову функцію прийому, валідації та обробки запитів, ініційованих клієнтами. Окрім того, сервер забезпечує встановлення з’єднання та управління транзакціями з базою даних для маніпуляцій з корпоративними даними. Механізм взаємодії між цими компонентами реалізується за принципом циклу "запит-відповідь". Після ініціації користувачем відповідної операції на вебсайті, клієнтський модуль формує HTTP-запит і здійснює його передачу на сервер, переходячи у режим очікування відповіді. Сервер, у свою чергу, приймає та обробляє запит, за необхідності звертаючись до сховища даних для виконання операцій вибірки чи модифікації. Після завершення обробки сервер генерує відповідь і повертає її клієнту для подальшої інтерпретації та відображення результатів відповідно до логіки системи.

2.2.1 Архітектура серверної частини вебзастосунку

Реалізація серверної частини системи передбачає застосування архітектурного патерну Model-View-Controller (MVC) [6]. Крім того, побудова програмного інтерфейсу (API) застосунку здійснюватиметься з повним дотриманням принципів REST-підходу (Representational State Transfer). Відповідно до обраної архітектури, серверний компонент структуровано на три функціональні рівні, що забезпечують чітке розділення відповідальності: рівень контролерів, рівень бізнес-логіки та рівень доступу до даних.

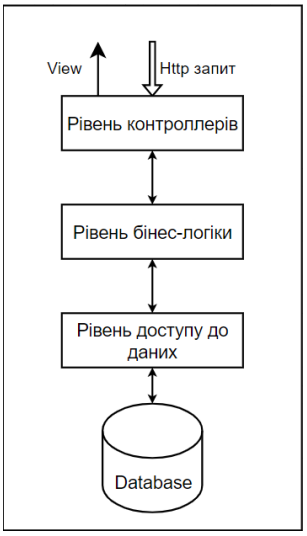

Рисунок 2.1 – Схема взаємодії рівнів серверної частини

Рівень контролерів (Presentation Layer) є першим рівнем, відповідальним за прийом, первинну валідацію та маршрутизацію запитів, що надходять від клієнтів. Основне завдання контролера полягає у визначенні необхідного сервісу бізнес-логіки для виконання операції, а також у формуванні фінальної відповіді. Як відповідь клієнту повертаються дані Моделі та посилання на Шаблон Представлення (View), що відповідає за коректну візуалізацію цих даних.

Бізнес-логіка системи інкапсульована у сервісному рівні. Цей рівень виконує основні функції обробки, валідації та трансформації даних відповідно до правил, встановлених системою. Сервіси отримують дані від контролерів і, керуючись логікою, забезпечують транзакційну цілісність операцій, взаємодіючи з нижнім рівнем для забезпечення стійкості даних.

Рівень доступу до даних (Data Access Layer, DAL) представлений набором репозиторіїв або об'єктів доступу до даних (DAO). Цей шар інтерфейсує сервісний рівень з фізичним сховищем даних і відповідає за всі типи операцій з базою даних, включаючи вибірку, запис, модифікацію та видалення інформації. Застосування даного архітектурного підходу забезпечує високий ступінь ізоляції рівнів, що сприяє підвищенню гнучкості, масштабованості та спрощенню обслуговування системи. Схема компонентів серверної частини, що відображає їх ієрархію та зв'язки, представлена у Додатку Б.

2.2.2 Архітектура клієнтської частини вебплатформи

Реалізація клієнтської частини вебплатформи ґрунтується на використанні бібліотеки React.js у поєднанні з мовою програмування TypeScript. Застосування архітектурного підходу Single Page Application (SPA) забезпечує формування динамічного вебінтерфейсу, де вся взаємодія користувача відбувається в межах єдиного HTML-документу. Така імплементація дозволяє здійснювати зміну вмісту сторінки без її повного перезавантаження, що підвищує швидкість роботи застосунку та значно покращує користувацький досвід (User Experience, UX). Архітектура клієнтського рівня умовно диференціюється на два основні модулі, попри їхню інтеграцію в єдину структуру проєкту: модуль відображення (View Module) та модуль сервісів (Services Module). Такий поділ забезпечує динамічну модифікацію DOM-дерева відповідно до результатів виконання програмної логіки на JavaScript.

Модуль відображення відповідає за візуалізацію даних, отриманих від серверної частини, та формування користувацького інтерфейсу із застосуванням стандартів HTML, CSS і компонентного підходу React. Його функціональні завдання включають рендеринг компонентів, управління їхнім внутрішнім станом (state), обробку ініційованих користувачами подій та забезпечення реактивності інтерфейсу.

Модуль сервісів забезпечує взаємодію клієнтського рівня з сервером. До його функцій належать формування HTTP-запитів, обробка відповідей, керування глобальним станом отриманих даних, обробка помилок та підготовка інформації для подальшого використання у модулі відображення. Компоненти цього модуля виступають посередниками між бізнес-логікою інтерфейсу та API серверної частини, надаючи доступ до зовнішніх даних.

2.3 Проектування бази даних

Для управління та зберігання даних системи обрано реляційну систему керування базами даних (СКБД) PostgreSQL. Розробка бази даних та взаємодія з нею здійснюватиметься за допомогою бібліотеки Ecto [7]. Цей інструмент, інтегрований у мову програмування Elixir, використовує підхід, концептуально аналогічний до парадигми "Code First". В Ecto моделі даних реалізуються через спеціалізовані модулі

Elixir, які слугують для репрезентації таблиць бази даних та описують відношення між ними. У цих модулях відбувається декларативне визначення структур даних і схем (Schema), які точно відображають поля відповідних таблиць у сховищі даних. Таким чином, логіка бази даних формується безпосередньо у коді застосунку. В додатку В наведено ERD-діаграму.

База даних матиме наступні сутності:

- evnts – таблиця, що містить інформація про події
  - title – назва події
  - start – дата початку події
  - end – дата кінця події
  - organization_id – ідентифікатор організації, якій належить подія
  - userId – ідентифікатор користувача, який організує подію
- form_types – таблиця, яка містить тип форми
  - name – назва типу
  - fields – JSON сукупність параметрів, що зберігає поля типу
  - organization_id – ідентифікатор організації, якому належить тип
- forms – таблиця, що містить екземпляри форм
  - name – назва форми
  - created_at – дата створення екземпляру
  - organization_id – посилання на організацію, якій належить форма
  - form_type_id – посилання на тип форми, на основі якої створена дана форма
- locations – таблиця, яка містить інформацію про фізичні локації
  - name – назва локації
  - location – JSONB поле, де зберігається дані про локацію
  - address – адреса в текстовому вигляді
  - type – тип локації (точка, лінія чи полігон)
  - organization_id – посилання на організацію, якій належить локація
- location_forms – проміжна таблиця для зв’язку локацій та форм
  - form_id – ідентифікатор на таблицю форм

    - location_id – ідентифікатор на таблицю локації
- location_workers – проміжна таблиця для поєднання локацій та працівників
    - location_id – ідентифікатор на таблицю локації
    - worker_id – ідентифікатор на таблицю працівників
- organizations – проміжна таблиця для поєднання локацій та працівників
    - name – назва організації
- permissions – таблиця, що зберігає дані про доступа
    - name – назва доступа
    - role_id – ідентифікатор ролі, якій належить доступ
    - organization_id – ідентифікатор організації, якій належить доступ
- roles – таблиця для зберігання ролей
    - name – назва ролі
    - organization_id – ідентифікатор організації, якій належить роль
- users – таблиця користувачів, яка зберігає інформацію про користувача
    - full_name – ім’я та прізвище
    - gender – гендер користувача
    - account_id – ідентифікатор акаунту користувача, де зберігаються аутентифікаційні дані користувача
    - organization_id – ідентифікатор організації, якій належить користувач
- user_roles – проміжна табличка для поєднання таблиці користувачів та ролей
    - role_id – ідентифікатор ролі
    - user_id – ідентифікатор користувача
- workers – представлення сутності прцівника у базі даних
    - name – назва працівника
    - email – електронна адреса
    - age – вік
    - title – позиція працівника
    - organization_id – посилання на організацію, якій належить працівник

2.4 Прототипування інтерфейсу користувача

Для створення прототипу графічного інтерфейсу було використано сервіс Figma. Створені прототипи являють собою загальну каркасну (скелетонну) структуру візуальної частини проєкту. Рівень деталізації прототипування визначено як високий, що вимагає максимального наближення зовнішнього вигляду фінального застосунку до прототипів. Такий підхід є необхідною умовою для забезпечення функціональної практичності та високої ергономічності інтерфейсу користувача. У додатку Д наведено усі прототипи, що були створені у процесі виконання цього етапу роботи.

Головний компонент навігації – бічна панель, сегментована на три функціональні розділи: секція стандартних форм, секція користувацьких (кастомних) форм та секція налаштувань (рис. 2.2). До стандартних форм віднесено такі функціональні модулі, як мапи, календар, локації та контакти. Усі інтерактивні опції бічного меню повинні мати ефект наведення курсору (hover effect), що забезпечує користувачеві необхідний візуальний зворотний зв'язок і підкреслює їхню інтерактивність, подібно до реалізації для пункту "Локації".

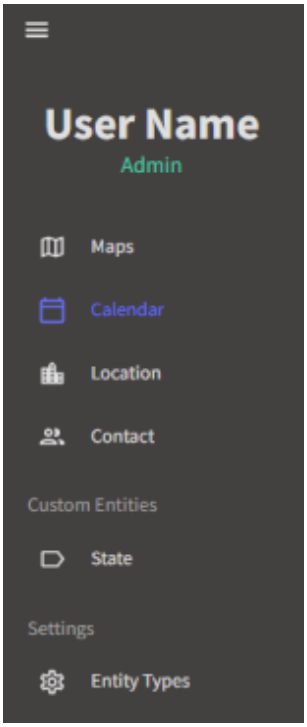

Рисунок 2.2 – Прототип бічної палені

При проектуванні візуального оформлення елементів інтерфейсу особливу увагу приділено колірній індикації функціональних кнопок. Зокрема, кнопки,

активація яких може призвести до незворотних операцій (наприклад, видалення), мають бути виконані у червоній колірній гамі з метою привернення уваги користувача та запобігання випадковій дії. Додатково, на кожній сторінці передбачено обов'язкове розміщення кнопок для повернення на попередній рівень навігації.

Модальне вікно, призначене для створення типу форми, повинно реалізовувати динамічну зміну структури (рис. 2.3). Це передбачає додавання додаткового поля відповідно до типу поля, який було обрано користувачем. Кожне поле форми має супроводжуватися контекстними підказками (tooltips), що забезпечує ергономічність інтерфейсу та сприяє швидкій орієнтації користувача щодо необхідного формату введення даних.

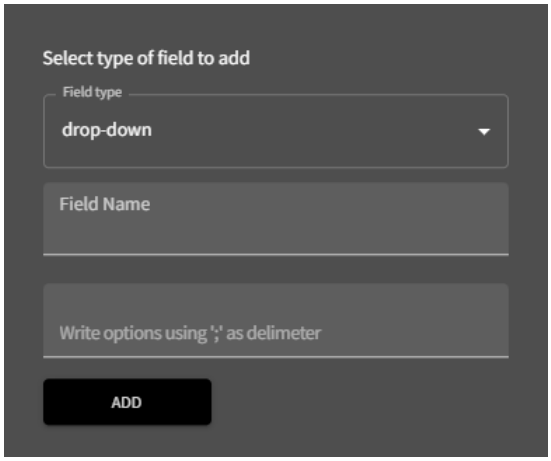

Рисунок 2.3 – Прототип модального вікна для типу форми

При здійсненні процесу заповнення форми всі некоректно введені дані підлягають автоматичній валідації з подальшою індикацією помилки шляхом виділення відповідного поля червоним кольором та відображенням контекстного текстового повідомлення.

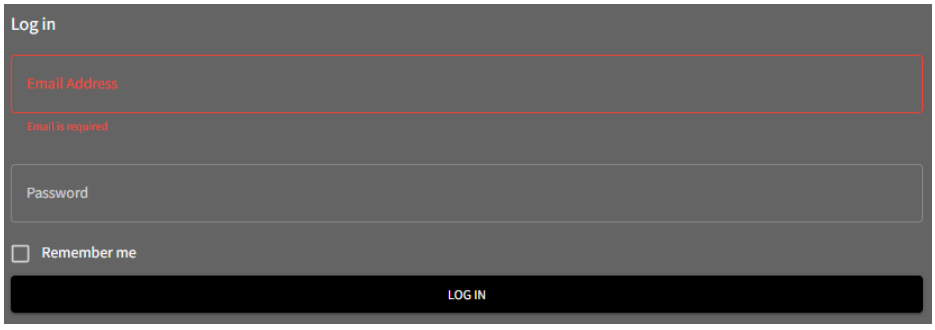

Рисунок 2.3 – Сторінка входу в систему

Висновки до розділу

У даному розділі було визначено та обґрунтовано структуру розроблюваної платформи, а також проаналізовано застосовані стратегії реалізації. Вебсистема матиме двокомпонентну архітектуру, що передбачає взаємодію клієнтської та серверної частин, яка здійснюватиметься за допомогою протоколів HTTP/HTTPS із використанням формату обміну даними JSON. Серверна частина базується на трирівневій архітектурі, відповідаючи при цьому стандартам проектування REST Web API. Клієнтський додаток для користувачів буде реалізовано у форматі односторінкового вебзастосунку (SPA).

Графічний інтерфейс користувача було спроектовано за допомогою вебінструменту Figma, при цьому основні елементи інтерфейсу були детально описані з дотриманням 10 принципів ергономіки Якоба Нільсена. В якості системи управління базами даних (СКБД) обрано PostgreSQL. Для розробки структури даних було прийнято рішення застосовувати підхід Code First, який передбачає створення міграцій до бази даних на основі запрограмованих моделей. У результаті була сформована логічна модель бази даних із детальним описом призначення кожної таблиці. Додатково було побудовано діаграму розгортання та діаграму послідовності, яка візуалізує взаємодію користувачів з компонентами та вузлами системи.

РОЗДІЛ 3. МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Кластеризація

У сучасних інформаційних системах приватних компаній формується значна кількість даних, пов’язаних із різними просторовими об’єктами: філіями, точками обслуговування, складами, технологічними зонами, клієнтами або шляхами пересування ресурсів. Зі зростанням обсягів таких даних постає потреба не лише у їх фіксації та зберіганні, але й у виявленні прихованих закономірностей, просторових патернів і структурних залежностей, які можуть бути безпосередньо пов’язані з ефективністю процесів управління.

Одним із ключових математичних підходів до аналізу великих масивів просторової інформації є кластеризація – метод поділу множини об’єктів на групи (кластери) таким чином, щоб елементи, які належать одному кластеру, були подібними між собою за певною мірою відстані або густини, тоді як об’єкти з різних кластерів – суттєво відрізнялися.

Кластеризація є важливим інструментом для компаній, діяльність яких передбачає роботу з територіально розподіленими об’єктами. На відміну від традиційних алгоритмів сегментації, що працюють з табличними даними, просторові методи кластеризації враховують географічну близькість та щільність розташування точок, що робить їх особливо корисними для оперативного та стратегічного управління бізнес-процесами.

Основні завдання кластеризації в управлінні просторовими даними

1. **Виявлення регіонів підвищеної бізнес-активності**. Аналізуючи великі набори локацій (клієнтських звернень, операційних точок, складських переміщень), система може автоматично визначати географічні зони з високою інтенсивністю подій. Такі зони часто відповідають перспективним районам для розвитку компанії чи збільшення кількості ресурсів.

2. **Аналіз розподілу клієнтів та заявок**. Кластеризація дозволяє зрозуміти, де зосереджені основні групи клієнтів або де найчастіше виникають певні типи звернень.

Це допомагає оптимізувати маршрути обслуговування, кількість персоналу, графік роботи та територіальну інфраструктуру.

3. **Ідентифікація потенційних зон розвитку**. Системи аналізу просторових даних можуть визначати «порожні» або недостатньо покриті області, у яких немає активних точок компанії, але є попит чи активність клієнтів. Це відкриває можливості для розширення мережі відділень, складів, торгових точок або сервісних центрів.

4. **Виявлення аномалій та відокремлених локальних випадків**. У бізнес-середовищі часто виникають поодинокі або випадкові точки, які не утворюють стійких закономірностей. Кластеризація дозволяє ідентифікувати їх як потенційні аномалії – наприклад, помилки введення даних, одиничні нестандартні географічні події чи непередбачувані звернення, що потребують окремого аналізу.

3.2 Алгоритм DBSCAN: математична модель

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) належить до алгоритмів щільнісної кластеризації. Його ключова ідея полягає у виділенні кластерів як областей підвищеної щільності точок у просторі [8]. Для глибшого розуміння доцільності вибору алгоритму проведемо порівняння з класичним методом k-means, що широко використовується у бізнес-аналітиці. Відмінності наведено у таб. 3.1.

Таблиця 3.1 – Порівняння DBSCAN та k-means

Критерій	DBSCAN	k-means
Тип кластерів	Будь-яка форма	Сферичні, приблизно однакового розміру
Параметри	ε, *minPts*	Кількість кластерів
Виявлення шуму	Так – класифікує окремі точки як шум	Ні – усі точки належать до кластерів

Продовження таблиці 3.1

Чутливість до масштабу	Стійкий до різних розмірів кластерів при правильному виборі похибки	Погано працює при різній густині даних
Складність	В середньому O(n logn)	В середньому O(nk) за ітерацію
Сценарій застосування	Геодані, нерівномірні розподіли, виявлення «гарячих зон»	Сегментація клієнтів за числовими атрибутами

Таким чином, у задачі просторового аналізу бізнес-локацій DBSCAN має суттєві переваги: він дозволяє формувати кластери довільної форми, виявляти шуми та природно підходить для географічних даних. k-means натомість краще застосовувати для задач, де дані рівномірно розподілені, а кількість кластерів відома наперед (наприклад, сегментація клієнтів за розміром бізнесу).

Алгоритм DBSCAN оперує двома параметрами:

- ε (eps) – максимальний радіус околу точки (epsilon-neighborhood);
- *minPts* – мінімальна кількість сусідів, необхідна для того, щоб точка вважалася core-точкою.

Формалізація:

Нехай множина даних - це $D = \{p1, p2, ..., pn\}$ де кожна p_i має координати у просторі R^m. Відстань між двома точками визначається метрикою $d(p_i, p_j)$. У нашому випадку варто враховувати масштаб карти, на якому виконується розрахунок. Якщо масштаб становить 1 см : 50 км і розрахунок здійснюється на рівні великих регіонів або навіть країн, для географічних координат доцільно використовувати відстань Гаверсина, оскільки вона враховує сферичність Землі:

$$d(p_i, p_j) = 2R\, arcsin \sqrt{\sin(\frac{\Delta\varphi}{2})^2 + \cos\varphi_i \cos\varphi_j \sin(\frac{\Delta\gamma}{2})^2} \qquad (3.1)$$

де R - радіус Землі, φ і γ - широта й довгота відповідних точок.

У випадку менших відстаней більш оптимальним варіантом є використання евклідової відстані, що дозволить скоротити час обрахунку алгоритму за рахунок меншої кількості звертань до тригонометричних функцій без особливих втрат у точності:

$$d(p_i, p_j) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (3.2)$$

де x_i та y_j – це плоскі координати точки, які можна отримати перевівши географічні координати за допомогою формули Web Mercatora (EPSG:3857).

$$x = R\gamma \quad (3.3)$$

$$y = R \ln(\tan\frac{\pi}{4} + \frac{\varphi}{2}) \quad (3.4)$$

де R – радіус земного еліпсоїду величиною у 6378137 м.

Правила формування кластерів:

1. Точка p є core-точкою, якщо $|N_\varepsilon(p)| \geq min\,Pts$, де $N_\varepsilon(p) = \{q \in D \mid d(p,q) \leq \varepsilon\}$
2. Точка є прикордонною, якщо вона не core, але належить до околу core-точки.
3. Точка вважається шумом, якщо вона не належить жодному кластеру.

Таким чином, DBSCAN формує кластери як зв’язні компоненти множини core-точок та їхніх прикордонних сусідів.

Алгоритм DBSCAN має різні асимптотичні складності залежно від способу пошуку сусідів:

- При використанні структур індексації простоору (KD-tree, R-tree) – O($n \log n$)
- При наївному пошуку сусідів – O(n^2)

3.3 Алгоритм KDE: математична модель

Оцінка густини розподілу (Kernel Density Estimation, KDE) є одним із базових непараметричних методів статистичного аналізу, що дає змогу побудувати неперервну оцінку щільності розподілу випадкової величини на основі дискретної множини спостережень [9]. На відміну від параметричних моделей, KDE не

передбачає припущень щодо форми розподілу даних, що робить метод універсальним і застосовним у широкому спектрі задач аналізу просторової інформації.

Сутність алгоритму полягає в тому, що для кожної точки вибірки формується локальна функція-ядро, центрована в цій точці. Надалі всі ці ядра агрегуються, утворюючи гладку оцінку щільності розподілу. Формально KDE для множини даних $X = \{x_1, x_2, x_3, .., x_n\}$ визначається як:

$$f_h(x) = \frac{1}{nh} \sum_{i=1}^{n} K(\frac{x - x_i}{h}) \quad (3.5)$$

де K – функція ядра (найчастіше гаусова), а h – ширина вікна (bandwidth), що визначає ступінь згладжування.

Вибір ядра визначає форму локальної оцінки щільності, однак практичні дослідження демонструють, що саме параметр h має вирішальний вплив на кінцевий результат. Занадто малий bandwidth призводить до «перенавчання» та появи надлишкових локальних максимумів, тоді як занадто великий – до надмірного згладжування, унаслідок чого зникають локальні особливості даних.

У задачах просторового аналізу KDE використовується переважно для побудови карти теплових зон (heatmap), яка відображає інтенсивність подій або об’єктів на певній території. Для географічних координат у двовимірному просторі формула набуває вигляду:

$$f_h(x) = \frac{1}{nh^2} \sum_{i=1}^{n} K(\frac{\|s - s_i\|}{h}) \quad (3.6)$$

де s – координата на карті, а відстань між точками розраховується за принципом опписаним у підпункті 3.2.

KDE має низку важливих властивостей, які роблять його придатним для інтеграції в системи обліку геоприв’язаних даних. По-перше, метод дозволяє візуалізувати області з високою концентрацією подій, не вимагаючи попередньої кластеризації чи знання параметрів розподілу даних. По-друге, KDE може застосовуватися у режимі реального часу, що є важливим для систем моніторингу обладнання, складів або об’єктів інфраструктури.

Алгоритм також має певні обмеження. Головною проблемою є чутливість до вибору bandwidth, оскільки невдале налаштування може суттєво спотворити результат. У випадку великих вибірок зростає обчислювальна складність, що може вимагати оптимізації, наприклад, використання KD-дерев або grid-based апроксимацій. Крім того, метод не дає чітких меж кластерів, натомість формує неперервну щільнісну поверхню, що може бути недоліком для задач, де потрібне конкретне групування.

3.4 Алгоритм LOF: математична модель

Алгоритм LOF (Local Outlier Factor) належить до класу методів виявлення аномалій, що базуються на оцінці локальної щільності розташування об’єктів у просторі. На відміну від глобальних статистичних підходів, які визначають аномалії як точки, що суттєво відхиляються від загального розподілу даних, LOF аналізує поведінку кожного об’єкта у контексті його найближчого оточення [10]. Такий підхід особливо важливий у ситуаціях, коли дані мають нерівномірну густину або структуру, що складається з кількох локальних груп, адже глобальні критерії в таких умовах часто виявляються неефективними.

Основна ідея LOF полягає у порівнянні локальної щільності точки зі щільністю її найближчих сусідів. Якщо точка суттєво «розріджена» відносно свого оточення, вона вважається аномальною. І навпаки, якщо щільність її локального оточення подібна до щільності сусідніх точок, така точка вважається типовою для свого середовища. LOF належить до непараметричних методів, тобто не робить припущень щодо форми розподілу даних. Це робить алгоритм придатним для аналізу просторових, економічних, комерційних та інших типів даних, де закономірності поводження об’єктів зазвичай не описуються простими математичними моделями.

Формалізація:

Нехай задано множину точок $X = \{x_1, x_2, x_3, .., x_n\}$ та метрику відстані d(x, y), яка визначає відстань між двома об’єктами. Алгоритм та принцип обчисленно відстані між точками пояснено в пункті 3.2. Сам алгоритм складається із декількох кроків:

1. **Визначення k-distance**. Відстань k-найближчого сусіда точки p: $k-distance(p) = \mathrm{d}(\mathrm{p}, q_k)$, де q_k – найближчий сусід
2. **Фіксація множини сусідів**. Множина сусідів визначається як:

$$N_k(p) = \{q \in X \mid d(p,q) \leq k - distance(p)\} \quad (3.7)$$

3. **Reachability distance**. Поняття досяжної відстані (reachability distance) забезпечує стабільність алгоритму в умовах кластерів високої густини:

$$reach\_dist_k(p,q) = max(k - distance(q, d(p,q)) \quad (3.8)$$

4. **Local Reachability Density (LRD)**. Локальна густина точки визначається як обернена середня досяжна відстань.

$$LRD_k(p) = (\frac{1}{|N_k(p)|} \sum_{q \in N_k(p)} reach\_dist_k(p,q))^{-1} \quad (3.9)$$

5. **Обчислення коефіцієнта LOF**. Кінцевий показник LOF визначається як середнє відношення густини сусідів до густини точки:

$$LOF_k(p) = \frac{1}{|N_k(p)|} \sum_{q \in N_k(p)} \frac{LRD_k(q)}{LRD_k(p)} \quad (3.10)$$

Результат LOF має чітку числову інтерпретацію:

- LOF ≈ 1.0 - точка належить до нормального локального розподілу.
- LOF > 1.5 - слабка аномалія або підозріле відхилення.
- LOF > 2.0 - значна аномальність.
- LOF > 3.0 - сильна аномалія, майже гарантовано неправильна або нетипова точка.

Таким чином алгоритм дозволяє як кількісно, так і якісно оцінювати ступінь «анормальності» кожного об’єкта.

Алгоритм LOF характеризується низкою суттєвих переваг, які роблять його ефективним інструментом для виявлення аномалій у складних просторових даних. Передусім він відзначається високою чутливістю до локальних відмінностей, оскільки оцінює поведінку кожної точки в контексті її найближчого оточення. Завдяки цьому LOF здатен ідентифікувати аномальні об’єкти навіть у випадках, коли загальний розподіл є неоднорідним або містить кластери різної густини. Крім того,

метод є непараметричним і не потребує припущень щодо нормальності чи симетрії даних, що забезпечує його універсальність у практичних застосуваннях. Важливою перевагою є також гнучкість у виборі метрики, що дозволяє використовувати геодезичні відстані й робить LOF придатним для аналізу просторових даних. Окремої уваги заслуговує його здатність природним чином виокремлювати локальні «аномальні острови», тобто виявляти нетипові точки всередині кластерів різної густини, що недоступно для багатьох глобальних методів детекції аномалій.

Разом з тим алгоритм має і певні обмеження. Обчислювальна складність зростає зі збільшенням обсягів даних, що може обмежувати його застосування у великих системах без додаткової оптимізації. Також LOF є чутливим до вибору параметра k, який визначає кількість сусідів і суттєво впливає на якість результатів. Крім цього, у випадку багатовимірних даних метод потребує попередньої нормалізації, оскільки різні масштаби ознак можуть спотворювати оцінки локальної щільності.

3.5 Практичне застосування математичних моделей

У межах розробленої вебсистеми для управління обліковими даними приватних компаній важливою складовою є можливість аналітичної обробки просторово прив’язаних відомостей, що стосуються локацій, товарів, техніки або інших сутностей, створених на основі відповідних шаблонів. Оскільки кожна сутність може бути прив’язана до певної локації, а локація може містити координати на карті, виникає потенціал для застосування математичних алгоритмів аналізу просторових даних. Використання таких алгоритмів дає змогу не лише відображати інформацію у графічній формі, але й автоматизувати виявлення закономірностей, ідентифікацію аномалій та підвищення ефективності бізнес-процесів.

Одним із ключових методів, що може бути реалізований у системі, є алгоритм DBSCAN, який належить до класу щільнісних методів кластеризації. Завдяки здатності працювати зі складними, нерівномірно розподіленими даними, DBSCAN дозволяє групувати локації або товари за їх географічною близькістю. Це дає змогу автоматично виявляти природні просторові кластери, такі як групи складів, техніки

або точок обслуговування, що фактично формують окремі логістичні зони. У контексті облікової системи це може бути корисно для аналізу регіонів підвищеної концентрації ресурсів, виявлення дублюючих локацій або пошуку територій із недостатнім покриттям. Важливим є також те, що DBSCAN дозволяє визначати ізольовані точки, які не належать жодному кластеру, що може вказувати на помилки введення координат, некоректні записи або атипове розташування одиниць техніки.

Другим ефективним підходом до обробки просторових даних є метод оцінки щільності ядра (KDE), який дозволяє формувати теплові карти (heatmaps). На відміну від кластеризації, яка зосереджується на групуванні точок за принципом їх близькості, KDE надає змогу оцінити загальну щільність розташування об’єктів у просторі. У системі управління обліком це відкриває широкі можливості для аналізу розподілу товарів, техніки або інших ресурсів. Heatmap дозволяє візуально виявляти осередки надмірної концентрації товарів, що може свідчити про неефективність розподілу ресурсів, або ж навпаки – знаходити «порожні зони», що потребують підсилення. У процесі управління запасами такі візуалізації дають можливість швидко оцінити, які регіони потребують додаткового постачання або оптимізації логістики, зменшують витрати на перевезення та покращують оперативне планування.

Третім алгоритмом, що може суттєво посилити функціональність системи, є метод локального фактора викидів (LOF), що застосовується для виявлення аномалій у структурі даних. LOF оцінює ступінь «відхилення» кожного об’єкта від його локального оточення, що дозволяє визначати нетипові записи. У межах описуваної системи алгоритм може бути використаний для виявлення помилково введених товарів, некоректно закріплених локацій або атипових змін у кількості ресурсів. Наприклад, якщо певний товар знаходиться у географічно неприродній для бізнесу зоні або належить до категорії, що не характерна для конкретного регіону, LOF визначить його як аномалію. Аналогічно, раптові зміни у кількості товарів на складі (надмірне збільшення або зменшення) також можуть бути оперативно виявлені як локальні відхилення. Це створює додатковий рівень контролю як за достовірністю

облікових даних, так і за потенційними ризиками, пов’язаними з втратою або переміщенням ресурсів.

У комплексі застосування DBSCAN, KDE та LOF створює цілісну математичну основу для автоматизованої просторової аналітики в системі управління обліком. DBSCAN забезпечує кластерне структурування даних, KDE формує глобальну картину просторової щільності, а LOF виявляє локальні аномалії та нетипові випадки. Усі ці підходи інтегруються з наявним функціоналом системи – можливістю створювати шаблони, формувати окремі одиниці товарів, закріплювати їх за локаціями та відображати на карті. В результаті вебсервіс отримує можливість не лише зберігати дані, але й автоматично аналізувати їхню структуру, що підвищує якість управлінських рішень, покращує контроль внутрішніх процесів та сприяє ефективнішому розподілу ресурсів у компанії.

Висновки до розділу

У даному розділі було детально розглянуто сучасні математичні методи аналізу просторових даних, що можуть бути інтегровані в систему управління обліком товарів, техніки та локацій приватних компаній. Особливу увагу приділено алгоритму DBSCAN як представнику щільнісної кластеризації, методу оцінки щільності ядра (KDE) та алгоритму локального фактора викидів (LOF). Їх застосування дає змогу розширити функціональні можливості вебсистеми шляхом автоматизованого виявлення закономірностей, оцінки концентрації об’єктів та ідентифікації аномальних випадків у структурі даних.

Алгоритм DBSCAN продемонстрував високу ефективність при роботі з географічно розподіленими даними, що характерно для локацій та об’єктів обліку. Завдяки здатності формувати кластери довільної форми та відокремлювати шумові точки, DBSCAN дозволяє виявляти природні зони бізнес-активності, групи складів чи техніки, а також визначати території, які можуть потребувати розширення інфраструктури або оптимізації логістики. На рівні прийняття рішень використання результатів кластеризації підтримує стратегічне планування, зокрема вибір місць для нових локацій чи маршрутизацію виїзних робіт.

Метод KDE, який базується на оцінці просторової щільності, забезпечує можливість формування теплових карт для візуального аналізу концентрації об’єктів. У контексті системи обліку це дозволяє швидко визначати області надмірного накопичення товарів, прогалини у розподілі ресурсів, а також регіони із підвищеним навантаженням на персонал або техніку. Візуалізація даних у вигляді heatmap істотно спрощує аналітичний процес та підсилює прийняття тактичних і оперативних рішень.

Алгоритм LOF, спрямований на виявлення локальних аномалій, дозволяє своєчасно ідентифікувати некоректні, нетипові або потенційно помилкові записи. У системі управління обліком LOF може сигналізувати про неправильне закріплення товарів за локаціями, помилки в координатах, нетипові зміни кількості ресурсів або інші аномальні сценарії. Це підвищує точність даних та сприяє зміцненню контролю за внутрішніми процесами підприємства.

Сукупне застосування DBSCAN, KDE та LOF формує комплексний підхід до математичної обробки просторових даних, який не лише забезпечує глибшу аналітику, але й безпосередньо підтримує функції управлінського ухвалення рішень. Інтеграція цих алгоритмів у систему обліку дозволяє переходити від пасивного зберігання даних до їх активного аналізу, що підвищує ефективність, гнучкість та конкурентоспроможність організації в умовах сучасного ринкового середовища.

РОЗДІЛ 4. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Вибір та обгрунтування засобів для реалізації проєкту

Під час проєктування серверної частини вебзастосунку було обрано технологію Phoenix, що дало змогу реалізувати масштабований, відмовостійкий та високопродуктивний програмний продукт [11]. Ключовими аргументами на користь Phoenix стала його структурна простота, високий рівень продуктивності та використання мови програмування Elixir, яка забезпечує функціональний підхід до розроблення та сприяє підвищенню читабельності й надійності коду. Висока швидкодія фреймворку пояснюється вбудованою підтримкою розподілених обчислень: Elixir працює поверх віртуальної машини Erlang, що дозволяє ефективно розпаралелювати процеси та забезпечувати міжпроцесну комунікацію. Властивість відмовостійкості досягається завдяки механізмам автоматичного відновлення – при виникненні помилки процес перезапускається у стабільному стані, що підвищує надійність системи загалом [12].

Архітектура серверної частини базується на патерні MVC, що забезпечує структурованість коду та зручність подальшого тестування. На рівні моделей реалізовано бізнес-логіку та операції маніпулювання даними; зокрема, модуль FormsManagementServer містить повний набір CRUD-операцій для роботи з формами. Для взаємодії з базою даних застосовується бібліотека Ecto, яка забезпечує захист від SQL-ін’єкцій та підтримує комбінування SQL-запитів із внутрішніми функціями Ecto для побудови складних запитів. Рівень контролерів виконує роль вхідної точки HTTP-запитів, які обробляються вбудованим вебсервером Cowboy; контролери передають дані до моделі та формують відповіді для рівня представлення. На рівні представлення здійснюється формування структури відповіді, зокрема JSON-об’єктів, що повертаються клієнтській частині. Наприклад, у модулі WebManagementServer визначено формат вихідних даних для локацій, а для рендерингу використовуються функції render та render_many модуля Phoenix.View.

У серверній частині реалізовано механізм аутентифікації на основі бібліотеки Guardian, яка забезпечує створення та валідацію JWT-токенів. Доступ до контролерів

можливий лише за умови передання валідного токена у заголовку Authorization, що гарантує належний рівень безпеки.

У ролі системи керування базами даних обрано PostgreSQL, оскільки вона забезпечує ефективну підтримку формату JSON/JSONB, що є критично важливим для роботи розробленої системи. Підтримка індексування JSONB-полів дає змогу оптимізувати пошук і підвищити швидкодію запитів [15].

Для клієнтської частини застосунку було обрано бібліотеку React, яка забезпечує компонентний підхід до розроблення інтерфейсів та високу продуктивність завдяки механізму Virtual DOM. Однією з ключових переваг React є використання Redux API, що дозволяє централізовано зберігати дані та передавати їх між компонентами без необхідності глибокої пропагації властивостей; у контексті зберігається, зокрема, інформація про типи форм, що використовуються на різних сторінках. Для здійснення HTTP-запитів створено спеціальний компонент Agent, який використовує бібліотеку Axios для обробки асинхронних операцій.

Інтерфейс користувача реалізовано з використанням бібліотеки Rewind UI, яка надає широкий набір готових компонентів. Використання Rewind UI забезпечує естетичний та уніфікований вигляд інтерфейсу, спрощує розробку адаптивного дизайну та підвищує загальну зручність роботи з графічними компонентами.

4.2 Технічна специфікація

4.2.1 Характеристика продукту

Дана вебсистема може надавати наступну функціональність:

1. Формування географічного об'єкта через інтерфейс введення з геопросторовими даними.
2. Конструювання гнучкої схеми для структури даних.
3. Фіксація текстових записів у календарі.
4. Генерація екземпляра форми на базі попередньо визначеного типу.
5. Створення асоціативного зв'язку форми із визначеними географічними областями.

4.2.2 Класи користувачів та їх характеристики

Функціональність системи передбачає реалізацію багаторівневої моделі управління доступом (Role-Based Access Control, RBAC). Ключовою сутністю в цій моделі є роль суперадміністратора, якому надаються повноваження зі створення нових користувачів та конфігурування відповідних ролей.

4.2.3 Обмеження проектування та реалізації

1. **Обмежений часовий ресурс.** Стислий термін реалізації продукту, що становив 3 місяці, вимагав пріоритизації функціоналу та застосування гнучких методологій.Можливість супроводу вебсервісу
2. **Обмеження обсягу функціоналу (MVP).** Фокусування виключно на базових функціональних вимогах, визначених у специфікації, що виключає впровадження некритичних для MVP (Minimum Viable Product) можливостей.
3. **Залежність від сторонніх бібліотек.** Залежність від зовнішніх відкритих бібліотек (React, Rewind UI, Guardian, Axios), що обумовлює ризик технологічної несумісності та необхідність постійного моніторингу їхніх оновлень.

4.2.4 Середовище функціонування

Експлуатаційне середовище системи є розподіленим і включає клієнтську та серверну платформи. Доступ до клієнтської частини здійснюється через стандартний веббраузер (зокрема, Edge або Chrome). Технічні вимоги до серверної платформи представлені у Таблиці 4.1.

Таблиця 4.1– Середовище функціонування серверної частини платформи

Вимога	**Характеристика**
Процесор	Intel Pentium G4600
Постійна пам’ять	2 GB
Операційна система	Linux

4.2.5 Припущення та залежності

Передбачається, що специфікація програмного продукту може бути переглянута у зв'язку з еволюцією його функціоналу, внесенням змін до чинного законодавства, або потребою у його конфігуруванні під особливі умови експлуатації.

4.2.6. Характеристики системи

4.2.6.1 Формування географічного об'єкта через інтерфейс введення з геопросторовими даними

4.2.6.1.1 Опис і пріоритет

Процедура ініціалізації локації дозволяє користувачеві виокремити фізичну область на карті, яка є операційною зоною для подальшого введення та управління даними в системі

Пріоритет характеристики - Високий

4.2.6.1.2 Функціональні вимоги

Req-1.1. Передбачити елементи введення інформації для реєстрації найменування, географічної адреси та категорії цільового об'єкта

Req-1.2. Забезпечити доступність глобального картографічного інтерфейсу для вибору необхідної області.

Req-1.3. Відобразити інструментарій географічного редагування, що включає такі об'єкти, як полігональні межі, графічні маркери та лінійні сегменти.

Req-1.4. Реалізувати функцію автоматизованого пошуку об'єкта на основі його назви.

Req-1.5. Після успішного завершення процесу створення, відповідна локація повинна бути візуалізована на основному картографічному інтерфейсі системи.

4.2.6.2 Конструювання гнучкої схеми для структури даних

4.2.6.2.1 Опис і пріоритет

Функція створення динамічного типу форми надає користувачеві інструментарій для визначення структури сутності, дані якої будуть зберігатися в системі, з подальшою можливістю асоціативного зв'язування цієї сутності з конкретною географічною локацією

4.2.6.2.2 Функціональні вимоги

Req-2.1. Наявність поля для занесення інформації про назву типу форми.

Req-2.2. Наявність модального вікна в якому користувач вибирає тип поля, вводить назву поля та додає відповідне поле до форми.

Req-2.3. Користувач повинен мати можливість створити нове поле з одним з наступних типів: текстове поле, числове поле, додатне числове поле, поле з випадаючим списком, поле з посиланням, поле для внесення дати.

Req-2.4. Можливість видалення поля зі списку.

Req-2.5. При успішному створені типу форми, нове значення повинно появитись у навігаційному меню.

4.2.6.3 Фіксація текстових записів у календарі

4.2.6.3.1 Опис і пріоритет

Система підтримує модуль календаря для внесення текстових нотаток з прив'язкою до часових параметрів. Інструментарій характеризується простотою взаємодії, дозволяючи вибирати як окремі дні/часові точки, так і проміжки часу для документування подій.

4.2.6.3.2 Функціональні вимоги

Req-3.1. Передбачити графічний компонент календаря для визначення моменту або часового діапазону для подальшого формування записів.

Req-3.2. Забезпечити варіативність відображення даних у компоненті календаря за такими режимами: місяці, тижні, окремі дні та у форматі хронологічного переліку.

Req-3.3. При активації існуючої події у календарі має з'являтися модальне вікно для отримання підтвердження на її видалення.

Req-3.4. Після успішної реєстрації нової події у календарі, вона повинна бути додана до загального переліку всіх зафіксованих подій.

4.2.6.4 Генерація екземпляра форми на базі попередньо визначеного типу

4.2.6.4.1 Опис і пріоритет

Функціональність створення форми на базі раніше визначеного типу забезпечує користувачеві можливість безпосереднього внесення даних, специфічних для його предметної області.

4.2.6.4.2 Функціональні вимоги

Req-4.1. Необхідно надати поле для внесення унікального ідентифікатора (назви) форми.

Req-4.2 Система повинна відображати повний набір успадкованих полів, визначених у базовому типі.

Req-4.3. Усі внесені дані мають пройти обов'язкову валідацію згідно з їхнім декларованим типом.

Req-4.4. Після успішної генерації, новий екземпляр форми повинен бути відображений у загальному списку.

4.2.6.5 Створення асоціативного зв'язку форми із визначеними географічними областями

4.2.6.5.1 Опис і пріоритет

Функція прив'язування створених форм до визначених локацій надає користувачеві можливість просторової контекстуалізації даних, що дозволяє ефективно асоціювати конкретні записи з відповідними фізичними областями та організовувати інформацію за географічним принципом.

4.2.6.5.2 Функціональні вимоги

Req-5.1. Інтерфейс створення форми повинен містити випадаючий список, який відображає доступні локації у межах поточної організації користувача.

Req-5.2. Необхідно забезпечити функціонал для асоціювання однієї форми з кількома організаціями

Req-5.3. При навігації на сторінку конкретної організації має бути відображений перелік, що містить посилання на всі форми, прив'язані до цієї організації.

4.2.7 Вимоги зовнішніх інтерфейсів

4.2.7.1 Програмні інтерфейси

Серверна частина програмного забезпечення спроектована для функціонування в операційному середовищі Linux. Доступ до клієнтської частини забезпечується через будь-який сучасний веббраузер, зокрема Chrome або Edge.

4.2.7.2 Комунікаційні інтерфейси

Для забезпечення коректного функціонування застосунку необхідне стабільне мережеве підключення (через Wi-Fi або мобільні дані). Взаємодія між клієнтським та серверним компонентами здійснюється відповідно до архітектури REST із використанням стандартних HTTP-методів (GET, POST, PUT, DELETE). Критичною вимогою для захисту даних є обов'язкове використання протоколу HTTPS як захищеної ітерації HTTP.

4.2.8 Нефункціональні вимоги

4.2.8.1 Вимоги продуктивності

Продуктивність системи залежатиме від навантаженості сервера в певний момент часу. Затримка відповіді сервера на дії користувача повинна бути обмежена до 3 секунд.

4.2.8.2 Вимоги надійності

Надійність програмного продукту повинна бути гарантована на двох архітектурних рівнях – клієнтському та серверному. На клієнтському рівні необхідно запобігти можливості виконання користувачем несанкціонованих операцій або доступу до обмежених ресурсів. У випадку ініціювання непередбачуваних дій, застосунок має надати відповідне повідомлення про помилку з чітким поясненням її причини. На серверному рівні критично важливо забезпечити валідацію всіх вхідних даних, які надходять від клієнта, перед їх збереженням у базі даних. У ситуаціях отримання некоректних даних або спроби виконання забороненої дії, сервер повинен згенерувати та надіслати відповідний код помилки, який буде коректно інтерпретований і відображений клієнтською частиною у вигляді інформативного повідомлення для користувача. Таким чином, дворівнева перевірка забезпечує як цілісність даних, так і захист системи від неналежного використання.

4.2.8.3 Вимоги безпеки

Для забезпечення належного рівня захисту даних та аутентифікаційних механізмів, критично важливим є впровадження комплексного наскрізного захисту. Уся комунікація між серверною та клієнтською частинами повинна здійснюватися виключно за допомогою захищеного протоколу HTTPS (Hypertext Transfer Protocol Secure). Це гарантує конфіденційність та цілісність даних, що передаються, шляхом

їх шифрування. Крім захисту комунікаційних каналів, передбачається посилений захист облікових даних: перед збереженням у базі даних паролі користувачів необхідно обробляти за допомогою надійних криптографічних хеш-функцій із застосуванням солі (salting). Це запобігає успішній реалізації атак за допомогою попередньо обчислених таблиць (Rainbow Table Attacks). Не менш важливим є впровадження моделі управління доступом на основі ролей (RBAC), що забезпечує авторизацію користувачів та надає доступ лише до тих ресурсів, які відповідають їхній ролі (наприклад, шляхом валідації JWT-токенів для кожного HTTP-запиту). Нарешті, усі дані, що надходять від клієнта, мають підлягати ретельній валідації на сервері, щоб запобігти ін'єкційним атакам (наприклад, SQL Injection та Cross-Site Scripting — XSS).

4.3 Опис роботи із системою

Для отримання доступу до вебзастосунку користувач повинен перейти за відповідним посиланням у будь-якому сучасному веббраузері, після чого система автоматично перенаправляє його на сторінку автентифікації. На цій сторінці необхідно ввести індивідуальні облікові дані (логін і пароль). У разі успішної автентифікації користувач спрямовується на головну сторінку сервісу.

Головна сторінка містить бічну навігаційну панель, розташовану з лівого боку інтерфейсу. Панель може бути згорнута за потреби шляхом натискання на піктограму у вигляді трьох горизонтальних ліній. Навігаційне меню структурується за тематичними розділами та містить посилання на основні функціональні модулі системи: карту, календар, сторінки локацій і працівників, перелік створених типів форм, а також секцію налаштувань.

У центральній частині головної сторінки розташована інтерактивна карта, на якій відображаються геопросторові дані всіх локацій, що належать організації, під обліковим записом якої здійснено вхід (рис. 4.1). Під час вибору конкретного полігону на карті з’являється модальне вікно з основною інформацією: назвою локації та її типом. За потреби користувач може перейти на сторінку локації, натиснувши на її назву у вікні.

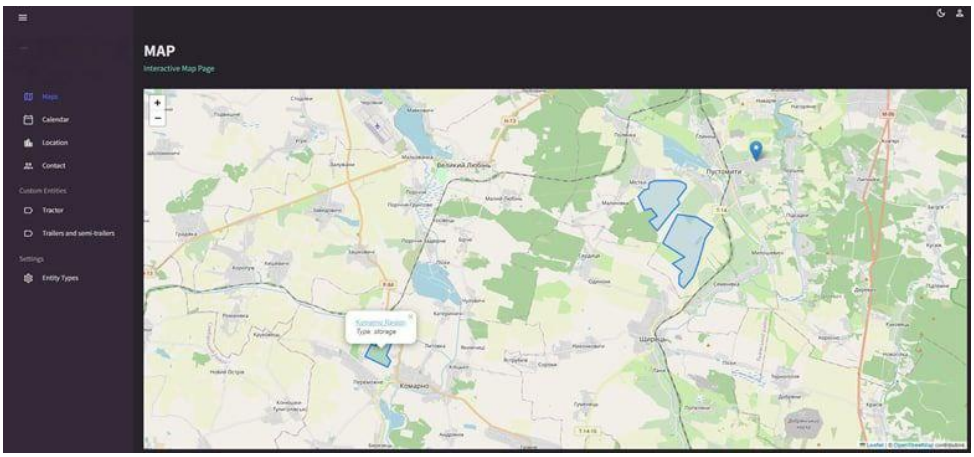

Рисунок 4.1 – Сторінка із відображенням локацій на мапі

При виборі в навігаційному меню опції «Locations» користувач перенаправляється на сторінку, що містить таблицю всіх локацій, створених у межах поточної організації. Крім перегляду даних у табличному форматі, система надає можливість експортувати їх у вигляді CSV-файлу. Для цього необхідно скористатися гіперпосиланням «Export» та обрати пункт «Download as CSV».

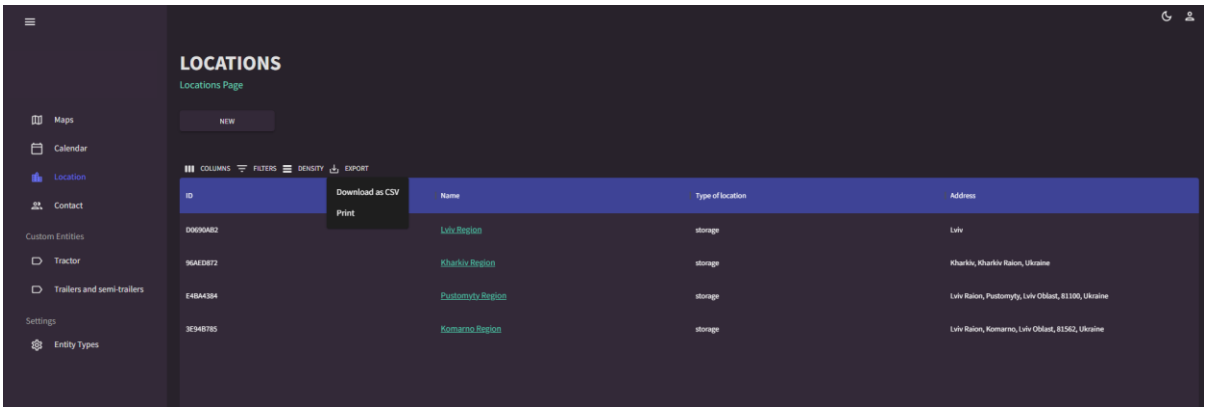

Рисунок 4.2 – Сторінка із списком локацій

Після натиснення на кнопку «New» система перенесе користувача на сторінку створення нової локації, де користувачу необхідно буде заповнити відповідні поля, які описують локацію, а також, при бажанні вибрати полігон, маркер, або лінію використовуючи панель інструментів, а також є можливість видалити, або відредагувати вибрану область на мапі (рис. 4.3).

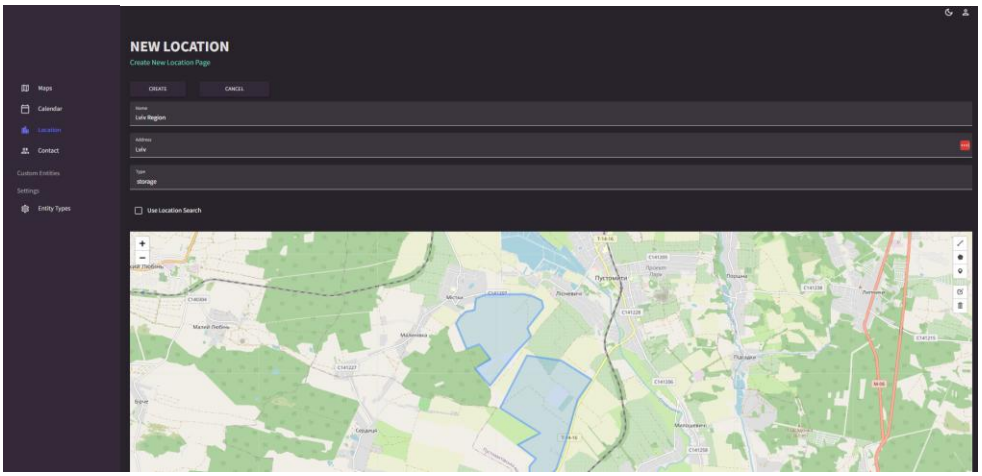

Рисунок 4.3 – Сторінка створення нової локації

При натиснені на чекбокс «Use Location Search» у користувача появиться текстове поле для автоматичного пошуку місцевості (рис. 4.4.).

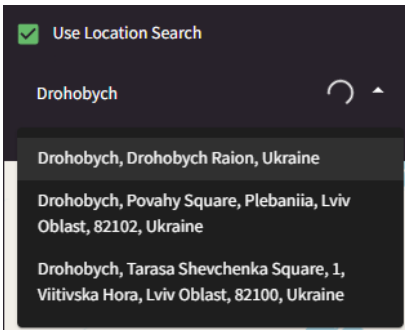

Рисунок 4.4 – Використання автоматичного пошуку місцевості

Поле введення назви локації реалізовує функцію автозаповнення (autocomplete): після початку введення користувачеві пропонується випадаючий список доступних варіантів. Вибір одного з елементів цього списку ініціює автоматичне масштабування та центрування картографічного інтерфейсу на вибраній географічній місцевості.

Після підтвердження операції на сторінці створення локації (натискання кнопки «Create»), система повинна надати користувачеві візуальний зворотний зв'язок у формі сповіщення (notification) про успішне завершення процесу.

При активації опції «Entity Types» у навігаційному меню відбувається перехід на сторінку, яка відображає вичерпний перелік усіх доступних типів форм у системі. Ця сторінка за своєю структурою є ідентичною табличному відображенню локацій. Для створення нового типу форми користувач має перейти до відповідного розділу.

На цій сторінці необхідно внести назву нового типу та визначити необхідний набір полів для сутності, яку він бажає створити (рис. 4.5.).

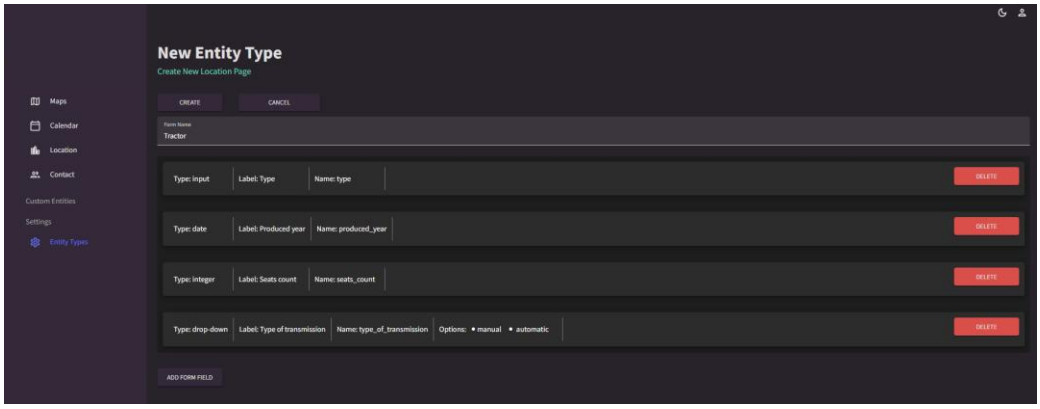

Рисунок 4.5 – Сторінка створення нового типу форм

Після натискання кнопки «ADD FORM FIELD» користувачеві відкривається модальне вікно, у якому необхідно обрати тип поля та задати його назву (рис. 4.6).

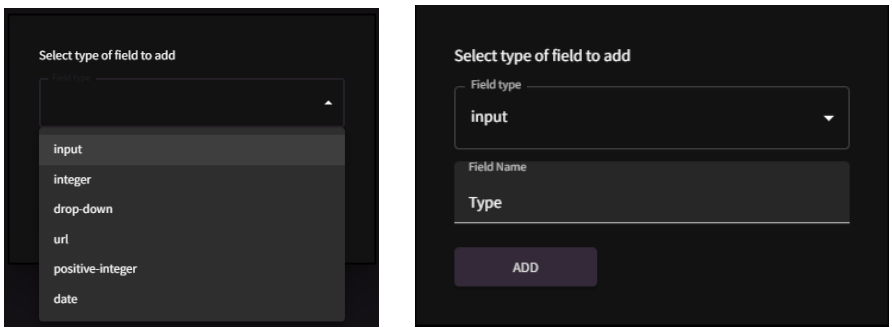

Рисунок 4.6 – Модальне вікно для вибору типу поля

Після натискання кнопки «Add» створене поле з’являється у відповідному списку. Далі, після натискання кнопки «Create» на сторінці формування нового типу форми, користувач отримує сповіщення про успішне створення, а також бачить новий елемент у лівому навігаційному меню (рис. 4.7).

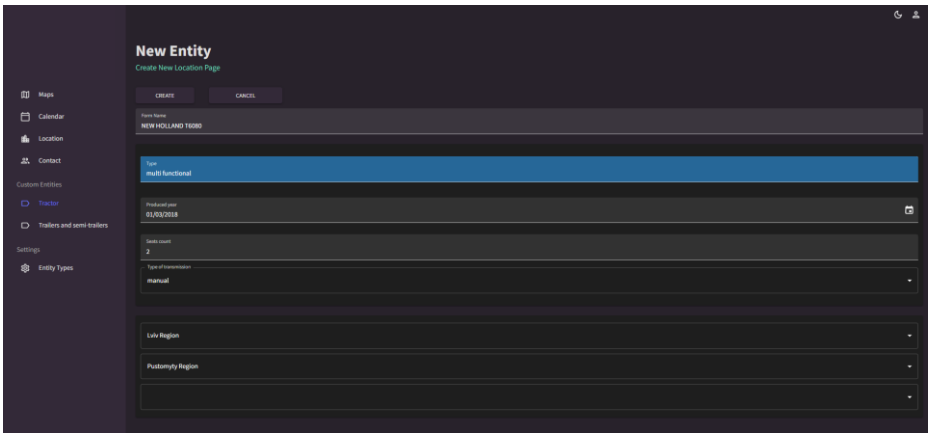

Рисунок 4.7 – Сторінка створення нової форми

Після успішного створення типу форми користувач отримує можливість створити її екземпляр, заповнивши всі необхідні дані у відповідних полях та обравши локацію за допомогою випадаючих списків (рис. 4.7). Після підтвердження створення нової форми система автоматично перенаправляє користувача на сторінку, де відображено всі форми відповідного типу (рис. 4.8).

Рисунок 4.8 – Сторінка всіх екземплярів конкретного типу

Після виконання попередніх дій із прив’язування форми до локації на головній сторінці відповідної локації відображається внутрішнє навігаційне меню, у якому містяться всі форми, приєднані до цієї локації (рис. 4.9).

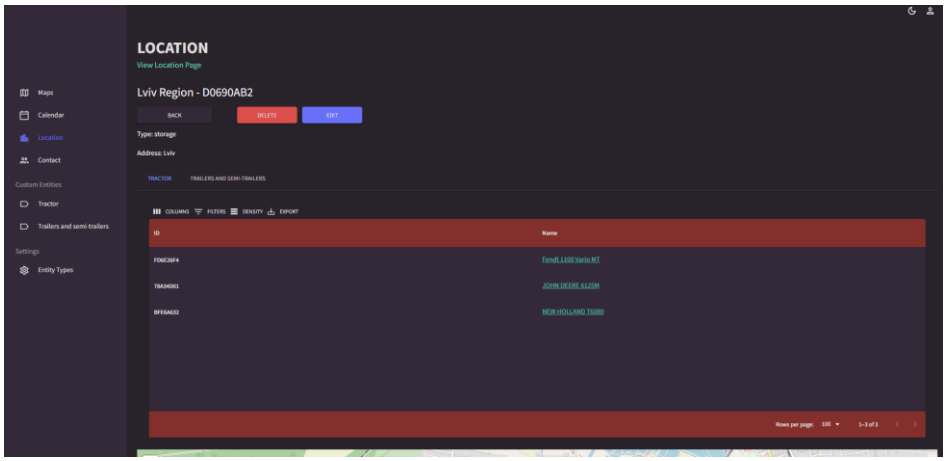

Рисунок 4.9 – Сторінка всіх форм для локації

У системі передбачено стандартну сутність «Contacts», яка забезпечує можливість внесення інформації про працівників та подальшого їх прив’язування до відповідних локацій.

Висновки до розділу

У цьому розділі було успішно завершено етап архітектурного проектування та технічної специфікації вебплатформи. Технологічний стек, обраний для реалізації, забезпечує високу масштабованість, відмовостійкість та продуктивність. Серверний компонент побудований на фреймворку Phoenix (мова Elixir), що використовує віртуальну машину Erlang (BEAM) для ефективного розподілу обчислень та механізмів самовідновлення. Архітектура системи відповідає патерну MVC та стандартам REST Web API. У якості СКБД обрано PostgreSQL, яка є критично важливою для роботи з динамічними даними завдяки підтримці JSONB та ефективному індексуванню. На рівні взаємодії з даними застосовано бібліотеку Ecto (підхід Code First), що підвищує безпеку, запобігаючи SQL-ін'єкціям. Клієнтський компонент реалізований як односторінковий застосунок (SPA) на базі React з використанням Redux для централізованого управління станом та бібліотеки Rewind UI для уніфікованого графічного інтерфейсу. Безпека комунікації забезпечується протоколом HTTPS та механізмом JWT-аутентифікації (Guardian). Технічна специфікація (п. 4.2), сформована в межах розділу, детально визначає функціональні вимоги (створення локацій, динамічних форм, нотаток), класи користувачів та встановлює обмеження реалізації і середовище функціонування. Таким чином, обрані архітектурні рішення, опис роботи із системою та деталізована технічна документація створюють надійну та гнучку основу для подальшої розробки, гарантуючи відповідність продукту високим атрибутам якості.

РОЗДІЛ 5. РОЗРОБЛЕННЯ СТАРТАП-ПРОЄКТУ

5.1. Опис ідеї проєкту

Ідея стартап-проєкту полягає у створенні вебзастосунку, який забезпечує можливість динамічного формування даних різних типів та інтеграції цих даних із фізичними об’єктами на карті. Система поєднує функціональність конструктора форм із можливостями картографічного менеджменту, що дає змогу користувачам створювати структури даних відповідно до конкретних потреб підприємства та відображати їх у просторовому контексті. Такий підхід робить застосунок універсальним інструментом для управління інформаційними ресурсами, що використовуються у приватних компаніях.

Сфера застосування проєкту є доволі широкою, оскільки система не прив’язана до конкретного домену. Вона може бути використана:

- у малих та середніх підприємствах для створення внутрішніх реєстрів, систем інвентаризації або контролю ресурсів;
- у сфері послуг – для ведення клієнтських карток, запису зустрічей, обліку сервісів або устаткування; у логістичних або транспортних компаніях для прив’язки об’єктів (склади, маршрути, точки доставки) до географічних координат;
- у будівельній та інженерній галузях – для фіксації стану об’єктів на місцевості;
- у сфері нерухомості – для роботи з об'єктами та їх атрибутами без використання складних GIS-платформ.

Універсальність інструменту дає змогу адаптувати систему до різних бізнес-процесів, зберігаючи гнучкість у налаштуванні структури даних та їх візуальному представленні.

Використання запропонованого стартап-проєкту забезпечує користувачам низку переваг:

- **Гнучкість роботи з даними**. Користувач може самостійно створювати структури даних та формувати потрібні сутності без необхідності залучення додаткових розробників.
- **Просторове представлення інформації**. Дані можуть бути прив’язані до об’єктів на карті, що полегшує аналіз, планування та прийняття рішень у сферах, пов’язаних з геолокацією.
- **Зменшення рутинної роботи**. Система автоматизує створення, перегляд і оновлення даних, що скорочує час виконання регулярних операцій.
- **Бізнес-адаптивність**. Проєкт може бути легко інтегрований у внутрішню інфраструктуру підприємства завдяки масштабованості та можливості моделювати дані під конкретні потреби.
- **Безпека даних**. Усі дані зберігаються у захищеному середовищі з дотриманням базових принципів конфіденційності й контролю доступу.

Під час аналізу ринку було виявлено, що наявні рішення, такі як конструктори форм (WuFoo, Typeform, Cognito Forms) або системи картографічного менеджменту (ArcGIS, Mapbox), зазвичай фокусуються лише на одній складовій – створенні структурованих форм або роботі з географічною інформацією. Жоден із рассмотрених продуктів не пропонує нативної та повноцінної інтеграції цих функцій у межах єдиного застосунку. Запропонований стартап вирізняється саме поєднанням: динамічного створення форм та інтерактивного картографічного модуля у межах однієї платформи, що робить його унікальним продуктом без прямого аналога у відповідному сегменті ринку. Така комбінація дозволяє користувачам працювати одночасно з атрибутною і геопросторовою інформацією, що недоступно у більшості існуючих сервісів.

Таблиця 5.1 – Порівняння непрямих сервісів-аналогів

Функціональність	**WuFoo**	**Cognito Forms**	**ArcGis**
Тип системи	Конструктор форм	Конструктор форм	Геоінформаційна система
Гнучкість створення даних	Середня-шаблонний підхід	Висока – багато типів полів	Обмежена
Картографічні можливості	Відсутні	Відсутні	Дуже широкі ГІС-функції
Інтеграції	Відсутні	Інтеграції із платіжними системами	Потужн професійні інтеграції
Вартість	Freemium, деякий функціонал платний	Помірна, залежить від плану	Дуже висока
Основні недоліки	Застарілий інтерфейс	Відсутність картографічного модуля	Висока вартість, складність користування

5.2. Аналіз технологічних можливостей реалізації ідей проєкту

Реалізація запропонованої ідеї стартап-проєкту передбачає створення вебзастосунку, який поєднує функціональність динамічного конструктора форм та інструментів картографічного менеджменту. Для цього необхідно проаналізувати доступні технології, оцінити їх готовність до використання та визначити, наскільки вони відповідають технічним вимогам проєкту.

Побудова системи ґрунтується на використанні сучасної вебархітектури, яка передбачає розроблення серверної та клієнтської частин. Серверна частина може

бути реалізована за допомогою фреймворку Phoenix, що працює на функціональній мові Elixir. Дана технологія добре підходить для створення високонавантажених та масштабованих вебсистем, забезпечуючи високу продуктивність та зручні можливості для обробки одночасних запитів. Клієнтська частина застосунку, у свою чергу, може бути реалізована на основі бібліотеки React у поєднанні з мовою TypeScript, що дає змогу створювати інтерактивний інтерфейс та забезпечує високу гнучкість у роботі з динамічними компонентами.

Функціональність конструктора форм може бути реалізована шляхом використання вже наявних вебтехнологій, зокрема React-компонентів для динамічної генерації полів, бібліотек для валідації форм та моделювання структури даних. Подібні технології є добре опрацьованими, активно підтримуються спільнотою та не потребують розроблення нових інструментів з нуля.

Для реалізації картографічної частини можуть бути використані сучасні картографічні бібліотеки, такі як Leaflet, Mapbox GL JS або OpenLayers. Ці інструменти дозволяють працювати з географічними даними, наносити об’єкти на карту, обробляти координати та забезпечувати інтерактивність елементів. Усі зазначені технології є відкритими або мають доступні тарифні плани, що робить їх придатними для використання в рамках проєкту без додаткових витрат на розробку власного картографічного рушія.

Для зберігання та обробки даних доцільно використовувати реляційну систему PostgreSQL, яка підтримує розширення PostGIS, що значно розширює можливості роботи з геопросторовими даними. PostgreSQL є стабільною, масштабованою та загальнодоступною технологією, що повністю відповідає потребам проєкту. Розгортання бази даних у Docker-контейнерах забезпечує зручність у конфігурації, розширенні та тестуванні системи.

Усі перелічені технології є поширеними, активно підтримуються розробницькими спільнотами та знаходяться у відкритому доступі. Вони не вимагають створення спеціалізованих інструментів або пропрієтарних рішень, що підтверджує технологічну здійсненність проєкту.

З огляду на проведений аналіз можна зробити висновок, що реалізація проєкту є технічно можливою. Обрані програмні засоби – Phoenix/Elixir для серверної частини, React/TypeScript для клієнтської, PostgreSQL із PostGIS для зберігання даних та картографічні бібліотеки для візуалізації – є доступними, сучасними та повністю здатними забезпечити виконання необхідного функціоналу. Відтак проєкт може бути успішно реалізований із використанням наявних технологій без потреби у створенні складних додаткових рішень.

5.3. Аналіз ринкових можливостей запуску стартап-проєкту

У сучасних умовах ринок рішень для управління даними та внутрішньої цифровізації бізнес-процесів демонструє стале зростання. Підприємства приватного сектору все частіше впроваджують вебсистеми для обліку ресурсів, інвентаризації, персоніфікації даних та організації внутрішніх робочих процесів. Попит на такі системи зумовлений потребою підвищення ефективності бізнесу, переходом до концепції «data-driven management» та загальною цифровою трансформацією підприємств [17].

Особливо актуальним є сегмент малого та середнього бізнесу, який потребує недорогих, гнучких та масштабованих інструментів для роботи з даними. Водночас у сфері послуг, логістики, будівництва та нерухомості спостерігається зростаюча потреба в рішеннях, що дозволяють поєднувати традиційний облік із картографічними можливостями, оскільки просторовий контекст відіграє важливу роль у щоденних процесах цих галузей [18].

На основі аналізу ринку можна виділити кілька основних груп потенційних користувачів:

- **Малі та середні підприємства** – потребують універсального інструменту для обліку ресурсів, клієнтської інформації, внутрішніх процесів та документування.
- **Логістичні та транспортні компанії** – зацікавлені у можливості прив’язки даних до геолокації, оптимізації маршрутів та контролю точок доставки.

- **Будівельні та інженерні організації** – використовують просторові дані для моніторингу об’єктів, ведення технічних журналів та перевірок.
- **Компанії сфери послуг** – потребують можливості реєстрації клієнтів, формування заявок та інтерактивної взаємодії з об’єктами на мапі.
- **Організації, що працюють з польовими даними** – для фіксації стану об’єктів, інфраструктури, територій або обладнання.

Для кожної групи користувачів ключовими вимогами є гнучкість створення сутностей, наочність представлення даних, надійність та доступність системи.

У процесі дослідження ринкового середовища доцільно виділити чинники, що сприяють та перешкоджають впровадженню проєкту. До факторів, що сприяють впровадженню такої системи ми можемо віднести:

- Загальний тренд на цифровізацію підприємств.
- Попит на інструменти, здатні оптимізувати роботу з великими обсягами даних.
- Відсутність універсальних рішень, які одночасно поєднують картографічні та формотворчі можливості.
- Зростання кількості компаній, які переходять до хмарних сервісів та вебплатформ.

До факторів, що перешкоджають впровадженню такої системи можна віднести:

- Конкуренція з окремими нішевими продуктами (конструктори форм, GIS-системи).
- Обмежена цифрова грамотність частини цільової аудиторії.
- Потенційні витрати на первинне впровадження та навчання персоналу.
- Наявність окремих безкоштовних інструментів, що можуть частково задовольнити базові потреби підприємств.

5.4. Розроблення ринкової стратегії проєкту

Формування ринкової стратегії є ключовим етапом у підготовці проєкту до виходу на ринок, оскільки визначає спосіб позиціонування продукту та підходи до взаємодії з потенційними споживачами. На цьому етапі першочерговим завданням є встановлення меж ринку, на якому планується просування продукту, а також виокремлення сегментів споживачів, для яких він має найбільшу практичну цінність. Сегментація дозволяє структурувати ринок відповідно до відмінностей у потребах, рівні технологічної готовності споживачів та характері їхньої взаємодії з цифровими інструментами.

Після аналізу структури ринку, потенційних користувачів та їхніх характеристик постає питання вибору стратегії охоплення ринку. У загальному вигляді ринкова стратегія може базуватися на трьох концептуальних підходах: концентрованому, диференційованому або масовому маркетингу. Кожен із них має різний рівень спеціалізації продукту, цільової спрямованості та охоплення.

З огляду на результати сегментації, найбільш доцільною для даного проєкту є стратегія диференційованого маркетингу. Обґрунтування вибору базується на таких міркуваннях.

По-перше, різні сегменти ринку характеризуються суттєвими відмінностями у способах використання цифрових інструментів. Частина споживачів орієнтується на базову функціональність та простоту інтерфейсу, тоді як інші очікують розширених можливостей або спеціалізованих модулів. Використання диференційованої стратегії дозволяє адаптувати продукт до особливостей окремих сегментів без необхідності радикальних змін у його фундаментальній структурі.

По-друге, орієнтація на декілька сегментів одночасно знижує залежність проєкту від одного ринкового напряму. Це забезпечує більшу стійкість у разі зміни ринкових умов, коливання попиту або появи нових конкурентів. Диференційований підхід дозволяє формувати окремі пропозиції для різних груп користувачів, вибудовуючи гнучкі комунікаційні моделі.

По-третє, застосування масового маркетингу в даному випадку є недоцільним, оскільки ринок програмних засобів характеризується високим рівнем

роздрібненості потреб. Натомість концентрований маркетинг, попри його точність, може обмежити потенціал зростання проєкту на ранніх етапах, ускладнивши розширення продукту на нові сегменти.

Узагальнюючи вищезазначене, стратегія диференційованого маркетингу забезпечує оптимальний баланс між точністю ринкового позиціонування, варіативністю пропозиції та можливістю масштабування продукту. Такий підхід дозволяє адаптувати функціональні особливості проєкту до вимог окремих категорій споживачів і водночас підтримувати цілісність продукту в рамках єдиної платформи, що є важливим для його сталого розвитку.

5.5. Маркетингова програма стартап-проєкту

Маркетингова програма охоплює концепцію товару, політику збуту, заходи з просування та базові підходи до формування ціни. Оскільки в межах проєкту обрано стратегію диференційованого маркетингу, маркетингова програма має враховувати особливості кількох ринкових сегментів і пропонувати адаптовані рішення для кожного з них.

Концепція товару полягає у створенні вебсистеми, яка поєднує інструменти для динамічного формування структурованих даних та можливості відображення інформації у просторовому контексті. Основним завданням продукту є надання користувачам інструменту, здатного адаптуватися до специфічних процесів кожної компанії, не обмежуючи їх заздалегідь встановленою структурою чи суворою предметною областю. У межах обраної ринкової стратегії продукт може мати різні модифікації функціональності залежно від сегмента: від базового набору функцій до розширених можливостей взаємодії з даними.

Політика збуту передбачає дистрибуцію продукту у вигляді вебсервісу, доступного через модель підписки (Software as a Service). Такий підхід дає змогу забезпечити доступність сервісу для різних груп користувачів, починаючи від невеликих організацій і завершуючи компаніями з ширшим спектром завдань, пов’язаних із роботою з даними. Доступ до функціоналу може бути структуровано за рівнями, що відповідають потребам окремих сегментів: базовий доступ для

користувачів із мінімальними вимогами, розширений – для компаній, що потребують персоналізації та інтеграцій.

Система збуту передбачає використання онлайн-каналів, зокрема офіційного вебресурсу продукту, партнерських платформ та спеціалізованих IT-каталогів. Для споживачів, що потребують демонстрації можливостей продукту перед придбанням, передбачено можливість ознайомлення через тестові або демонстраційні версії.

Заходи з просування спрямовані на поінформованість цільових сегментів про переваги продукту та формування зацікавленості у його використанні. У межах стратегії диференційованого маркетингу просування має враховувати особливості різних груп споживачів. Основними інструментами просування є: контент-маркетинг, включно з освітніми матеріалами щодо можливостей роботи з даними; застосування цільової комунікації через професійні платформи, орієнтовані на певні галузі; пошукова оптимізація та онлайн-реклама для ширшого охоплення аудиторії; участь у галузевих заходах, виставках і форумах для формування довіри до продукту серед професійних користувачів. Особливе значення має демонстрація практичних сценаріїв застосування продукту, що дозволяє адаптувати комунікаційні повідомлення відповідно до потреб окремих сегментів.

Попередній аналіз можливостей ціноутворення передбачає застосування моделі, що поєднує цінову диференціацію та гнучкість у формуванні тарифів. Оскільки проєкт орієнтується на різні групи користувачів, доцільним є використання декількох тарифних планів із різним рівнем функціональності. Така модель дозволяє адаптувати вартість відповідно до цінності, яку продукт створює для конкретного сегмента споживачів. Базовий тариф передбачає доступ до основних функцій продукту, тоді як розширені тарифи містять додаткові можливості, пов’язані з персоналізацією, інтеграціями або підвищеною гнучкістю системи. Для окремих категорій користувачів може бути передбачено індивідуальне ціноутворення залежно від масштабу застосування системи.

Висновки до розділу

У межах даного розділу було сформовано комплексну маркетингову стратегію стартап-проєкту, яка ґрунтується на результатах аналізу ринкового середовища, характеристиках потенційних споживачів та особливостях конкурентної ситуації. Здійснена сегментація ринку дала змогу виокремити групи користувачів, для яких розроблений продукт має найбільшу цінність.

На основі проведеного аналізу було обґрунтовано доцільність використання стратегії диференційованого маркетингу, яка забезпечує збалансований підхід до охоплення кількох ринкових сегментів та дозволяє адаптувати продукт до різних сценаріїв використання без втрати його цілісності. Подальший розгляд компонентів маркетингового комплексу дав змогу сформувати цілісну маркетингову програму, що охоплює концепцію товару, політику збуту, заходи з просування та підходи до ціноутворення.

Таким чином, розроблена маркетингова стратегія створює підґрунтя для ефективного позиціонування стартап-проєкту на ринку та визначає ключові напрями його подальшої комерційної реалізації. Вона враховує як особливості потреб цільових груп, так і динаміку ринкових тенденцій, що забезпечує стійкі передумови для успішного впровадження продукту та його подальшого розвитку.

ВИСНОВКИ

У магістерській кваліфікаційній роботі здійснено комплексне дослідження, спрямоване на розроблення вебплатформи для систематизації та ефективного управління корпоративними даними приватних компаній. У ході роботи було виконано аналіз проблемної області, визначено функціональні вимоги до програмного продукту, спроєктовано його архітектуру та реалізовано програмне забезпечення, що поєднує інструменти динамічного створення форм із елементами картографічниих систем.

На основі вивчення сучасного стану ринку було встановлено, що наявні вебсервіси зазвичай орієнтовані або на генерацію форм, або на роботу з геоданими й не містять інтегрованого рішення, яке поєднує ці функціональні складові в межах однієї платформи. Це підтверджує актуальність створення універсальної системи, що дозволяє приватним компаніям здійснювати облік та аналіз даних у просторовому контексті.

У роботі сформовано інформаційну модель системи, визначено основні об’єкти дослідження, розроблено UML- та ER-діаграми, які описують структурні та поведінкові характеристики платформи. Визначено вхідні та вихідні дані, сформульовано обмеження та зв’язки між сутностями, що забезпечує логічну цілісність і коректність функціонування системи.

Під час проєктування було обґрунтовано вибір клієнт-серверної архітектури. Серверну частину реалізовано з використанням Phoenix/Elixir, що забезпечує високий рівень продуктивності, масштабованості та відмовостійкості. Клієнтська частина створена на основі React/TypeScript із застосуванням архітектури SPA, що гарантує швидку та інтерактивну взаємодію користувача із системою. Для зберігання даних використано PostgreSQL з PostGIS, що дозволяє ефективно працювати з геопросторовими даними.

У математичному модулі досліджено та обґрунтовано застосування алгоритму DBSCAN для кластеризації просторових об’єктів. Порівняльний аналіз з алгоритмом k-means показав переваги DBSCAN у випадках нерівномірного

розподілу даних, наявності шумів та потреби у формуванні кластерів довільної форми. Продемонстровано можливість практичного використання кластеризації для визначення зон підвищеної бізнес-активності, оптимізації логістичних маршрутів та планування роботи компанії.

У програмній частині реалізовано всі ключові функціональні модулі: створення локацій із геометрією на карті, формування динамічних типів форм, створення та прив’язка екземплярів форм до локацій, управління подіями календаря, робота з користувачами, ролями та правами доступу. Розроблено зручний та інтуїтивний інтерфейс на основі прототипів, створених у Figma, що відповідає принципам ергономіки та забезпечує комфортну взаємодію користувачів із системою.

Створений вебзастосунок може бути впроваджений за моделлю SaaS, що підвищує його комерційний потенціал та робить продукт доступним для широкого кола підприємств. Гнучка архітектура та модульність системи дозволяють надалі розширювати її функціональність, інтегрувати платіжні сервіси, інструменти аналітики або зовнішні API.

Розроблена вебплатформа є конкурентоспроможним рішенням, яке забезпечує приватним компаніям ефективні засоби управління структурованими та геопросторовими даними, сприяє оптимізації бізнес-процесів і має значний потенціал для подальшого розвитку та практичного впровадження.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Digital Transformation Strategy for Growth [Електронний ресурс] – Режим доступу до ресурсу: https://pxp.io/blog/digital-transformation-strategy.
2. Digitalisation in Europe – 2025 edition [Електронний ресурс] – Режим доступу до ресурсу: https://ec.europa.eu/eurostat/web/interactive-publications/digitalisation-2025
3. Vărzaru A. A., Bocean C. G. Digital Transformation and Innovation: The Influence of Digital Technologies on Turnover from Innovation Activities and Types of Innovation. Systems. 2024. Т. 12, № 9. С. 359.
4. Talo M. C., Emanuel A. W. R. Systematic Review of Enterprise Resource Planning (ERP) System Implementation in Organizations: Challenges and Successes to Company Performance. 2025. Т. 10, №. 2. С. 1–11.
5. Software Development Cost Breakdown for Businesses [Електронний ресурс] – Режим доступу до ресурсу: https://gloriumtech.com/software-development-cost-breakdown-for-businesses.
6. MVC Architecture Explained: Model, View, Controller [Електронний ресурс] – Режим доступу до ресурсу: https://www.codecademy.com/article/mvc-architecture-model-view-controller
7. Ecto Lessons [Електронний ресурс] – Режим доступу до ресурсу: https://elixirschool.com/en/lessons/ecto.
8. Естер М., Крігель Г.-П., Сандер Й., Сюй С. Алгоритм щільнісної кластеризації DBSCAN для виявлення кластерів у великих просторових базах даних // Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96). — Portland, 1996. — С. 226–231.
9. Сільверман Б. В. Оцінювання густини для статистики та аналізу даних : монографія / пер. з англ. — Лондон : Chapman & Hall, 1986. — 176 с.
10. Бройніг М. М., Крігель Г.-П., Нг Р. Т., Сандер Й. Виявлення локальних аномалій методом LOF // ACM SIGMOD Conference, 2000 [Електронний ресурс]. — URL: https://doi.org/10.1145/342009.335388

11. Тейт Б., ДеБенедетто С. Programming Phoenix LiveView: Interactive Elixir Web Applications Without Writing Any JavaScript / B. Tate, S. DeBenedetto. – Raleigh: The Pragmatic Bookshelf, 2021. – 280 с.
12. Томас Д. Programming Elixir 1.6: Functional | Concurrent | Pragmatic | Fun / D. Thomas. – Raleigh: The Pragmatic Bookshelf, 2018. – 384 с.
13. TypeScript Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.typescriptlang.org/docs/handbook/intro.html.
14. Leaflet [Електронний ресурс] – Режим доступу до ресурсу: https://leafletjs.com/reference.html#map-factory.
15. Молінаро А., де Грааф Р. SQL Cookbook: Query Solutions and Techniques for Database Developers. – 2-ге вид. – Sebastopol: O'Reilly Media, 2020. – 800 с.
16. Leaflet [Електронний ресурс] – Режим доступу до ресурсу: https://leafletjs.com/reference.html#map-factory.
17. Бриньолфссон Е., Гітт Л. М., Кім Х. Х. Strength in Numbers: How Does Data-Driven Decisionmaking Affect Firm Performance? / E. Brynjolfsson, L. M. Hitt, H. H. Kim. – Cambridge: MIT Sloan School of Management Working Paper, 2011. – (SSRN Scholarly Paper No. 1819486).
18. Geographic Information Systems Market Global Report, 2024–2025–2033: Growing Demand for Cutting-edge GIS Solutions in Real Estate, Transportation, Military, and Agriculture. / Research and Markets. – GlobeNewswire. – 2025. – 28 січ.

ДОДАТОК А

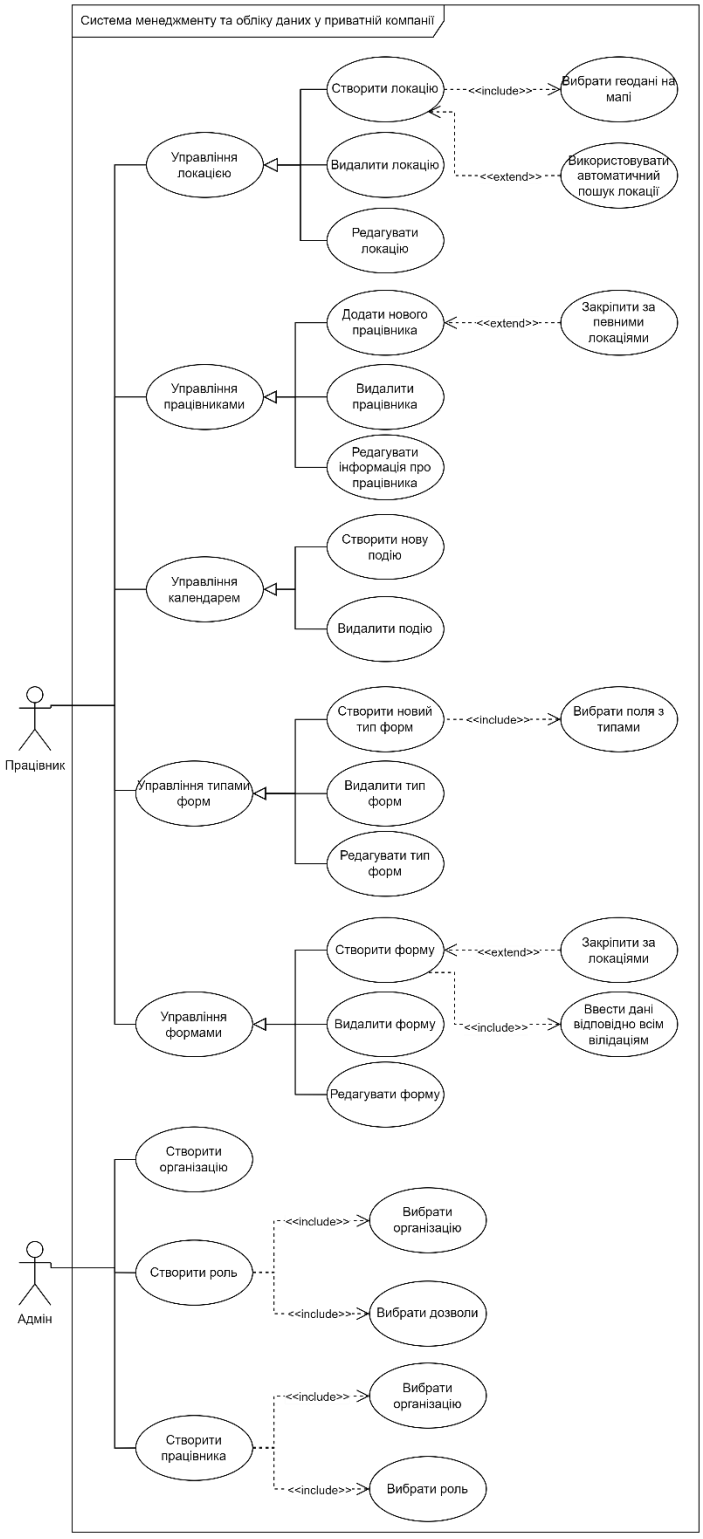

Рисунок А.1 – Діаграма прецедентів

ДОДАТОК Б

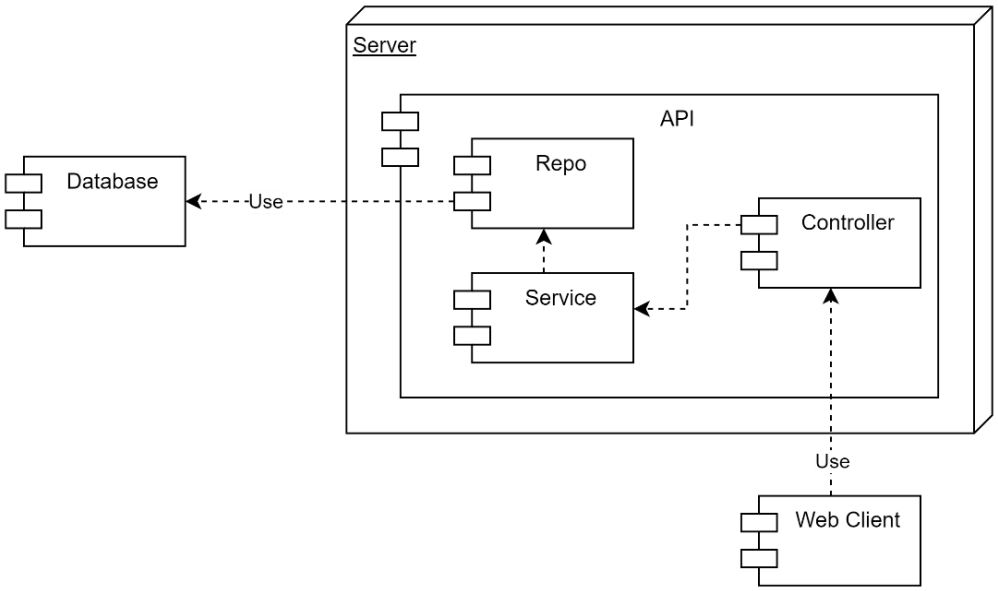

Рисунок Б.1 – Діаграма компонент

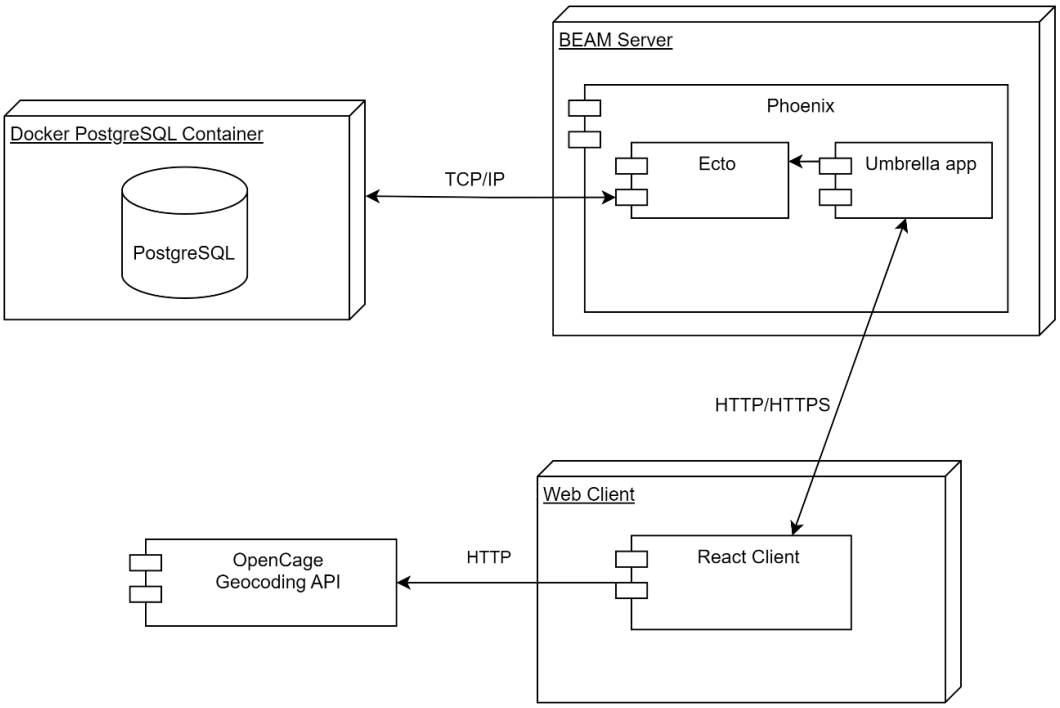

Рисунок Б.2 – Діаграма розгортання системи

ДОДАТОК В

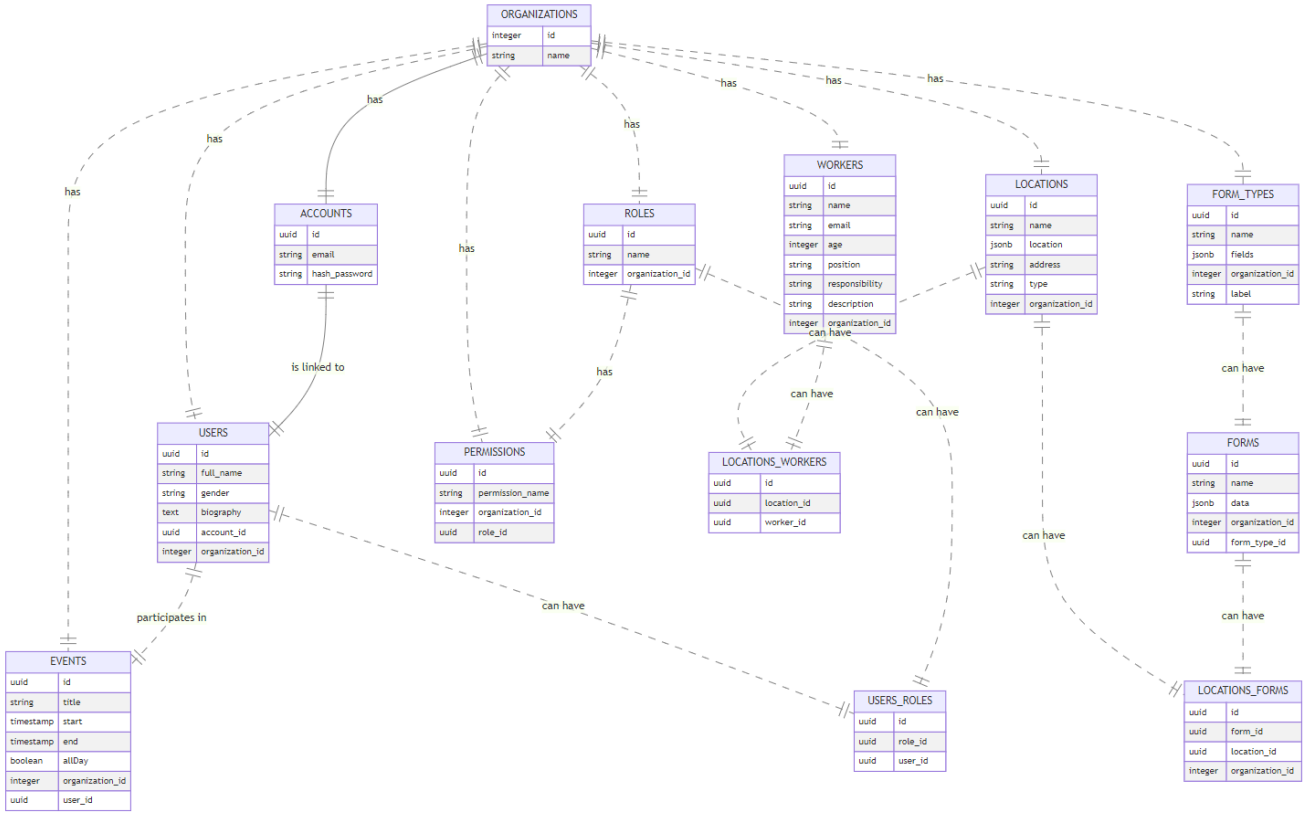

Рисунок В.1 – Діаграма логічної моделі бази даних

ДОДАТОК Г

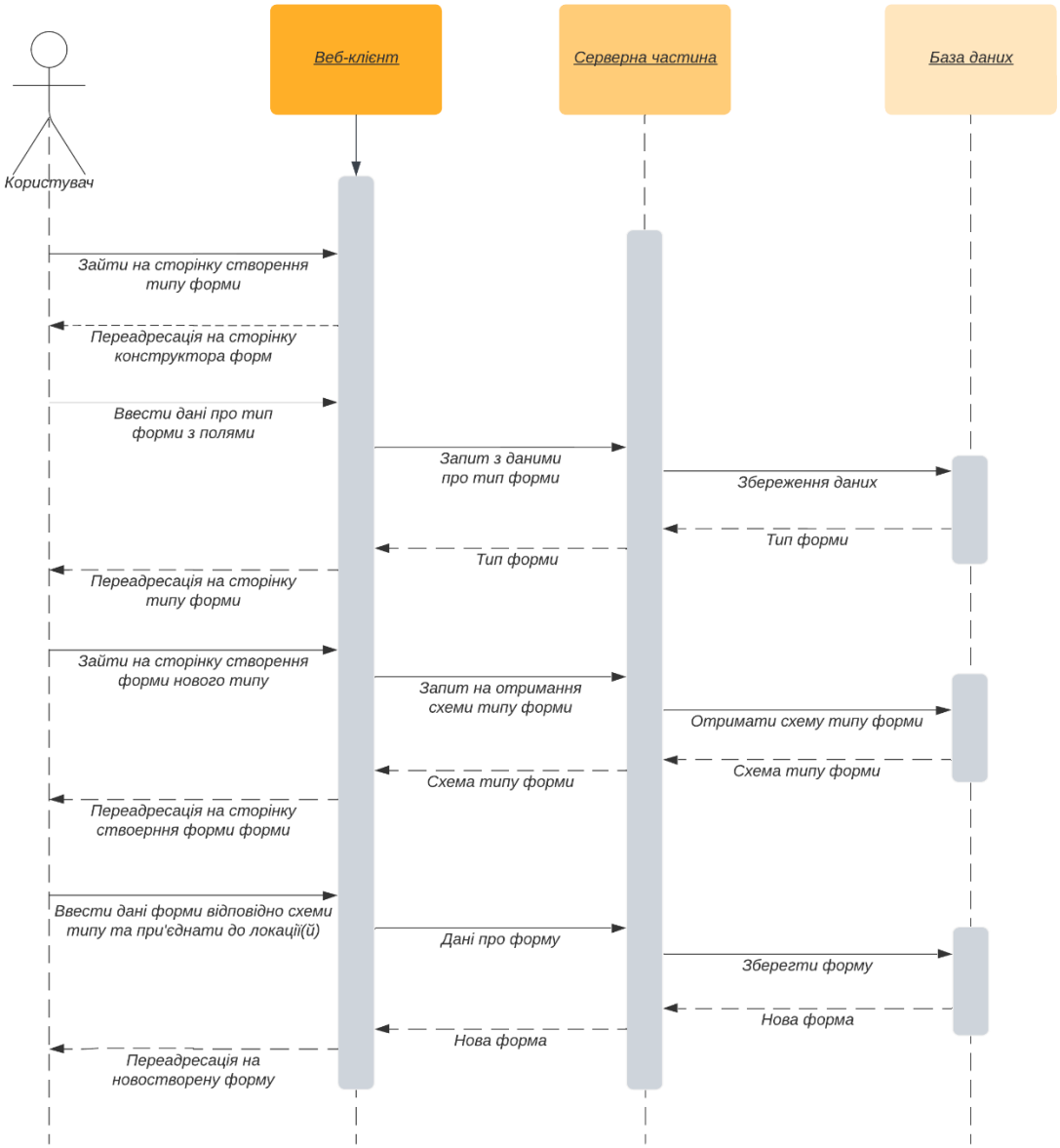

Рисунок Г.1 – Діаграма послідовності

ДОДАТОК Д

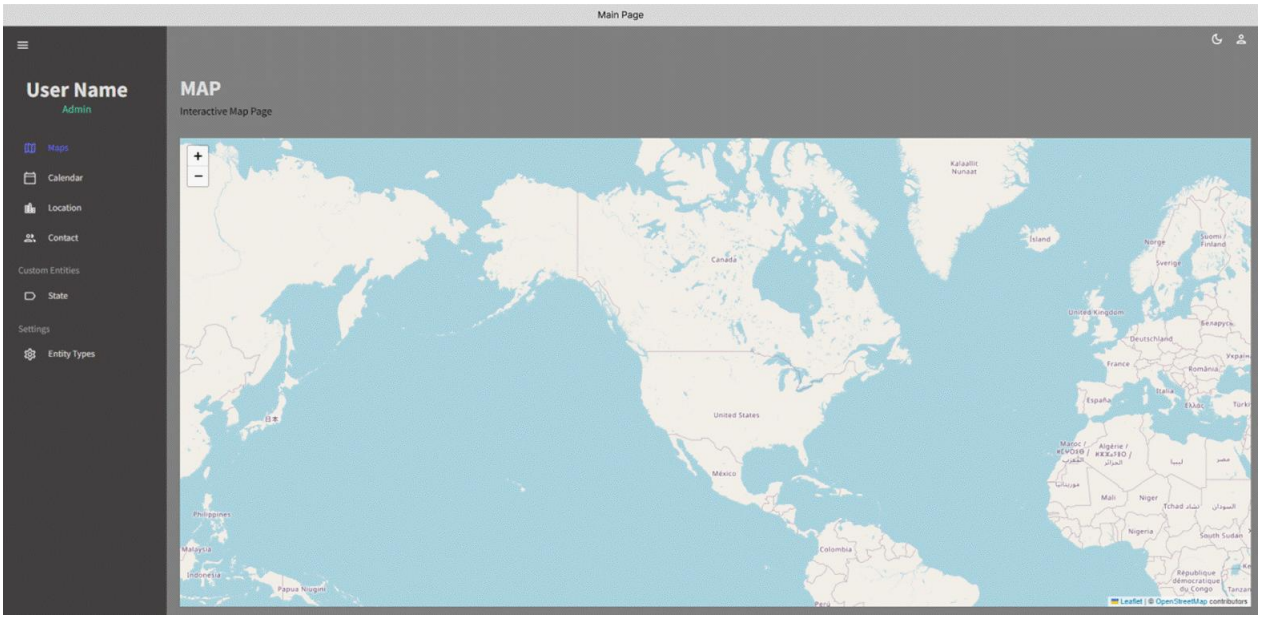

Рисунок Д.1 – Сторінка з головною мапою

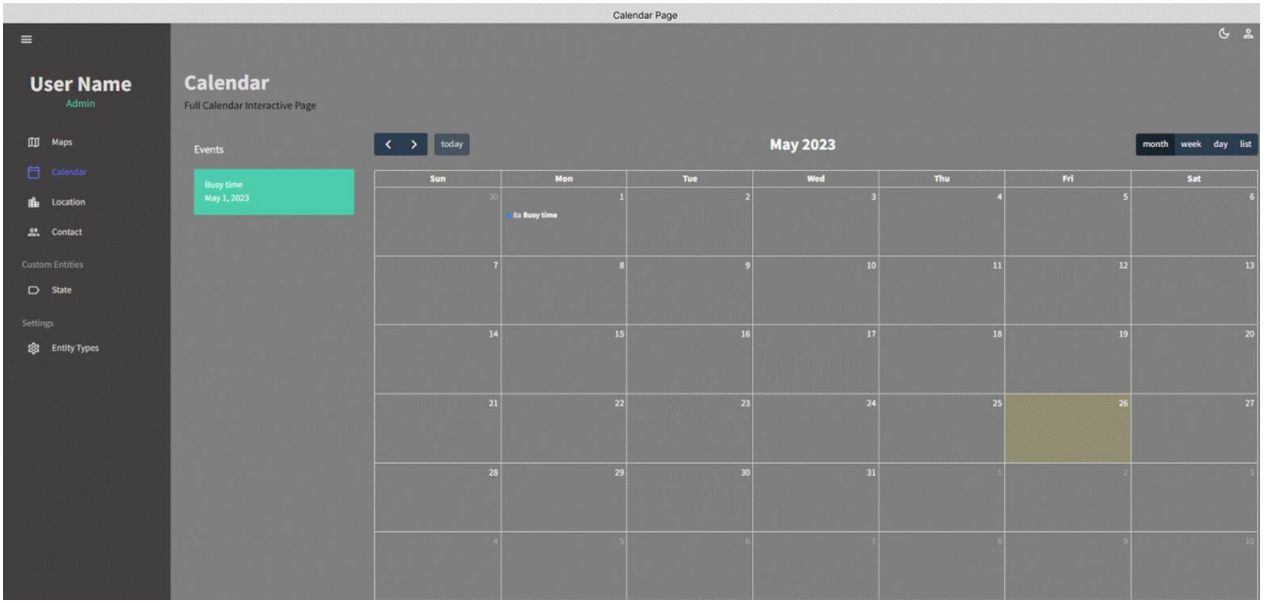

Рисунок Д.2 – Сторінка із календарем

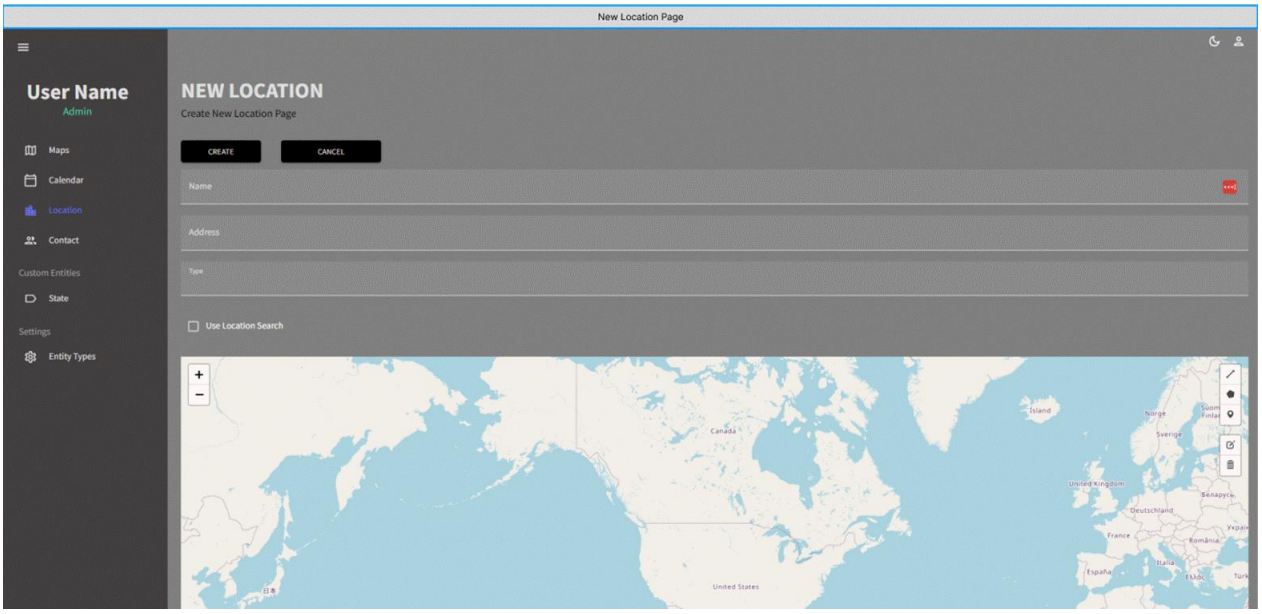

Рисунок Д.3 – Сторінка створення локації

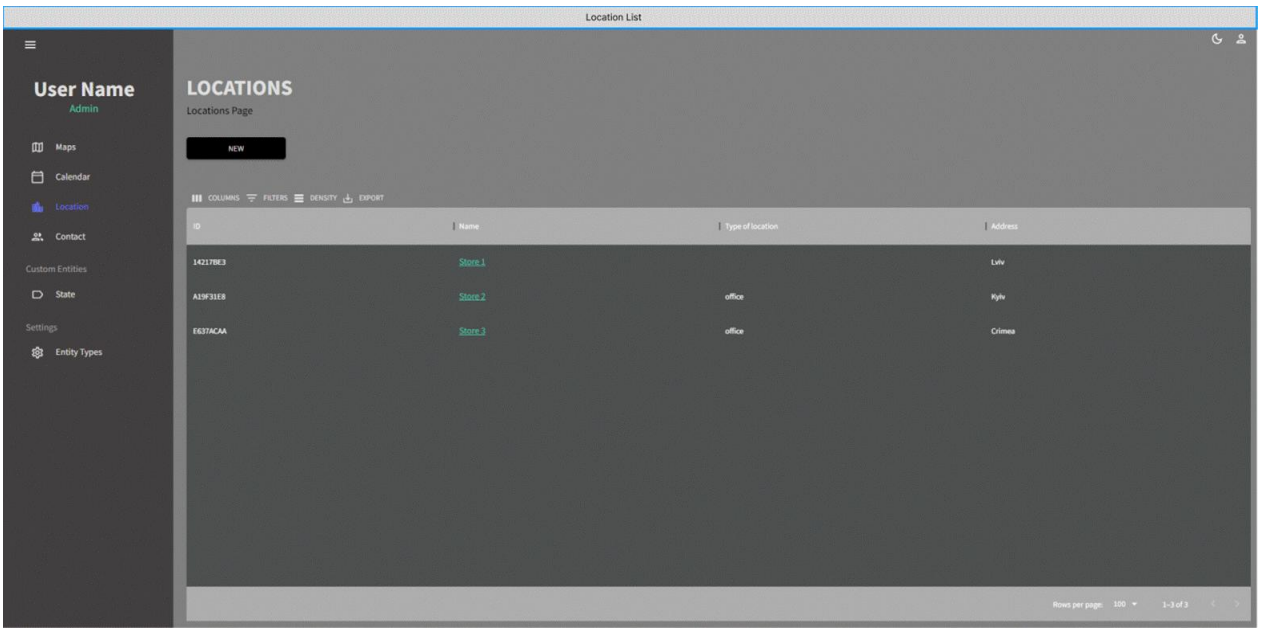

Рисунок Д.4 – Сторінка із списком локацій

Рисунок Д.5 – Вікно входу в систему

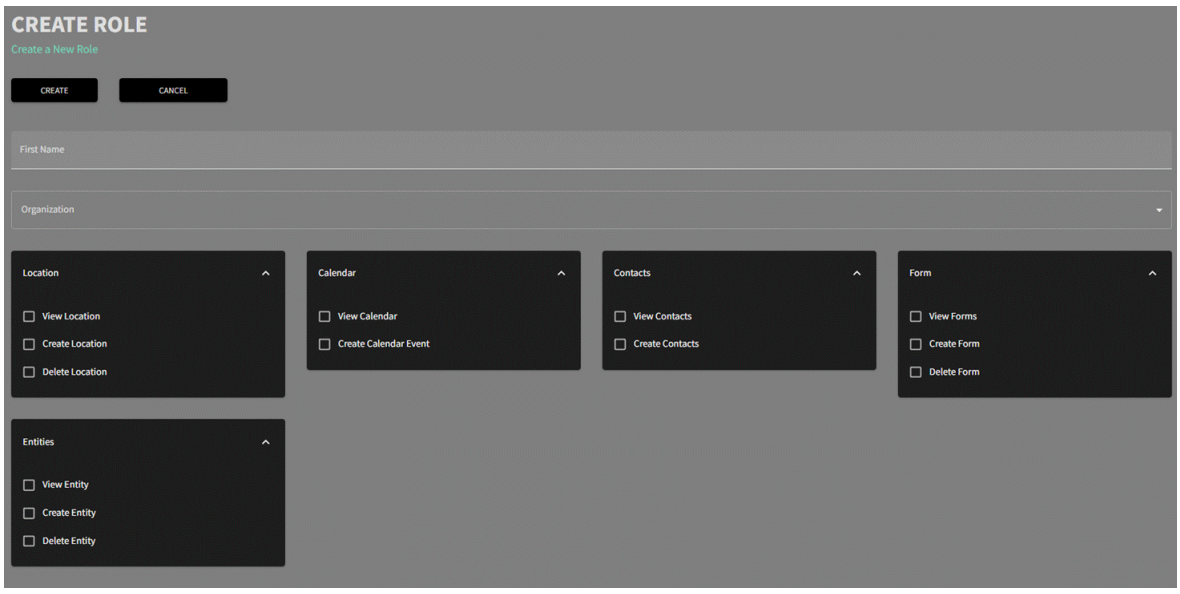

Рисунок Д.6 – Сторінка створення ролі

ДОДАТОК Е

Додаток Е.1

Файл location_controller.ex

```
defmodule
ManagementServerWeb.LocationController
do
  use ManagementServerWeb, :controller

  alias    ManagementServer.Locations,
as: LocationContext
  alias
ManagementServer.Locations.Location

  action_fallback
ManagementServerWeb.FallbackController

  @spec index(Plug.Conn.t(), map()) ::
Plug.Conn.t()
  def index(conn, _params) do
    locations =
      conn
      |>
LocationContext.list_locations()

    render(conn,         "index.json",
locations: locations)
  end

  @spec create(Plug.Conn.t(), map()) ::
Plug.Conn.t()
  def                create(%{assigns:
%{organization_id:  org_id}}  =  conn,
%{"location" => raw_location_attrs}) do
    enriched_attrs                    =
inject_organization_id(raw_location_at
trs, org_id)

    with    {:ok,    %Location{}    =
location_record}                    <-
LocationContext.create_location(enrich
ed_attrs) do
      conn
      |> put_status(:created)
      |> render("show.json", location:
location_record)
    end
  end

  @spec  show(Plug.Conn.t(),  map())  ::
Plug.Conn.t()
  def     show(conn,     %{"id"     =>
location_id}) do
    location_record                   =
LocationContext.get_location!(location
_id)

    render(conn,
"show_with_forms.json",      location:
location_record)
  end

  @spec update(Plug.Conn.t(), map()) ::
Plug.Conn.t()
  def    update(conn,    %{"id"    =>
location_id,       "location"       =>
update_attrs}) do
    location_record                   =
LocationContext.get_location!(location
_id)

    with    {:ok,    %Location{}    =
updated_location} <-

LocationContext.update_location(locati
on_record, update_attrs) do
      render(conn,         "show.json",
location: updated_location)
    end
  end

  @spec delete(Plug.Conn.t(), map()) ::
Plug.Conn.t()
  def    delete(conn,    %{"id"    =>
location_id}) do
    location_record                   =
LocationContext.get_location!(location
_id)

    with    {:ok,    %Location{}}    <-
LocationContext.delete_location(locati
on_record) do
      send_resp(conn, :no_content, "")
    end
  end

  # ----------------------------------
-----------------------------------
---
  # Private helpers
  # ----------------------------------
-----------------------------------
---

  defp
inject_organization_id(location_params
, org_id) do
    Map.put(location_params,
"organization_id", org_id)
  end
```

```
end
```

Файл сервісу location_service.ex

```
defmodule ManagementServer.Locations do
  @moduledoc """
  Context module responsible for operations on locations and related data.
  """

  import Ecto.Query, warn: false

  alias ManagementServer.Repo

  alias ManagementServer.{
    Locations.Location,
    LocationsForms.LocationForm,
    Forms.Form,
    FormTypes.FormType
  }

  @doc """
  Returns all locations available for the organization associated with the given connection.
  """
  def list_locations(%{assigns: %{account: %{user: %{organization_id: org_id}}}}) do
    Location
    |> where([loc], loc.organization_id == ^org_id)
    |> Repo.all()
  end

  @doc """
  Fetches a single location by ID and augments it with grouped form information.

  Raises if the location does not exist.
  """
  def get_location!(location_id) do
    base_location =
      Location
      |> where([loc], loc.id == ^location_id)
      |> Repo.one!()

    grouped_forms = load_grouped_forms_for_location(location_id)

    Map.put(base_location, :forms, grouped_forms)
  end

  @doc """
  Creates a new location.
  """
  def create_location(attrs \\ %{}) do
    %Location{}
    |> Location.changeset(attrs)
    |> Repo.insert()
  end

  @doc """
  Updates an existing location.
  """
  def update_location(%Location{} = location_struct, attrs) do
    location_struct
    |> Location.changeset(attrs)
    |> Repo.update()
  end

  @doc """
  Deletes a location.
  """
  def delete_location(%Location{} = location_struct) do
    Repo.delete(location_struct)
  end

  @doc """
  Returns an `%Ecto.Changeset{}` for tracking location changes.
  """
  def change_location(%Location{} = location_struct, attrs \\ %{}) do
    Location.changeset(location_struct, attrs)
  end

  # ----------------------------------------------------------------------
  # Private helpers
  # ----------------------------------------------------------------------

  defp load_grouped_forms_for_location(location_id) do
    LocationForm
    |> join(:inner, [lf], f in assoc(lf, :form))
    |> join(:inner, [lf, f], ft in FormType, on: ft.id == f.form_type_id)
    |> where([lf, _f, _ft], lf.location_id == ^location_id)
```

```
    |> select([_lf, f, ft], %{
      id: f.id,
      name: f.name,
      form_type_id: ft.id,
      form_type_name: ft.name,
      form_type_label: ft.label
    })
    |> Repo.all()
    |> Enum.group_by(&
&1.form_type_label)
  end
end
```

Файл схеми location_entity.ex

```
defmodule
ManagementServer.Locations.Location do
  use Ecto.Schema
  import Ecto.Changeset

  @primary_key    {:id,    :binary_id,
autogenerate: true}
  @foreign_key_type :binary_id
  @timestamps_opts              [type:
:utc_datetime_usec]

  @location_fields   ~w(name   location
address type organization_id id)a
  @required_fields   ~w(name   location
address)a

  schema "locations" do
    field :name, :string
    field :address, :string
    field :location, :map
    field :type, :string

    belongs_to(
      :organization,

ManagementServer.Organizations.Organiz
ation,
      foreign_key: :organization_id,
      type: :integer
    )

    has_many           :location_form,
ManagementServer.LocationsForms.Locati
onForm
    has_many      :forms,      through:
[:location_form, :form]

    timestamps()
  end

  @type t :: %__MODULE__{
          id: Ecto.UUID.t() | nil,
          name: String.t() | nil,
          address: String.t() | nil,
          location: map() | nil,
          type: String.t() | nil,
          organization_id: integer() |
nil
        }

  @doc false
  @spec changeset(t() | %__MODULE__{},
map()) :: Ecto.Changeset.t()
  def     changeset(%__MODULE__{}     =
location_struct,      params)      when
is_map(params) do
    location_struct
    |> cast(params, @location_fields)
    |>
validate_required(@required_fields)
  end
end
```

Файл схеми location_page.ex

```
defmodule
ManagementServerWeb.LocationView do
  use ManagementServerWeb, :view

  alias __MODULE__
  alias ManagementServerWeb.FormView

  # --------- Public render callbacks -
----------------------------------
--

  def render("index.json", assigns) do
    locations    =    Map.get(assigns,
:locations, [])

    %{
      data:     render_many(locations,
LocationView, "location.json")
    }
  end

  def render("show.json", assigns) do
    location   =   Map.fetch!(assigns,
:location)

    %{
      data:       render_one(location,
LocationView, "location.json")
    }
  end

  def   render("show_with_forms.json",
assigns) do
```

```
    location    =    Map.fetch!(assigns,
:location)

    %{
      data:        render_one(location,
LocationView, "location_forms.json")
    }
  end

  def           render("location.json",
%{location: location_struct}) do

base_location_payload(location_struct)
  end

  def     render("location_forms.json",
%{location: location_struct}) do
    location_struct
    |> base_location_payload()
    |>                   Map.put(:forms,
location_struct.forms)
  end

  # --------- Private helpers --------
-------------------------------------
---
```

Файл router.ex

```
defmodule ManagementServerWeb.Router
do
  use ManagementServerWeb, :router
  use Plug.ErrorHandler

  alias PhoenixSwagger.Plug.SwaggerUI
  alias
ManagementServerWeb.Auth.{Pipeline,
SetAccount}

  ## -------- Error handling ---------
-------------------------------------
---

  # Required by Plug.ErrorHandler
  defp handle_errors(conn, %{reason:
%Phoenix.Router.NoRouteError{message:
msg}}) do
    send_json_error(conn, %{errors:
msg})
  end

  defp handle_errors(conn, %{reason:
%{message: msg}}) do
    send_json_error(conn, %{error:
msg})
  end
```

```
  defp
base_location_payload(location_struct)
do
    %{
      id: location_struct.id,
      organization_id:
location_struct.organization_id,
      name: location_struct.name,
      location:
location_struct.location,
      type: location_struct.type,
      updated_at:
location_struct.updated_at,
      address: location_struct.address
    }
  end

  # kept as a utility helper, but
renamed and made clearer
  defp
extract_form_from_location(location_fo
rm) do
    Map.get(location_form, :form)
  end
end
```

```
  defp handle_errors(conn, _info) do
    # Generic fallback (optional, but
safer)
    send_json_error(conn, %{error:
"Internal server error"})
  end

  defp send_json_error(conn, payload)
do
    conn
    |> json(payload)
    |> halt()
  end

  ## -------- Pipelines --------------
-------------------------------------
---

  pipeline :api_stack do
    plug :accepts, ["json"]
    plug :fetch_session
    # plug CORSPlug, origins: ["*"],
allow_headers: ["*"]
  end

  pipeline :protected_api do
    plug Pipeline
    plug SetAccount
```

```
  end

  ## -------- Swagger UI -------------
-------------------------------------
---

  scope "/api/swagger" do
    forward "/", SwaggerUI,
      otp_app: :management_server,
      swagger_file: "swagger.json"
  end

  ## -------- Public API -------------
-------------------------------------
---

  scope "/api", ManagementServerWeb do
    pipe_through :api_stack

    get "/", DefaultController, :index

    post "/accounts/create",
AccountController, :create
    post "/accounts/sign_in",
AccountController, :sign_in

    post "/roles/create",
RoleController, :create

    post "/organizations/create",
OrganizationController, :create
  end

  ## -------- Authenticated API ------
-------------------------------------
--

  scope "/api", ManagementServerWeb do
    pipe_through [:api_stack,
:protected_api]

    # Accounts
    get  "/accounts",
AccountController, :index
    get  "/accounts/current",
AccountController, :current_account
    get  "/accounts/sign_out",
AccountController, :sign_out
    get  "/accounts/refresh_session",
AccountController, :refresh_session
    post "/accounts/update",
AccountController, :update

    # Users
    get  "/users",
UserController, :index
    put  "/users/update",
UserController, :update
    post "/users/create",
UserController, :create

    # Roles
    get "/roles", RoleController,
:index

    # Organizations
    get "/organizations",
OrganizationController, :index
    get "/organizations_roles",
OrganizationController,
:organizations_with_roles

    # Locations
    resources "/locations",
LocationController, except: [:new,
:edit]

    # Forms
    resources "/forms",
FormController, except: [:new, :edit]

    # Workers
    resources "/workers",
WorkerController, except: [:new,
:edit]

    # Events
    resources "/events",
EventController, except: [:new, :edit]

    # Form types
    resources "/form_types",
FormTypeController, except: [:new,
:edit]
  end
end
```

Додаток Е.2

Файл LocationDetailsScreen/index.tsx

```
import React, {
  useCallback,
  useEffect,
  useMemo,
  useRef,
  useState,
  SyntheticEvent,
} from "react";
import {
  Box,
  Typography,
  Tabs,
  Tab,
  useTheme,
} from "@mui/material";
import {
  DataGrid,
  GridColDef,
  GridToolbar,
} from "@mui/x-data-grid";
import {
  MapContainer,
  TileLayer,
  FeatureGroup,
  GeoJSON,
  useMap,
} from "react-leaflet";
import { Link, useNavigate, useParams } from "react-router-dom";
import { renderToStaticMarkup } from "react-dom/server";
import * as L from "leaflet";
import { useSnackbar } from "notistack";

import Header from "src/shared/ui/components/Header";
import useApi from "src/shared/agent";
import Button from "src/shared/ui/Button";
import { utilities } from "src/shared";
import { tokens } from "src/shared/global/theme";

interface TabPanelProps {
  children?: React.ReactNode;
  tabIndex: number;
  currentIndex: number;
}

const TabPanel: React.FC<TabPanelProps> = ({
  children,
  tabIndex,
  currentIndex,
  ...rest
}) => {
  const isActive = currentIndex === tabIndex;

  return (
    <div
      role="tabpanel"
      hidden={!isActive}
      id={`location-tabpanel-${tabIndex}`}
      aria-labelledby={`location-tab-${tabIndex}`}
      {...rest}
    >
      {isActive && (
        <Box sx={{ p: 3 }}>
          <Typography component="div">{children}</Typography>
        </Box>
      )}
```

```
          </div>
        );
      };

      const buildTabA11yProps = (index: number) => ({
        id: `location-tab-${index}`,
        "aria-controls": `location-tabpanel-${index}`,
      });

      const AutoZoomToFeature: React.FC<{ featureGeometry: any }> = ({
        featureGeometry,
      }) => {
        const leafletMap = useMap();

        useEffect(() => {
          if (!featureGeometry) return;

          const geoLayer = L.geoJson(featureGeometry);
          const bounds = geoLayer.getBounds();
          const center = bounds.getCenter();

          leafletMap.flyTo([center.lat, center.lng], 14, {
            animate: true,
            duration: 2.5,
          });
        }, [featureGeometry, leafletMap]);

        return null;
      };

      const LocationDetailsScreen: React.FC = () => {
        const theme = useTheme();
        const palette = tokens(theme.palette.mode);
        const { enqueueSnackbar } = useSnackbar();
        const { locationId } = useParams();
        const { get, del } = useApi();
        const navigate = useNavigate();

        const [locationData, setLocationData] = useState<LocationType | null>(null);
        const [activeTabIndex, setActiveTabIndex] = useState(0);

        const featureLayerRef = useRef<FeatureGroup | null>(null);

        const dataGridColumns: GridColDef[] = useMemo(
          () => [
            {
              field: "id",
              headerName: "ID",
              flex: 1,
              renderCell: (params) => (
                <div>{utilities.generateIdSlug(params.row?.id)}</div>
              ),
            },
            {
              field: "name",
              headerName: "Name",
              flex: 1,
              renderCell: (params) => (
                <Typography
                  color={palette.greenAccent[500]}
                  component={Link}
                  to={`${params.row?.id}`}
```

```
            >
              {params.row?.name}
            </Typography>
          ),
        },
      ],
      [palette.greenAccent]
    );

    const handleTabChange = (_: SyntheticEvent, newIndex: number) => {
      setActiveTabIndex(newIndex);
    };

    const loadLocation = useCallback(async () => {
      try {
        const response = await get(`/locations/${locationId}`);

        setLocationData(response.data?.data);
      } catch (error) {
        // eslint-disable-next-line no-console
        console.error("Failed to load location", error);
      }
    }, [get, locationId]);

    useEffect(() => {
      void loadLocation();
    }, [loadLocation]);

    const attachPopupToFeature = (feature: any, layer: any) => {
      layer.on({
        click: () => {
          const popupInstance = L.popup();

          const popupContent = (
            <>
              <div
className="disabled">
                <a
href={`/locations/${feature?.properties?.id}`}>

{feature?.properties?.name}
                </a>
              </div>
              <div  className="d-flex">
                <i>Type: {feature?.properties?.type}</i>
              </div>
            </>
          );

          const container = document.createElement("div");
          container.innerHTML = renderToStaticMarkup(popupContent);

popupInstance.setContent(container);

          layer

.bindPopup(popupInstance, {
              direction: "center",
              permanent: true,
              className: "labelstyle",
            })
            .openPopup();
        },
      });
    };

    const deleteLocation = useCallback(async () => {
      if (!locationId) return;
```

```
          try {
            await
del(`/locations/${locationId}`);

enqueueSnackbar(`${locationData?.name}
is deleted.`, {
              variant: "info",
              anchorOrigin: {
                horizontal: "right",
                vertical: "top",
              },
            });
            navigate("/locations");
          } catch (error) {
            //  eslint-disable-next-line
no-console
            console.error("Failed    to
delete location", error);
          }
        },    [del,    enqueueSnackbar,
locationData?.name,          locationId,
navigate]);

        const formGroups = useMemo(
          ()                          =>
Object.keys(locationData?.forms ?? {}),
          [locationData?.forms]
        );

        return (
          <Box m="20px">
            <Header    title="LOCATION"
subtitle="View Location Page" />

            <Typography    variant="h3"
component="h2" marginBottom="15px">
              {locationData?.name}    -
{utilities.generateIdSlug(locationData?
.id)}
            </Typography>

            <Box
              marginBottom="10px"
              display="flex"
              justifyContent="space-
between"
              alignItems="center"
              width="300px"
            >
              <Box mr="50px">
                <Button    label="Back"
url="/locations" />
              </Box>

              <Button
                styles={{
                  backgroundColor:
palette.redAccent[500],
                  marginRight: "10px",
                }}

onClick={deleteLocation}
              >
                Delete
              </Button>

              <Button
                styles={{
backgroundColor:
palette.blueAccent[500] }}
                onClick={()          =>
navigate(`/locations/edit/${locationId}
`)}
              >
                Edit
              </Button>
            </Box>

            <Typography    variant="h6"
component="h2" marginBottom="15px">
              Type:
{locationData?.type}
            </Typography>
```

```
            <Typography     variant="h6"
component="h2" marginBottom="15px">
              Address:
{locationData?.address}
            </Typography>

            <Box sx={{ width: "100%" }}>
              <Box  sx={{  borderBottom:
1, borderColor: "divider" }}>
                <Tabs

value={activeTabIndex}

onChange={handleTabChange}
                  aria-label="location
forms tabs"
                  sx={{
                    "& .Mui-selected": {
                      color:    "#88a6ff
!important",
                    },
                  }}
                >

{formGroups.map((formName, index) => (
                    <Tab
                      key={formName}
                      label={formName}

{...buildTabA11yProps(index)}
                    />
                  ))}
                </Tabs>
              </Box>


{formGroups.map((formName, index) => (
                <TabPanel
                  key={formName}

currentIndex={activeTabIndex}
                  tabIndex={index}
                >
                  <Box
                    height="50vh"
                    sx={{
                      "&   .MuiDataGrid-
root": {
                        border: "none",
                      },
                      "&   .MuiDataGrid-
cell": {
                        borderBottom:
"none",
                      },
                      "&  .name-column--
cell": {
                        color:
palette.greenAccent[300],
                      },
                      "&   .MuiDataGrid-
columnHeaders": {

backgroundColor:
palette.redAccent[700],
                        borderBottom:
"none",
                      },
                      "&   .MuiDataGrid-
virtualScroller": {

backgroundColor: palette.primary[400],
                      },
                      "&   .MuiDataGrid-
footerContainer": {
                        borderTop:
"none",

backgroundColor:
palette.redAccent[700],
                      },
                      "&   .MuiCheckbox-
root": {
```

```
                        color:
`${palette.greenAccent[200]}
!important`,
                      },
                      "&   .MuiDataGrid-
toolbarContainer .MuiButton-text": {
                        color:
`${palette.grey[100]} !important`,
                      },
                    }}
                  >
                    <DataGrid
                      rows={locationData?.forms[formName]   ??
[]}
                      columns={dataGridColumns}
                      slots={{  toolbar:
GridToolbar }}
                    />
                  </Box>
                </TabPanel>
              ))}
            </Box>

            <Box height="70%">
              <MapContainer
                center={[45.2595092,   -
104.5204334]}
                zoom={3}
                style={{  height:  "65vh"
}}
              >
                <FeatureGroup
ref={featureLayerRef}>
                  <GeoJSON
key={locationData?.id}
                    onEachFeature={attachPopupToFeature}
                    data={locationData?.location?.features
?? null}
                  />

                  <TileLayer
                    attribution='&copy;
<a
href="https://www.openstreetmap.org/cop
yright">OpenStreetMap</a> contributors'
                    url="https://{s}.tile.openstreetmap.org
/{z}/{x}/{y}.png"
                  />


                  {!!locationData?.location?.features?.[0
] && (
                    <AutoZoomToFeature
                      featureGeometry={locationData.location.
features[0]}
                    />
                  )}
                </FeatureGroup>
              </MapContainer>
            </Box>
          </Box>
        );
      };

      export                        default
LocationDetailsScreen;
```