

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАДИМА ГЕТЬМАНА**

**Навчально-науковий інститут
«Інститут інформаційних технологій в економіці»**

Кафедра інформаційних систем в економіці

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА
«Інформаційні управляючі системи і технології»

галузь знань	12 Інформаційні технології
спеціальність	122 Комп'ютерні науки

Форма навчання: очна (денна)

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему: «Розроблення інформаційної системи для обліку та управління оборонними ресурсами»

здобувача Желноватенка Євгенія Анатолійовича

Науковий керівник: д.е.н., професор Устенко С.В.

(*nidnuc*)

Робота допущена до захисту перед екзаменаційною комісією з атестації здобувачів вищої освіти (ЕК)

Завідувач кафедри: к.е.н., доцент Тішков Б.О. _____
(підпис)

(*nidnuc*)

Київ 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАДИМА ГЕТЬМАНА
Навчально-науковий інститут «Інститут інформаційних технологій в економіці»
Кафедра інформаційних систем в економіці

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА
«Інформаційні управляючі системи і технології»

галузь знань 12 Інформаційні технології
спеціальність 122 Комп'ютерні науки

ПОГОДЖЕНО:
Керівник проєктної групи (гарант)
освітньо-професійної програми

_____ Устенко С.В.
« ____ » _____ 202_ р.

ЗАТВЕРДЖУЮ:
Завідувач кафедри інформаційних
систем в економіці

_____ Тішков Б.О.
« ____ » _____ 202_ р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

здобувача вищої освіти Желноватенка Євгенія Анатолійовича

_____ денної _____ форми навчання

на підготовку кваліфікаційної магістерської роботи
на тему: «Розроблення інформаційної системи для обліку та управління оборонними ресурсами»

Тему затверджено наказом ректора Університету від «07» березня 2025 р. №464-ст

Кваліфікаційна магістерська робота виконується на матеріалах
відкритих публікацій та інтернет-джерел

План кваліфікаційної магістерської роботи

Розділ I ДОСЛІДЖЕННЯ ТА АНАЛІЗ ПІДХОДІВ ДО СТВОРЕННЯ ІНФОРМАЦІЙНИХ СИСТЕМ УПРАВЛІННЯ ОБОРОННИМИ РЕСУРСАМИ

Розділ II ХАРАКТЕРИСТИКА ІНФОРМАЦІЙНОЇ СИСТЕМИ УПРАВЛІННЯ ОБОРОННИМИ РЕСУРСАМИ ТА МЕТОДИ І МОДЕЛІ

Розділ III РОЗРОБЛЕННЯ ПРОЄКТНИХ РІШЕНЬ ТА ЇХ РЕАЛІЗАЦІЯ В ІНФОРМАЦІЙНІЙ СИСТЕМІ УПРАВЛІННЯ ОБОРОННИМИ РЕСУРСАМИ

Об'єкт дослідження: процеси обліку, моніторингу та управління матеріально-технічними ресурсами у сфері оборонного забезпечення

Предмет дослідження: методи, технології та архітектурні підходи до проєктування й реалізації інформаційної системи для автоматизації управління оборонними ресурсами

Мета кваліфікаційної магістерської роботи: розроблення масштабованої інформаційної системи для автоматизованого обліку та управління оборонними ресурсами з використанням сучасного стеку технологій

Конкретні завдання, які здобувач повинен виконати для досягнення поставленої мети:

У розділі I розглядається предметна область управління оборонними ресурсами, аналізуються особливості функціонування наявних інформаційних систем у цій сфері, виявляються основні проблеми, пов'язані з недостатньою автоматизацією, інтеграцією та відповідністю міжнародним стандартам. Визначаються вимоги до сучасної інформаційної системи управління оборонними ресурсами та обґрунтовується вибір архітектурних принципів, технологій і засобів реалізації, що відповідають умовам експлуатації, вимогам до безпеки та можливості масштабування.

У розділі II описується структура інформаційної системи, обґрунтовується побудова її архітектури на основі тривіневого підходу із поділом на рівні представлення, бізнес-логіки та доступу до даних. Представлено логіку взаємодії компонентів системи, описано використання шаблонів проєктування (Repository, Unit of Work, DTO, AutoMapper) та методів реалізації механізмів автентифікації й авторизації з використанням ASP.NET Core Identity і JWT. Наведено опис моделей для роботи з просторовими даними та інтерактивної візуалізації розміщення ресурсів за допомогою Leaflet. Функціональна структура доповнюється UML-діаграмами, які формалізують внутрішні процеси й взаємозв'язки.

У розділі III виконується проєктування бази даних з урахуванням вимог до структури, нормалізації, підтримки геолокації та забезпечення цілісності даних. Реалізується серверна частина системи на основі ASP.NET Core Web API з побудовою RESTful-ендпоінтів, а також клієнтська частина засобами Angular із застосуванням Angular Material, реактивних форм і маршрутизації. Наведено підходи до тестування функціоналу системи, перевірки безпеки доступу, продуктивності та відповідності заданим вимогам. Завершальний етап включає демонстрацію реалізованого інтерфейсу, визначення ресурсних вимог до технічного забезпечення та опис перспектив подальшого розвитку системи.

**Завдання підготував
науковий керівник**

(підпис)

(ініціали, прізвище)

« ____ » _____ 202_ р.

**Завдання одержав
здобувач**

(підпис)

(ініціали, прізвище)

« ____ » _____ 202_ р.

Реферат

Кваліфікаційна магістерська робота містить 66 сторінок, 3 таблиці, 62 рисунки, список використаних джерел з 51 найменування, додатки.

«Розроблення інформаційної системи для обліку та управління оборонними ресурсами»

Об'єктом дослідження кваліфікаційної магістерської роботи є процеси автоматизації обліку, моніторингу та управління оборонними ресурсами в інформаційних управляючих системах.

Предметом дослідження є сукупність теоретичних, методичних і прикладних підходів до проєктування та впровадження веборієнтованої інформаційної системи обліку та управління оборонними ресурсами з використанням сучасного технологічного стеку.

Мета і завдання дослідження. Основною метою кваліфікаційної магістерської роботи є підвищення ефективності функціонування логістичних процесів, прискорення обробки даних та забезпечення сумісності з оборонними стандартами НАТО шляхом розроблення масштабованої модульної ІУС на основі ASP.NET Core, Angular та MSSQL Server.

Відповідно до поставленої мети визначено такі завдання:

- дослідити предметну область, проаналізувати існуючі інформаційні системи обліку та управління оборонними ресурсами;
- обґрунтувати вибір підходів і технологій для створення інформаційної системи обліку та управління оборонними ресурсами;
- визначити структуру та надати характеристику інформаційної системи управління оборонними ресурсами;
- описати методи, моделі й архітектурні рішення, що були використані при розробленні системи;
- спроектувати базу даних для зберігання структурованої інформації про ресурси, підрозділи та локації;
- реалізувати програмне забезпечення з використанням ASP.NET Core, Angular та MSSQL Server;
- обґрунтувати та описати технічне забезпечення, необхідне для роботи системи;
- представити результати розроблення, тестування та апробації інформаційної системи в умовах моделювання її реального використання.

Теоретична, методична та практична значущість отриманих результатів.

У роботі узагальнено вимоги до ІУС військового призначення, обґрунтовано архітектурно-технологічні рішення, які забезпечують адаптивність, масштабованість і сумісність з сучасними безпековими системами. Практична цінність полягає у створенні функціональної ІУС, готової до впровадження у військових частинах та структурах територіальної оборони для підвищення ефективності управління оборонними ресурсами.

Рік виконання кваліфікаційної магістерської роботи – 2025.

Рік захисту роботи – 2025.

Ключові слова: інформаційна система, оборонні ресурси, облік, управління, військова логістика, ASP.NET Core, Angular, MSSQL Server.

Відгук

про кваліфікаційну магістерську роботу
здобувача навчально-наукового інституту «Інститут інформаційних технологій в економіці»

освітньо-професійної програми «Інформаційні управляючі системи і технології»

Желноватенка Євгенія Анатолійовича

на тему «Розроблення інформаційної системи для обліку та управління
оборонними ресурсами»

1. Актуальність теми: дослідження визначається високим професійним рівнем реалізації повноцінного комплексу заходів, спрямованих на проектування, розроблення, тестування та впровадження інформаційної системи, яка відповідає сучасним вимогам автоматизації обліку ресурсів оборонного призначення.
2. Позитивні риси кваліфікаційної магістерської роботи: Автор роботи провів аналіз предметної області управління оборонними ресурсами, чинних підходів до побудови інформаційних систем у військовій та оборонній галузях, а також вивченню досвіду стандартизації згідно з принципами НАТО (зокрема, стандартами STANAG); обґрунтовано вибір підходів та технологій для створення системи, зокрема — використання стеку **ASP.NET Core (C#), Angular, MSSQL Server**, які забезпечують гнучкість у розробці та можливість масштабування системи;
3. Наявність і цінність самостійних розробок автора В розробленій ІУС в частині програмного забезпечення реалізовано клієнтську частину з використанням **Angular** та бібліотеки Angular Material. У рамках реалізації здійснено повний цикл тестування системи та проведено аналіз ресурсних характеристик і сформовано перелік технічних вимог до серверної інфраструктури, рекомендовано середовище розгортання на базі IIS Express у середовищі Windows 10 з використанням SQL Server.
4. Наявність недоліків: Рекомендація – потрібно впровадити дану систему в оборонний сектор як на рівні окремих функціональних компонентів, так і в цілому системи.
5. Загальна оцінка кваліфікаційної магістерської роботи та її допущення до захисту перед ЕК: є завершеним проектом, що поєднує теоретичну новизну, практичну значущість і відповідає вимогам освітньо-професійної програми спеціальності F3 «Комп'ютерні науки». Отримані результати можуть бути використані як у реальному впровадженні в оборонних структурах, так і в навчальному процесі при підготовці фахівців з інформаційних управляючих систем і технологій і заслуговує на найвищу оцінку.

Науковий керівник _____ д.е.н., професор _____

_____ Устенко С.В.

«15» травня 2025 р.

Рецензія
на кваліфікаційну магістерську роботу
здобувача вищої освіти
Желноватенка Євгенія Анатолійовича

Тема «**Розроблення інформаційної системи для обліку та управління оборонними ресурсами**»

Актуальність теми кваліфікаційної магістерської роботи і доцільність її розроблення: Тема відповідає сучасним вимогам цифровізації оборонного сектору. Робота спрямована на підвищення ефективності обліку ресурсів та узгоджується з міжнародними стандартами, зокрема НАТО.

Якість проведеного дослідження: Проведено глибокий аналіз предметної області, обґрунтовано вибір технологій, реалізовано базу даних, серверну та клієнтську частини системи. Матеріал подано логічно та послідовно.

Позитивні риси кваліфікаційної магістерської роботи: Систему реалізовано з використанням сучасного стеку технологій. Відзначається чітка структура, наявність діаграм, прототипів, інтеграція з картою та реалізована багаторольова авторизація.

Зауваження: Деякі фрагменти тексту бажано вдосконалити стилістично. Можна деталізувати перспективи інтеграції з іншими системами.

Практична значимість висновків і рекомендацій: Результати можуть бути використані в оборонних структурах для автоматизації обліку ресурсів. Система має потенціал для подальшого розвитку та впровадження.

Місце роботи та посада рецензента

Науковий ступінь, учене звання (за наявності) Провідний інженер-програміст ТОВ «Уніті-Барс»

Підпис засвідчують: Піванова М.І.

Місце печатки організації, де працює рецензент



ЗМІСТ

ВСТУП.....	4
РОЗДІЛ 1 ДОСЛІДЖЕННЯ ТА АНАЛІЗ ПІДХОДІВ ДО СТВОРЕННЯ ІНФОРМАЦІЙНИХ СИСТЕМ УПРАВЛІННЯ ОБОРОННИМИ РЕСУРСАМИ	7
1.1 Аналіз існуючих інформаційних систем у сфері оборонних ресурсів	7
1.1.1 Дослідження сфери управління оборонними ресурсами	7
1.1.2 Дослідження інформаційних систем для обліку та управління оборонними ресурсами	8
1.2 Обґрунтування вибору підходів і технологій для розробки інформаційної системи обліку та управління оборонними ресурсами	14
РОЗДІЛ 2 ХАРАКТЕРИСТИКА ІНФОРМАЦІЙНОЇ СИСТЕМИ УПРАВЛІННЯ ОБОРОННИМИ РЕСУРСАМИ ТА МЕТОДИ І МОДЕЛІ.....	17
2.1 Структура і характеристика системи	17
2.2 Методи та моделі в інформаційних системах управління оборонними ресурсами	24
РОЗДІЛ 3 РОЗРОБЛЕННЯ ПРОЕКТНИХ РІШЕНЬ ТА ЇХ РЕАЛІЗАЦІЯ В інформаційній системі управління оборонними ресурсами.....	27
3.1 Проектування бази даних для інформаційної системи управління оборонними ресурсами	27
3.1.1 Інфологічна модель бази даних	27
3.1.2 Фізична модель бази даних	29
3.1.3 Оцінка обсягів даних та продуктивності	30
3.2 Проектування бази знань інформаційної управляючої системи та/або засоби інтелектуального аналізу даних.....	31
3.3. Програмне забезпечення.....	32
3.3.1 Проектування програмного забезпечення	32
3.3.2 Вибір архітектури програмного забезпечення	35
3.3.3 Інструментальні засоби розробки для створення прикладного програмного забезпечення	39
3.3.4 Тестування програмного забезпечення.....	42

3.3.5 Керівництво (інструкція) користувача.....	45
3.4. Технічне забезпечення.....	48
3.4.1 Обґрунтування вибору технічного забезпечення	48
3.4.2 Ресурсні вимоги до технічного забезпечення	51
3.5 Реалізація інформаційної системи управління оборонними ресурсами	53
ВИСНОВКИ.....	58
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	60
ДОДАТКИ.....	67

ВСТУП

Актуальність теми. Сучасні умови забезпечення національної безпеки України характеризуються посиленням викликів та загроз, які потребують комплексного та оперативного реагування. Це зумовлює необхідність значного підвищення ефективності використання та управління оборонними ресурсами, серед яких особливе місце займають матеріальні, технічні, людські та фінансові ресурси. Високий рівень управління цими ресурсами є визначальним фактором для забезпечення стійкої обороноздатності держави, ефективної логістики військових підрозділів і оперативного реагування на кризові ситуації.

Існуючі системи обліку оборонних ресурсів в Україні стикаються з проблемами недостатньої автоматизації, низького рівня інтеграції даних, повільної обробки інформації та недостатньої гнучкості щодо змінюваних вимог. Особливого значення набуває завдання інтеграції до стандартів НАТО, які передбачають широке застосування сучасних інформаційних технологій [1]. В умовах євроатлантичної інтеграції України надзвичайно важливо створити ефективну та сучасну інформаційну систему для обліку й управління оборонними ресурсами, яка відповідатиме міжнародним стандартам та вимогам.

Таким чином, розроблення новітньої інформаційної системи, що забезпечить автоматизацію процесів обліку та управління оборонними ресурсами, є актуальним завданням, реалізація якого сприятиме підвищенню ефективності логістичних процесів, оперативності прийняття управлінських рішень та зниженню загальних витрат, а також суттєвому посиленню обороноздатності держави.

Мета дослідження полягає у розробленні та впровадженні інформаційної системи для комплексної автоматизації процесів обліку, моніторингу та управління оборонними ресурсами з використанням новітніх програмних і технічних засобів.

Для досягнення поставленої мети визначені наступні завдання:

- Провести комплексний аналіз сучасного стану та виявити основні проблеми і недоліки існуючих систем управління оборонними ресурсами.

- Дослідити міжнародний досвід, зокрема країн-членів НАТО, та проаналізувати існуючі передові інформаційні системи у військовій логістиці.
- Визначити і науково обґрунтувати оптимальні технології та інструментальні засоби для створення ефективної інформаційної системи.
- Розробити концептуальну та логічну моделі даних, необхідних для комплексного і гнучкого управління оборонними ресурсами.
- Спроекувати і реалізувати базу даних та інтегрувати аналітичні модулі, що забезпечують підтримку прийняття управлінських рішень у реальному часі.
- Розробити програмне забезпечення інформаційної системи з використанням сучасного технологічного стеку C#, ASP.NET Core, RESTful API, Angular та MSSQL Server.
- Провести комплексне тестування розробленої системи, визначити її ефективність та здійснити аналіз результатів.
- Розробити рекомендації щодо подальшого впровадження та експлуатації інформаційної системи у військових структурах.

Об'єктом дослідження є процеси обліку та управління оборонними ресурсами України.

Предметом дослідження є методи, інформаційні системи та технології, що застосовуються для автоматизації обліку та управління ресурсами у сфері оборони.

Методи дослідження. У процесі дослідження використані методи системного аналізу, методи моделювання інформаційних процесів, методи об'єктно-орієнтованого аналізу та проектування, а також методи програмування, тестування програмного забезпечення і баз даних.

Теоретична та практична значущість отриманих результатів. Теоретична значущість дослідження полягає в узагальненні та поглибленні теоретичних підходів щодо проектування інформаційних систем військового призначення, удосконаленні методів управління оборонними ресурсами. Практична цінність полягає у створенні сучасної інформаційної системи, яка може

бути ефективно впроваджена у військових структурах України для покращення управлінської діяльності, оптимізації витрат і підвищення готовності військових формувань.

Інформаційна база дослідження включає чинні нормативно-правові акти у сфері національної оборони, матеріали наукових досліджень, технічні звіти, а також відкриті офіційні джерела Міністерства оборони України, НАТО та інших міжнародних організацій.

Структура магістерської роботи складається зі вступу, трьох основних розділів, висновків, списку використаних джерел та додатків. Загальний обсяг роботи становить 114 сторінок, враховуючи рисунки, таблиці, діаграми та додатки.

РОЗДІЛ 1

ДОСЛІДЖЕННЯ ТА АНАЛІЗ ПІДХОДІВ ДО СТВОРЕННЯ ІНФОРМАЦІЙНИХ СИСТЕМ УПРАВЛІННЯ ОБОРОННИМИ РЕСУРСАМИ

1.1 Аналіз існуючих інформаційних систем у сфері оборонних ресурсів

1.1.1 Дослідження сфери управління оборонними ресурсами

Облік оборонних ресурсів є однією з критично важливих складових у системі забезпечення національної безпеки будь-якої держави, зокрема України. Під предметною областю обліку оборонних ресурсів розуміється сукупність процесів, процедур, нормативно-правових актів та організаційних заходів, спрямованих на ефективне управління матеріальними, технічними, фінансовими та людськими ресурсами, необхідними для функціонування військових підрозділів та оборонних структур.

До ключових ресурсів, що підлягають обліку та управлінню в системі оборони, належать:

- Матеріально-технічні ресурси (озброєння, військова техніка, боєприпаси, спорядження, паливно-мастильні матеріали);
- Людські ресурси (особовий склад військових формувань, резервісти, мобілізаційні ресурси);
- Фінансові ресурси (бюджетні та позабюджетні кошти, призначені для фінансування оборонних програм);
- Інформаційні ресурси (бази даних, інформаційні системи та засоби зв'язку).

Облік цих ресурсів є складним процесом, що вимагає ретельного планування, контролю і координації дій численних структурних підрозділів та установ оборонного сектору. Важливим аспектом предметної області є документування та звітність, що дозволяють оперативно контролювати витрати та використання ресурсів, запобігати їх неефективному або нецільовому використанню.

Сучасна практика показує, що ефективне управління ресурсами можливе лише за умови впровадження комплексних інформаційних систем, що дозволяють автоматизувати процеси збору, обробки, зберігання та аналізу інформації. Особливої уваги потребує інтеграція з міжнародними стандартами НАТО, що визначають вимоги до сумісності інформаційних систем та форматів даних. Використання стандартів НАТО забезпечує прозорість управління ресурсами, значно спрощує процеси взаємодії з міжнародними партнерами, а також дозволяє оперативно обмінюватися інформацією в умовах колективних безпекових заходів [2].

Станом на сьогоднішній день в Україні існують певні виклики в організації обліку оборонних ресурсів, зокрема, недостатня інтеграція інформаційних потоків між різними структурними підрозділами оборонного сектору, обмежена автоматизація логістичних процесів та низький рівень стандартизації даних, що негативно впливає на оперативність прийняття управлінських рішень. Додатковим проблемним аспектом є недостатня узгодженість нормативної бази, яка регулює облік ресурсів, що ускладнює розробку і впровадження єдиної уніфікованої системи обліку. У зв'язку з цим виникає необхідність аналізу та перегляду чинного законодавства та нормативно-методичного забезпечення [3-14].

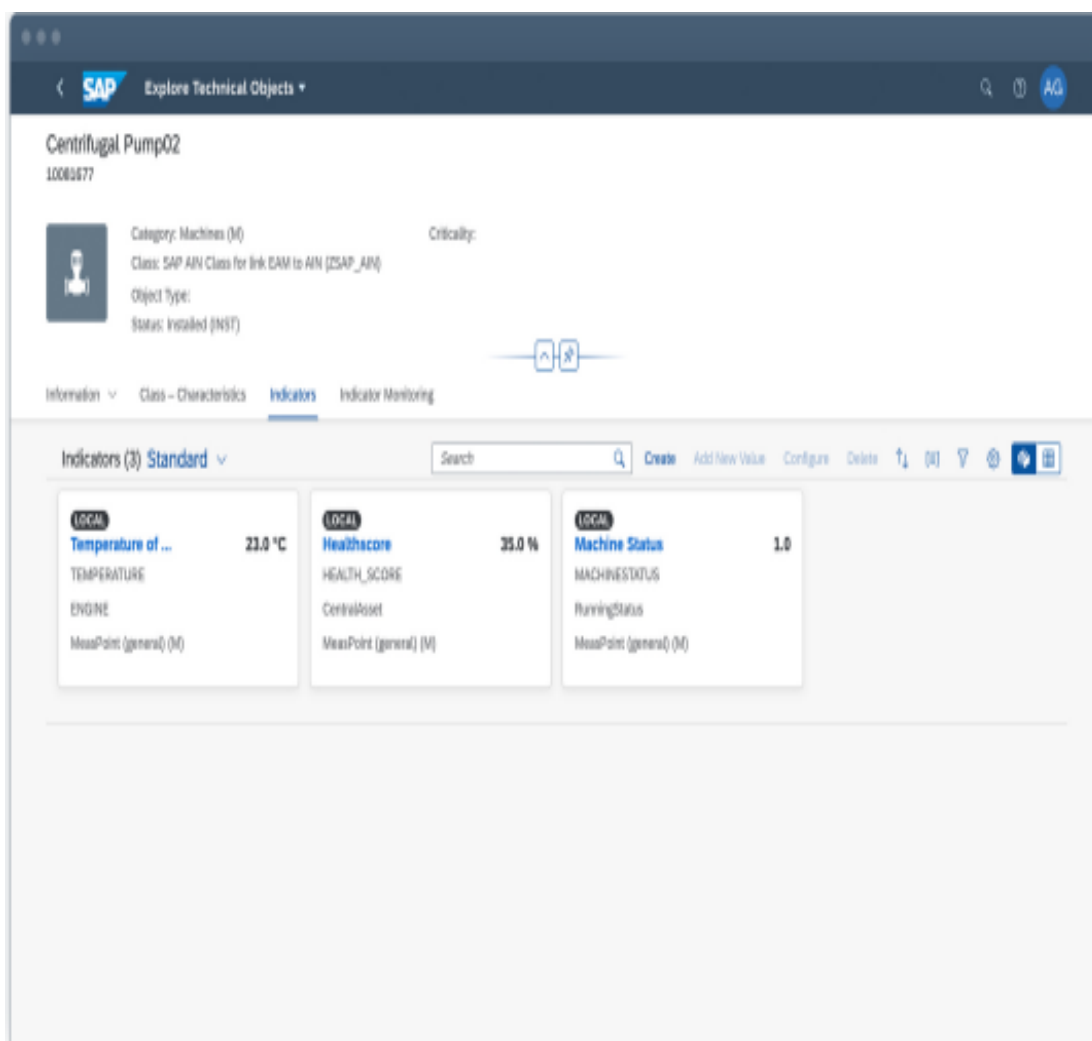
Таким чином, комплексний аналіз предметної області обліку оборонних ресурсів дозволяє виявити критичні точки та визначити напрями удосконалення інформаційної підтримки процесів управління ресурсами в оборонній сфері України, зокрема, шляхом підвищення інтегрованості, автоматизації процесів і стандартизації даних.

1.1.2 Дослідження інформаційних систем для обліку та управління оборонними ресурсами

У сучасній військовій логістиці інформаційні управляючі системи (ІУС) відіграють ключову роль у забезпеченні обліку, контролю та оптимізації використання матеріально-технічних ресурсів. З огляду на динамічні умови сучасної війни, підвищені вимоги до точності даних, швидкості реагування та

сумісності із системами партнерів по безпеці, актуальним є впровадження сучасних, гнучких і масштабованих ІУС в оборонному секторі.

В Україні одним із перших масштабних проєктів у цій сфері є впровадження SAP Defence & Security — автоматизованої системи управління оборонними ресурсами, яка дозволяє обліковувати залишки, вести обробку заявок, прогнозувати потреби та оптимізувати логістичні процеси. До системи вже підключено понад 200 військових частин, і її застосування дозволило зменшити час на обробку запитів з декількох тижнів до кількох днів. Система має модульну структуру та дозволяє адаптувати функціонал під потреби конкретних підрозділів, що підвищує її гнучкість у використанні [15].



a)

The screenshot displays the SAP Planning and Dispatching interface. The main window shows a map of Southern California with a route planned for Thursday, March 21, 2022. The route starts at a location near Forest Knolls and passes through several areas including Haight-Ashbury, Ashbury Heights, Corona Heights, The Castro, Dolores Heights, and Noe Valley, ending near Forest Hill. The route is color-coded (purple, green, red, blue). The left sidebar shows 'Route Details' with a list of stops including 'Regal Laptop', 'Check Printer', and 'Maintenance Hdl.'. The right sidebar shows a list of technicians with their names and contact information.

Рисунок 1.1 – Інтерфейс системи SAP Defence & Security.

SAP Defence & Security реалізована на базі стандартної корпоративної платформи SAP ERP, з інтегрованими модулями для управління ланцюгами постачання, обліку техніки, формування потреб, управління персоналом та фінансами. Важливим аспектом є підтримка процесів планування місій, автоматичного створення маршрутів постачання та прогнозування витрат ресурсів

на основі статистичних моделей. Система має вбудовані механізми контролю доступу та аудитів дій користувачів, що відповідає високим вимогам інформаційної безпеки. Крім того, вона здатна інтегруватися з геоінформаційними системами, зовнішніми складами, а також з IT-інфраструктурою НАТО через підтримку відповідних стандартів обміну даними (наприклад, STANAG 4329 та STANAG 4439). Таке технічне рішення не лише підвищує рівень оперативного управління ресурсами, а й формує технологічне підґрунтя для повноцінної сумісності з логістичними системами союзників по Альянсу [16-17].

Ще одним напрямом розвитку ІУС є цифровізація тилового та матеріального забезпечення. Міністерство оборони України розробляє низку систем, призначених для обліку продовольства, екіпірування, медичних засобів, запасів паливно-мастильних матеріалів тощо. Ці системи підтримують інтеграцію з внутрішніми складськими базами, дозволяють автоматично формувати звітність і передбачають використання електронного документообігу, що значно підвищує швидкість і точність операційних процесів.

Важливим є і впровадження принципів сумісності з інформаційними системами партнерів по безпеці. На цьому напрямі активно застосовуються підходи, які закладені у стандартах НАТО — зокрема, STANAG 4607, 4609 та 4586, що регламентують формат передачі даних, структуру повідомлень і протоколи взаємодії. Це створює передумови для спільного використання логістичних та ресурсних систем в умовах міжнародного співробітництва.

Окремо варто згадати і про розробку інтегрованих систем ситуаційної обізнаності (наприклад, система Delta, що зображена на рис. 1.2), які хоча й не належать безпосередньо до ІУС у класичному розумінні, однак мають тісний зв'язок із задачами управління ресурсами. Такі платформи забезпечують візуалізацію ресурсів на карті, синхронізацію даних із розвідки, складів, тилових служб та бойових підрозділів, що значно покращує точність і своєчасність управлінських рішень.

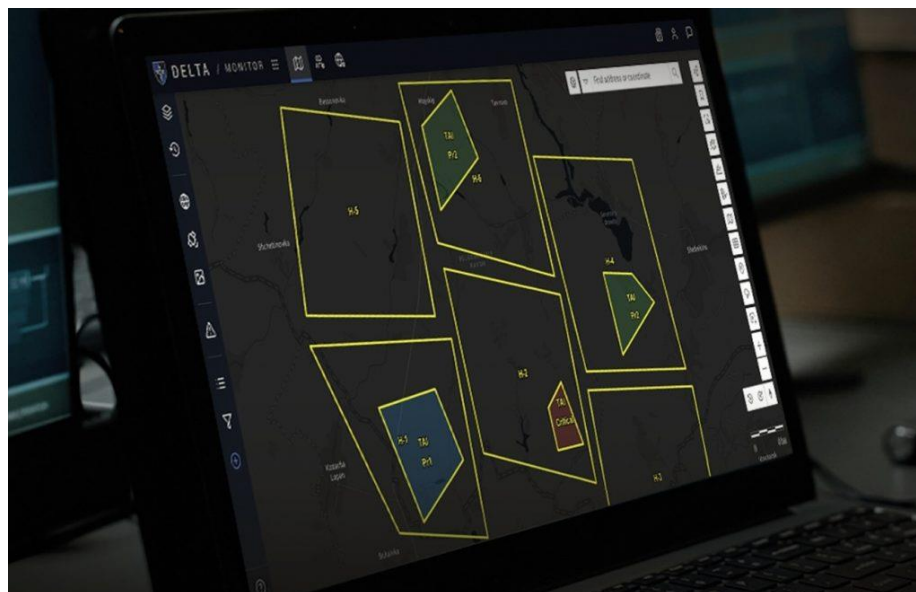


Рисунок 1.2 – Інтерфейс системи Delta

Джерело: <https://military.com/uk/news/systema-delta-vvedena-v-ekspluatatsiyu-dlya-usih-pidrozdiliv-oborony-ukrayiny/>

Система «Дельта» є сучасною цифровою хмарною платформою ситуаційної обізнаності, розробленою для інтеграції даних з розвідки, дронів, супутникових знімків, наземних сенсорів і відкритих джерел з метою формування єдиної оперативної картини бойових дій. Користувачі системи після верифікації отримують доступ до геоінформаційного інтерфейсу, який дозволяє додавати, редагувати та аналізувати об'єкти противника й власних підрозділів. Передбачено багаторівневу авторизацію, що забезпечує диференційований доступ відповідно до функціоналу та повноважень. «Дельта» синхронізується з іншими службовими інструментами, такими як чат-бот «Ворог», захищений месенджер Element, а також із системами ведення вогню, забезпечуючи повноцінну інтеграцію у єдину систему управління обороною. Ефективність платформи була підтверджена у бойових умовах — зокрема, під час контрнаступальних операцій ЗСУ у 2022–2023 роках, коли щодня ідентифікувалося понад півтори тисячі цілей. Станом на початок 2024 року система введена в експлуатацію для всіх Сил оборони України, що стало важливою віхою у впровадженні принципів мережево-центричної війни [18-19].

Проте аналіз ринку та поточного стану впровадження ІУС в Україні вказує на наявність низки проблем, які обмежують ефективність використання наявних рішень. До таких проблем належать:

- обмежена адаптованість закордонного програмного забезпечення до реалій національної нормативно-правової бази;
- фрагментарність систем, відсутність єдиного інформаційного простору;
- нестача спеціалізованого кадрового ресурсу для впровадження та супроводу ІУС;
- недостатня інтеграція з наявними базами даних, створеними на застарілих платформах;
- ризики, пов'язані з інформаційною безпекою та захистом критичної інфраструктури.
- У зв'язку з цим постає нагальна потреба у створенні нових інформаційних управляючих систем, які:
 - повністю відповідатимуть вимогам українського законодавства та оборонних стандартів;
 - підтримуватимуть швидке масштабування та гнучке налаштування;
 - інтегруватимуться з наявними базами даних та складськими системами;
 - забезпечуватимуть високий рівень інформаційної безпеки та стійкості до зовнішніх загроз;
 - матимуть зручний користувацький інтерфейс та мінімальну потребу в навчанні персоналу.

Таким чином, узагальнення наявного досвіду впровадження ІУС в Україні дозволяє окреслити вимоги до нової інформаційної системи, що розробляється в межах цієї роботи. Це дає можливість уникнути повторення помилок, врахувати кращі практики та реалізувати рішення, що реально покращить ефективність управління оборонними ресурсами.

1.2 Обґрунтування вибору підходів і технологій для розробки інформаційної системи обліку та управління оборонними ресурсами

Після всебічного аналізу предметної області, а також вивчення сучасних інформаційних систем, які застосовуються в галузі оборонної логістики, постала необхідність не лише описати, а й чітко обґрунтувати вибір архітектурних підходів, програмного стеку та інструментальних засобів, що були використані при створенні власної інформаційної системи. Усі рішення ухвалювались із врахуванням ключових принципів розробки критичних інформаційних систем: надійності, безпеки, масштабованості, сумісності, а також відповідності вимогам законодавства та потенційним стандартам НАТО. При цьому враховувались як технічні, так і організаційні обмеження функціонування оборонного сектору України, зокрема потреба у швидкому розгортанні системи, обмеженість ресурсів та важливість збереження гнучкості майбутнього масштабування.

Основоположним підходом при проєктуванні системи стала концепція модульності. Це означає, що кожна функціональна підсистема — облік матеріально-технічних засобів, керування кадрами, логістика, звітність, аналітика — була реалізована як самостійний компонент, що взаємодіє з іншими через чітко визначені інтерфейси. Модульність сприяє кращому розділенню відповідальності, пришвидшує тестування та дозволяє розгортати систему поетапно. Такий підхід також полегшує подальше обслуговування, оновлення й масштабування системи у випадку розширення функціоналу чи змін у нормативно-правовій базі.

Для забезпечення ефективної взаємодії між клієнтською та серверною частиною було застосовано архітектуру «клієнт-сервер» із RESTful API. Цей вибір був продиктований вимогами до масштабованості, сумісності з іншими системами, а також підтримки розподіленої інфраструктури. REST API у форматі JSON дозволяє створити легкий, інтероперабельний та зрозумілий канал обміну даними між компонентами системи, а також забезпечити майбутню інтеграцію з іншими цифровими платформами Збройних Сил України.

На серверному боці реалізація була виконана з використанням C# у середовищі ASP.NET Core. Цей вибір обумовлено високою продуктивністю фреймворку, його широкою підтримкою з боку спільноти, вбудованими механізмами безпеки, а також глибокою інтеграцією з іншими рішеннями Microsoft, що є особливо цінним у контексті використання SQL Server. ASP.NET Core підтримує масштабовану, багаторівневу архітектуру, що дозволяє ефективно управляти залежностями, застосовувати шаблони проєктування (наприклад, Dependency Injection) та реалізовувати бізнес-логіку відповідно до принципів SOLID.

СУБД Microsoft SQL Server була обрана як надійне рішення для зберігання структурованих даних. Крім підтримки транзакцій та індексації, вона забезпечує розширені механізми доступу та безпеки, що є критичними для оборонних застосунків. У межах реалізації було застосовано тип даних GEOGRAPHY для зберігання координат розміщення об'єктів, що дало змогу пов'язувати ресурси з конкретними локаціями та в подальшому реалізовувати географічну візуалізацію.

Клієнтська частина була реалізована з використанням Angular — сучасного фронтенд-фреймворку, що підтримує декларативний підхід до UI, компонентну архітектуру, двостороннє зв'язування даних та типізацію через TypeScript. Angular було обрано за його здатність забезпечити продуктивну, підтримувану та масштабовану клієнтську архітектуру. Додатково, для побудови UI-елементів застосовано Angular Material — бібліотеку, що надала змогу швидко реалізувати інтерфейси з високим рівнем зручності для користувача, відповідно до принципів Material Design. Це стало особливо важливим у контексті формування списків, таблиць ресурсів, фільтрів, вкладок тощо.

З метою реалізації контролю доступу до різних функціональних частин системи впроваджено механізми аутентифікації та авторизації. Основою цього блоку стала бібліотека ASP.NET Core Identity, яка забезпечила розмежування прав доступу на основі ролей: адміністратора, менеджера ресурсів, логіста тощо. Для захисту API від несанкціонованого доступу та підтримки безпечної передачі даних

між клієнтом і сервером реалізовано JWT-аутентифікацію (JSON Web Token), що дозволяє децентралізовано перевіряти права користувачів.

Розробка велась із використанням системи контролю версій Git, а хостинг репозиторіїв було здійснено на платформі GitHub. Це дало можливість забезпечити збереження історії змін, паралельну роботу над окремими гілками функціоналу, перегляд Pull Requests і централізоване управління розробкою. Хоча повноцінна реалізація CI/CD не впроваджувалась у рамках проєкту, підхід до керування версіями дозволив ефективно контролювати якість розробки та швидко усувати помилки.

Підсумовуючи вищевикладене, можна зазначити, що обрані підходи та засоби реалізації виявилися адекватними викликам і вимогам, які постають перед сучасною оборонною ІТ-системою. Комбінація модульності, багаторівневої архітектури, сучасного стеку технологій, високого рівня безпеки, зручного інтерфейсу та орієнтації на подальший розвиток забезпечила надійне підґрунтя для створення інформаційної системи, яка відповідає як поточним, так і перспективним потребам оборонного управління.

РОЗДІЛ 2

ХАРАКТЕРИСТИКА ІНФОРМАЦІЙНОЇ СИСТЕМИ УПРАВЛІННЯ ОБОРОННИМИ РЕСУРСАМИ ТА МЕТОДИ І МОДЕЛІ

2.1 Структура і характеристика системи

Інформаційна система для обліку та управління оборонними ресурсами розроблена з дотриманням сучасних архітектурних підходів, орієнтованих на модульність, масштабованість, тестованість і безпечність. Архітектура системи реалізована за принципом тривірневої структури (three-tier), де кожен рівень виконує власну роль: представлення (Presentation Layer), бізнес-логіка (Application Layer) та доступ до даних (Data Access Layer).

Рівень представлення

Фронтенд реалізовано за допомогою Angular з використанням Angular Material для побудови сучасного, адаптивного і доступного інтерфейсу користувача. Компоненти розроблялися згідно з принципами розділення відповідальності та повторного використання. Ключовий акцент зроблено на взаємодії користувача із системою за допомогою таблиць з серверною пагінацією, фільтрацією, сортуванням та формами редагування/створення сутностей. Комунікація між Angular-додатком і бекендом здійснюється через RESTful API з використанням HTTP-запитів у форматі JSON [20-23].

Рівень бізнес-логіки

Цей рівень реалізовано на платформі ASP.NET Core з використанням сучасного підходу до побудови сервісно-орієнтованої архітектури. Центральною частиною цієї структури є реалізація шаблонів Repository та Unit of Work, які забезпечують абстрагування доступу до бази даних і централізоване управління транзакціями.

Усі репозиторії впроваджуються через механізм Dependency Injection, що дозволяє легко тестувати компоненти, дотримуватись принципу інверсії залежностей (SOLID) та забезпечити розширюваність системи. Для зменшення кількості дубльованого коду використовується AutoMapper у поєднанні з DTO

(Data Transfer Object), що забезпечує чітке відокремлення логіки API від внутрішніх моделей домену.

Для деталізації архітектурної структури інформаційної системи було побудовано серію UML-діаграм класів, які ілюструють ключові компоненти та їхню структурну організацію. З метою підвищення наочності діаграми були поділені за архітектурними шарами та функціональними ролями.

На рис. 2.1 зображено контролери Web API, які забезпечують маршрутизацію HTTP-запитів до відповідних частин логіки системи. Кожен контролер відповідає за обробку конкретного типу запитів, перевірку прав доступу та взаємодію з сервісним або репозиторним рівнем.

На рис. 2.2 представлено структуру контексту бази даних ArgusDbContext, який інкапсулює всі таблиці даних системи у вигляді властивостей типу DbSet<>. Саме через цей контекст реалізується взаємодія з Microsoft SQL Server, визначаються зв'язки між сутностями, а також виконуються транзакції.

Інтерфейси репозиторіїв, що забезпечують абстракцію над операціями доступу до даних, подано на рис. 2.3. Вони визначають загальні контракти для роботи з базою, забезпечуючи слабку зв'язаність між рівнями застосунку та полегшуючи тестування.

На рис. 2.4 показано реалізації репозиторіїв, кожен з яких реалізує свій інтерфейс та містить методи для опрацювання окремих типів сутностей: ресурсів, локацій, користувачів, ролей тощо. Такий підхід відповідає принципам SOLID і дозволяє ефективно організувати доступ до бази даних через шаблон Unit of Work [24].

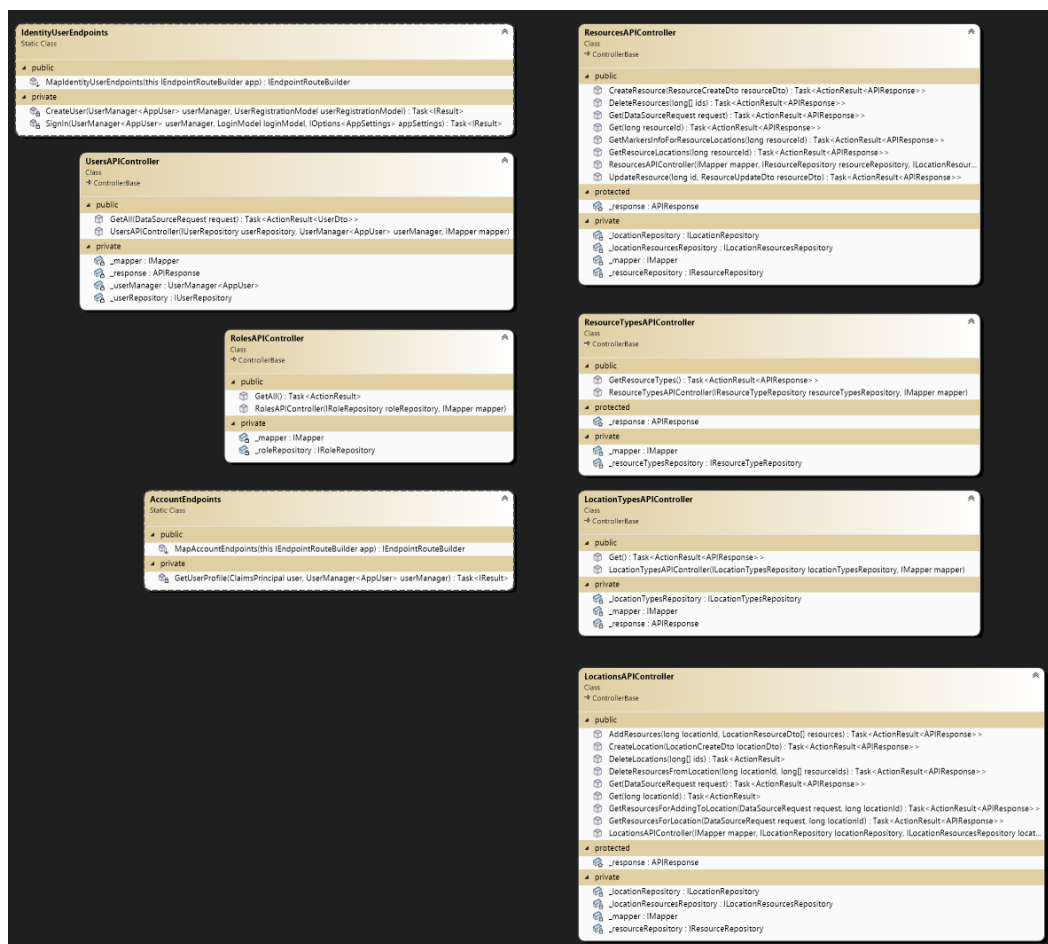


Рисунок 2.1 – Діаграма класів контролерів API інформаційної системи обліку та управління оборонними ресурсами

Джерело: розроблено автором

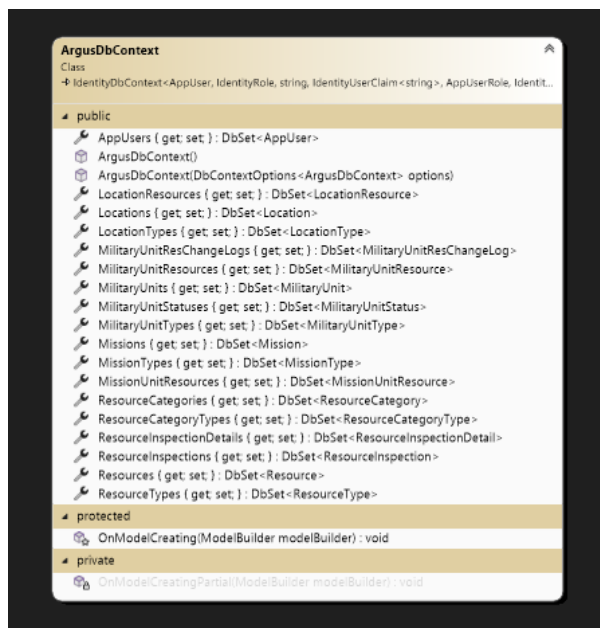


Рисунок 2.2 – Діаграма класу ArgusDbContext у структурі доступу до даних

Джерело: розроблено автором

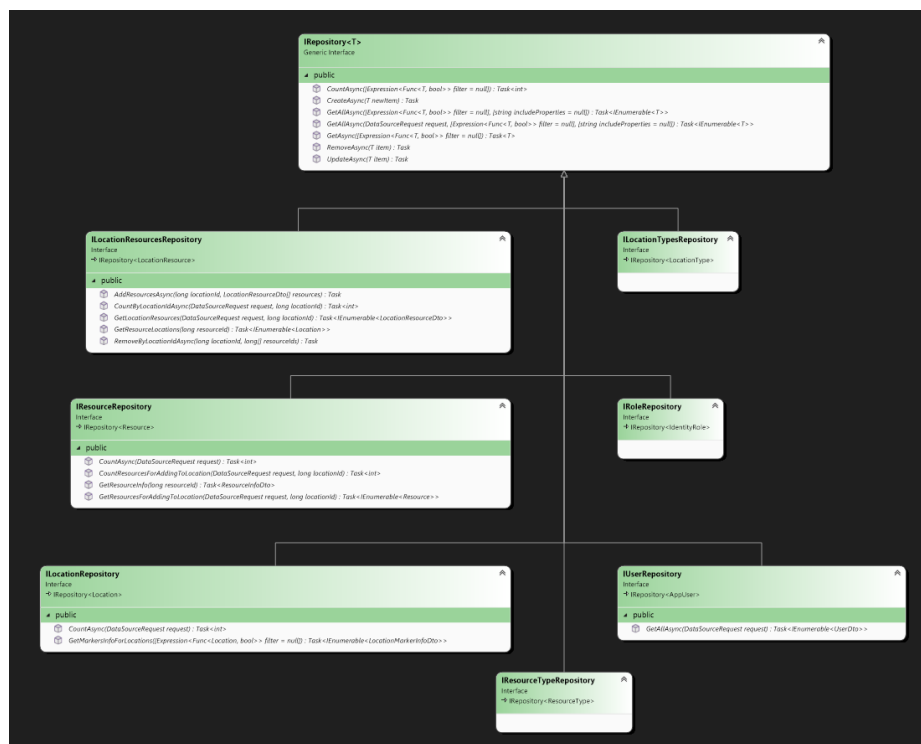


Рисунок 2.3 – Діаграма класів інтерфейсів репозиторіїв інформаційної системи обліку та управління оборонними ресурсами

Джерело: розроблено автором

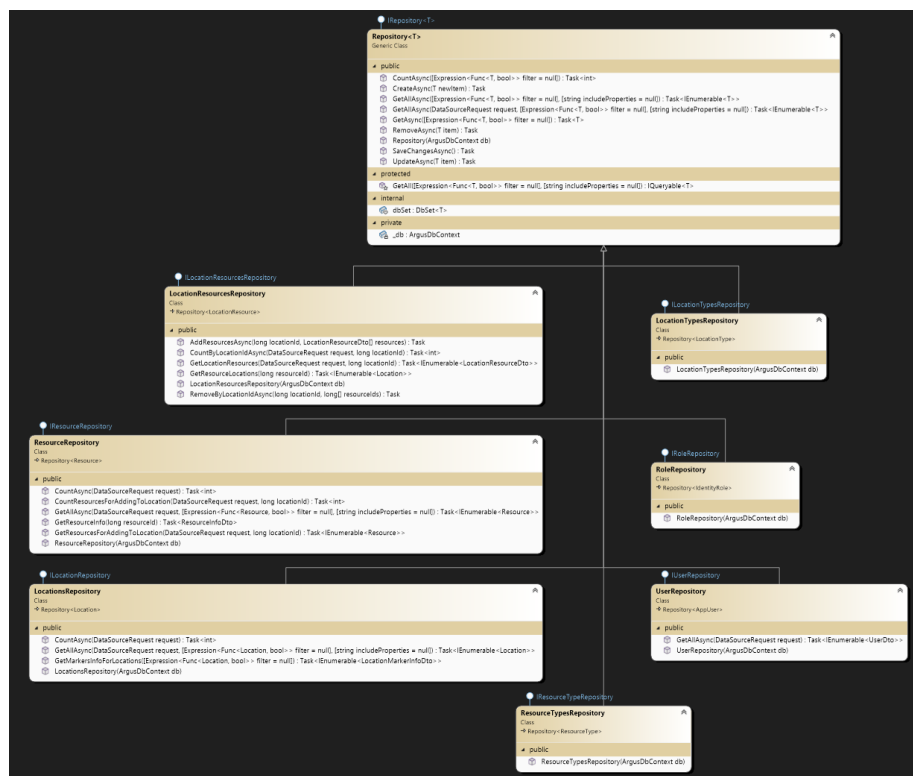


Рисунок 2.4 – Діаграма класів реалізацій репозиторіїв інформаційної системи обліку та управління оборонними ресурсами

Джерело: розроблено автором

Реалізовано автентифікацію та авторизацію з використанням ASP.NET Core Identity. Доступ до функціональності системи контролюється через ролі користувачів із застосуванням Claims-based авторизації. Завдяки цьому досягнуто гнучкого управління правами доступу на рівні API-контролерів.

У межах контролерів реалізовано серверну обробку запитів для Angular-таблиць — фільтрація, сортування, пагінація, вибірка по зв'язаних сутностях із використанням Lazy loading, що значно покращує продуктивність при роботі з великими наборами даних.

Рівень доступу до даних

База даних реалізована в Microsoft SQL Server. Початкове проєктування здійснено у парадигмі Database First. У подальшому для впровадження змін використовувалися механізми EF Core Migrations. Дані зберігаються у формі нормалізованих таблиць із відповідними зовнішніми ключами, індексами та обмеженнями цілісності. Для просторового розміщення ресурсів застосовано тип “GEOGRAPHY”, який дозволяє зберігати координати об'єктів і виконувати гео-запити, зокрема для аналізу розподілу ресурсів по території.

Переваги обраної архітектури

Завдяки такому архітектурному підходу досягається:

- легкість розширення системи без порушення існуючої логіки,
- зручне розгортання на різних середовищах (локальних або хмарних),
- можливість підключення додаткових модулів чи клієнтів без змін у структурі ядра системи,
- підтримка незалежної розробки окремих компонентів,
- чітка ізоляція відповідальності між рівнями.

Обрана модель дозволяє масштабувати систему як вертикально (ресурсами серверів), так і горизонтально (шляхом розділення компонентів на мікросервіси у майбутньому). Усе це робить архітектуру придатною до інтеграції в масштабніші оборонні IT-інфраструктури, що відповідає стратегічним вимогам військової

логістики та національної безпеки. Структурну взаємодію основних компонентів інформаційної системи наведено на рисунку 2.5.

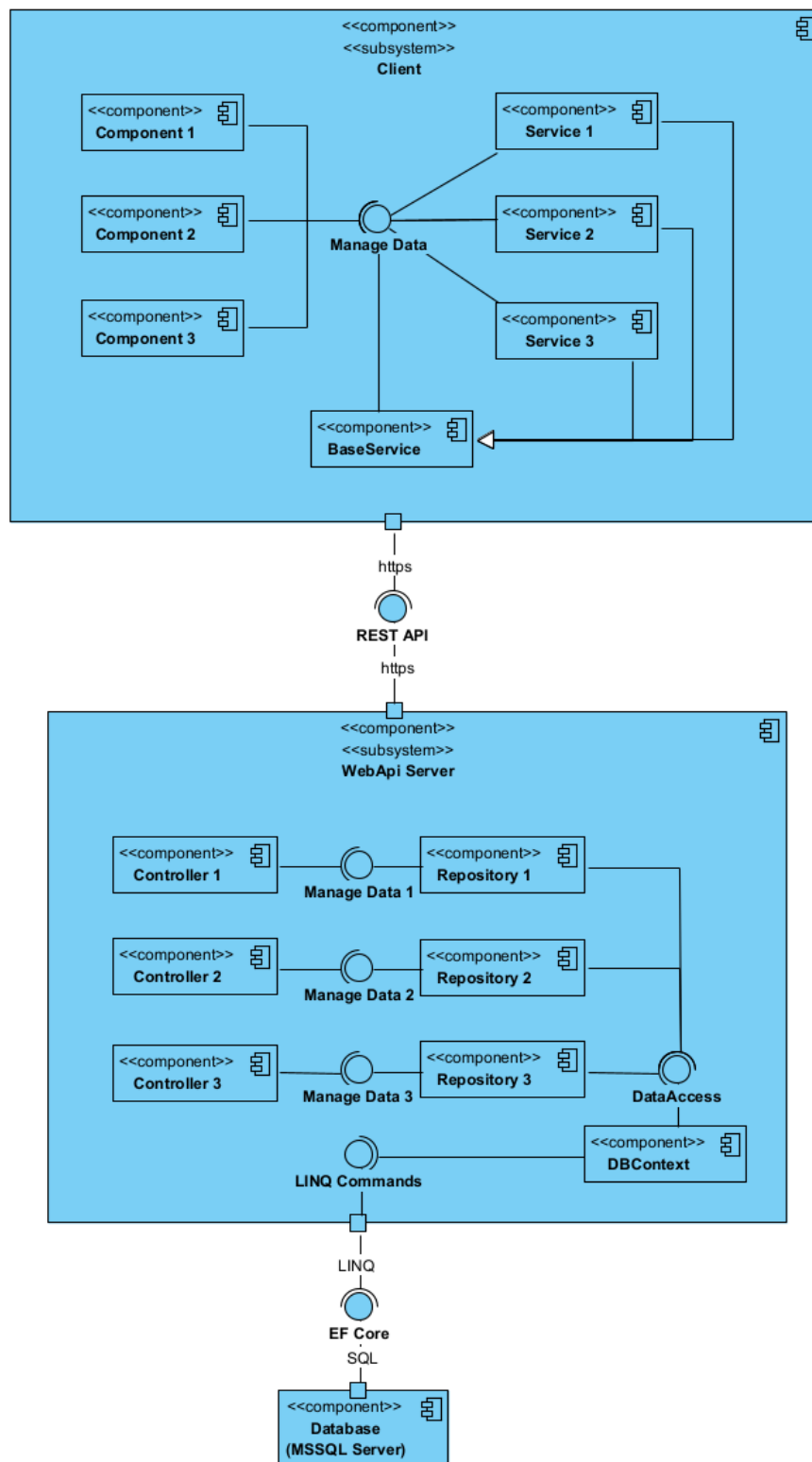


Рисунок 2.5 – UML-діаграма компонентів інформаційної системи обліку та управління оборонними ресурсами

Джерело: розроблено автором

Як показано на рисунку, система реалізує поділ на клієнтську частину (Angular), серверну частину (ASP.NET Core WebAPI), рівень обробки даних (EF Core) та рівень зберігання (SQL Server). Взаємодія між компонентами відбувається через RESTful API, а логіка обробки запитів організована на основі шаблонів Repository та Unit of Work. Така архітектура забезпечує масштабованість, незалежність компонентів та адаптивність до змін у функціональній моделі.

Для формалізації функціонального призначення системи та визначення сценаріїв її використання користувачами різних ролей було побудовано діаграму варіантів використання (див. рис. 2.6).

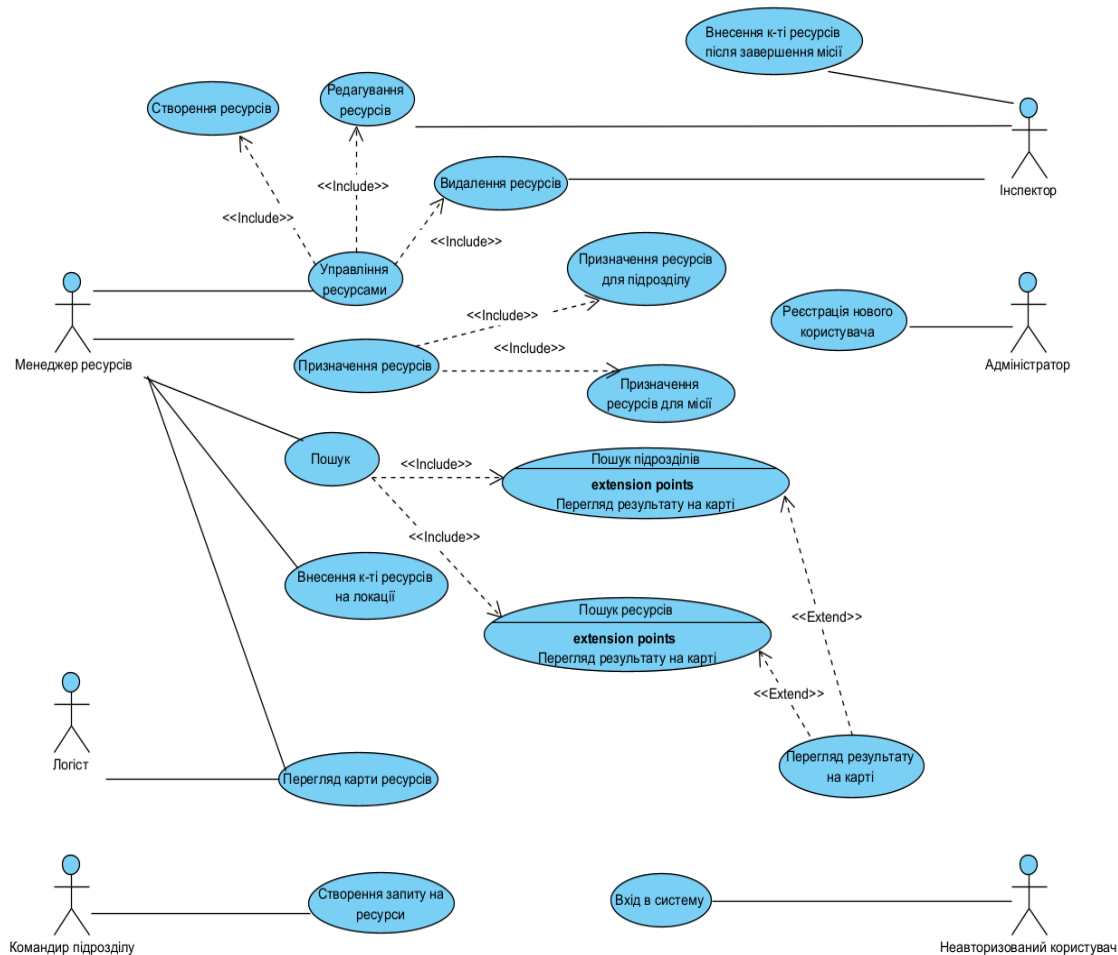


Рисунок 2.6 – UML-діаграма варіантів використання інформаційної системи обліку та управління оборонними ресурсами

2.2 Методи та моделі в інформаційних системах управління оборонними ресурсами

Під час проєктування та реалізації інформаційної системи обліку та управління оборонними ресурсами були використані сучасні методи моделювання, організації даних, архітектурного розділення компонентів та забезпечення захищеного доступу. Враховуючи вимоги до систем військово-логістичного призначення — такі як надійність, масштабованість, підтримка просторових операцій, обґрунтоване використання ресурсів та розширюваність — було обрано низку відповідних технологічних рішень, шаблонів і моделей.

Моделювання предметної області

Предметна область — це оборонні ресурси, організаційно-територіальні одиниці, логістичні потоки та операції з переміщення, обліку й списання матеріальних цінностей. Для її формалізації було побудовано логічну модель, у якій ресурси мають визначені категорії, типи, стани, призначення, координати розміщення та належать до певної локації.

В основі лежить **об'єктно-орієнтована модель**, реалізована засобами C# та EF Core. Вона передбачає чітку ієрархію сутностей із відповідними атрибутами, ключами та навігаційними властивостями. Модель підтримує Lazy Loading для ефективного завантаження зв'язаних об'єктів, що дозволяє оптимізувати витрати пам'яті при роботі з великими обсягами даних.

Зберігання та обробка просторових даних

Система має функціональність просторового позиціонування, яка реалізована через тип даних GEOGRAPHY у Microsoft SQL Server. Це дало змогу інтегрувати координати ресурсів, виконувати фільтрацію та агрегацію за територіальною ознакою, зокрема:

- фільтрацію ресурсів у межах радіуса від заданої точки;
- візуалізацію просторових даних у зовнішніх модулях;
- підготовку до інтеграції з геоінформаційними системами.

Архітектурні шаблони

Серверна частина системи реалізована за принципами чистої архітектури з використанням шаблонів:

- Repository — інкапсулює доступ до даних, забезпечуючи відокремлення бізнес-логіки від механізмів зчитування та збереження даних. Це дозволяє легко змінювати джерело даних або реалізацію без порушення структури застосунку.

- Unit of Work — об'єднує кілька репозиторіїв у межах одного запиту, забезпечуючи координацію змін і транзакційну цілісність даних.

- DTO (Data Transfer Object) — виконує функцію відділення внутрішніх моделей від транспортного рівня, що дозволяє створювати адаптовані форми даних для передачі між сервером і клієнтом.

- AutoMapper — автоматизує процес трансформації моделей у DTO та навпаки, що значно зменшує обсяг повторюваного коду та знижує ризик помилок.

Ці шаблони були обрані як перевірені індустріальні рішення, які сприяють модульності, тестованості та підтримуваності коду в довгостроковій перспективі.

Модель авторизації та захисту доступу

Забезпечення контролю доступу до функціональності реалізовано через ASP.NET Core Identity з ролями, політиками та JWT-аутентифікацією. Кожна роль (адміністратор, логіст, переглядач тощо) має чітко визначений набір дозволених дій, доступ до яких обмежено через політику авторизації. Це відповідає принципу RBAC (Role-Based Access Control).

Ідентифікація користувачів, перевірка прав доступу, шифрування токенів та їхнє обмеження в часі — усе це підвищує загальний рівень безпеки системи. Крім того, контролери серверної частини покриваються авторизаційними атрибутами, що унеможливорює несанкціонований доступ навіть при знанні шляхів API [25-34].

Методи обробки даних

Для ефективної роботи з великими наборами даних у системі реалізовано серверну фільтрацію, сортування та пагінацію. Застосування запитів LINQ

дозволяє реалізовувати гнучкі механізми агрегації, вибірки та аналізу, які можуть бути адаптовані під змінні вимоги користувачів.

Запити, які використовуються у звітах або таблицях Angular Material, формуються динамічно на сервері, що забезпечує високу продуктивність при роботі з тисячами записів та дозволяє уникати перевантаження клієнтської частини.

Обґрунтування вибору методів

Застосовані моделі та методи були обрані з урахуванням:

- вимог до систем обліку з критичним рівнем відповідальності;
- необхідності масштабованості та гнучкості в розгортанні (on-premise / cloud);
- досвіду впровадження подібних рішень у західних інформаційних системах оборонного призначення;
- підтримки модульності, повторного використання коду, тестованості та адаптації до змін.

РОЗДІЛ 3

РОЗРОБЛЕННЯ ПРОЕКТНИХ РІШЕНЬ ТА ЇХ РЕАЛІЗАЦІЯ В ІНФОРМАЦІЙНІЙ СИСТЕМІ УПРАВЛІННЯ ОБОРОННИМИ РЕСУРСАМИ

3.1 Проектування бази даних для інформаційної системи управління оборонними ресурсами

3.1.1 Інфологічна модель бази даних

На початковому етапі розроблення системи управління оборонними ресурсами було здійснено детальний аналіз предметної області з метою виявлення основних сутностей, їхніх властивостей та взаємозв'язків. Це дозволило сформулювати концептуальну (інфологічну) модель, яка відображає логіку предметної області, незалежно від обраної платформи реалізації.

Ключові сутності моделі охоплюють:

Resources – матеріальні об'єкти обліку, які підлягають зберіганню, переміщенню, інспекції та використанню. Кожен ресурс має унікальний серійний номер, тип та географічну позицію.

MilitaryUnits – організаційні одиниці з визначеною ієрархією (**ParentUnitId**), типом, статусом та призначенням командиром. Ієрархічна побудова дозволяє відобразити структуру ЗСУ або аналогічних організацій за рівнями командування.

Missions – військові місії або завдання з визначеними цілями, часовими рамками, ієрархічною підпорядкованістю та набором залучених ресурсів.

Locations – географічні точки (склади, бази, логістичні вузли), на яких фізично розміщуються ресурси. Усі локації пов'язані з підрозділами і мають типи (наприклад, “мобільна точка”, “основний склад”).

Users – персонал, який має доступ до системи. Для реалізації ролей користувачів застосовано таблиці **Roles** та **UserRoles**, що дозволяє створювати гнучку модель доступу.

3.1.2 Фізична модель бази даних

Фізична модель є конкретною реалізацією інфологічної структури з урахуванням вимог обраної платформи розробки. Для розгортання системи було обрано Microsoft SQL Server, що забезпечує розширену підтримку транзакцій, високий рівень безпеки, зручні засоби адміністрування та повну інтеграцію з технологіями ASP.NET Core.

Фізична модель побудована з дотриманням наступних принципів:

- Типізація полів. Використано типи bigint, int, smallint, varchar, text, datetime, а також geography для представлення координат.
- Цілісність даних. Усі таблиці мають первинні ключі (PK), зовнішні ключі (FK), обмеження на обов'язковість (NOT NULL) та перевірки (CHECK), що унеможливорює логічно некоректні операції.
- Аудит. У критичних таблицях передбачено поля для фіксації автора змін, часу останнього оновлення, зокрема: AuthorId, UpdatedAt, CreatedAt.
- Індокси. Для оптимізації вибірок додано індокси до полів, які використовуються в запитах (наприклад, SerialNumber, MissionId, UnitId, InspectionDate).
- Просторові поля. Застосування типу geography дозволяє інтегрувати геолокаційні функції (визначення відстаней, кластеризація за координатами, візуалізація на мапі).

Всі SQL-інструкції для створення структури бази даних надано у додатку А. Ці скрипти охоплюють створення таблиць, обмежень, індоксів, початкового наповнення довідників.

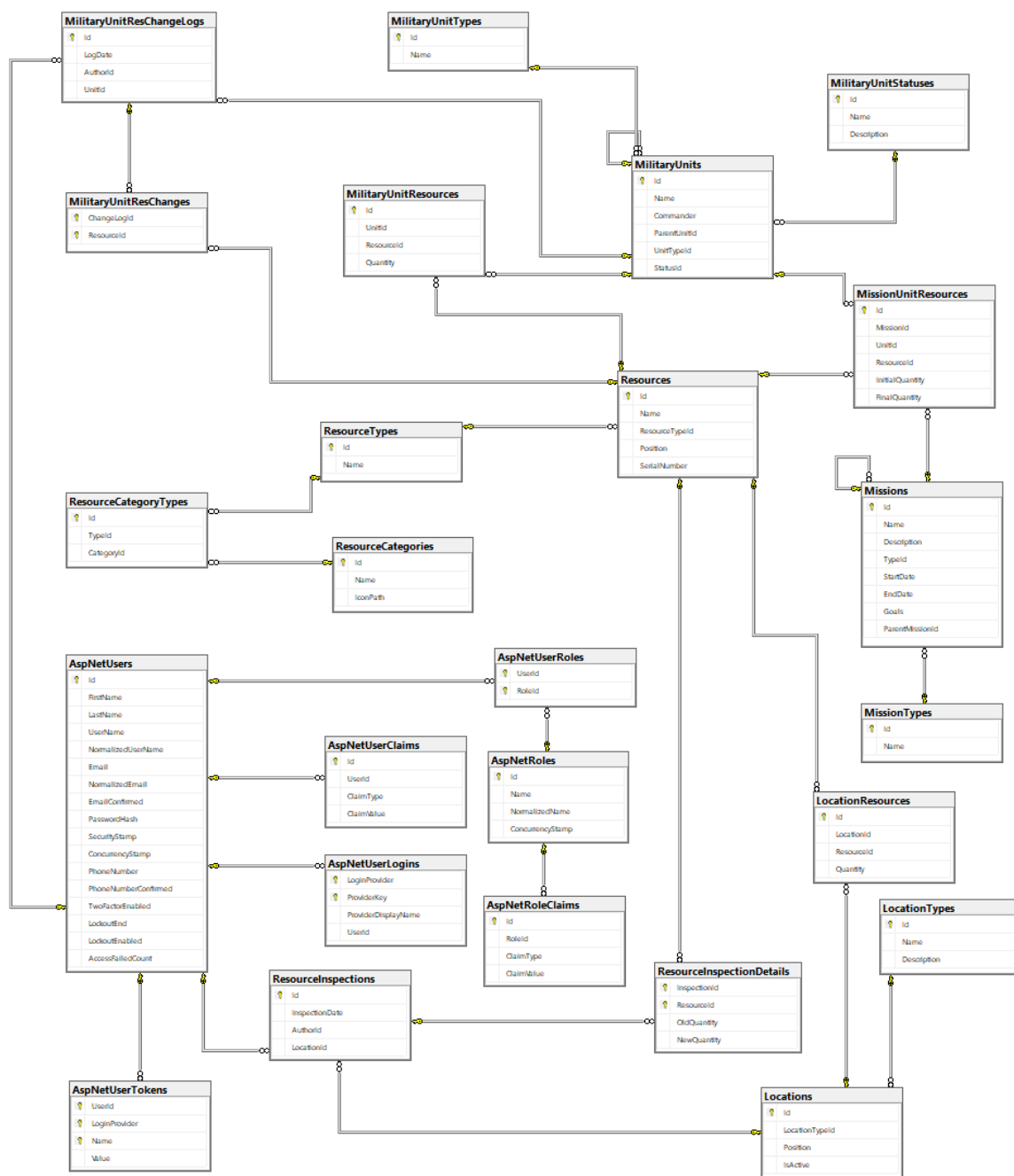


Рисунок 3.2 – Фізична модель бази даних, реалізована у Microsoft SQL Server Management Studio

Джерело: згенеровано автором

3.1.3 Оцінка обсягів даних та продуктивності

Для аналізу масштабованості та продуктивності системи було змодельовано типовий сценарій експлуатації. На основі очікуваної інтенсивності змін розраховано прогнозовані річні обсяги даних у таблицях:

Таблиця 3.1 – Прогнозовані обсяги даних у таблицях бази даних

Назва таблиці	Середній розмір запису, байт	Кількість записів/рік	Річний обсяг, МБ
MissionUnitResources	180	25000	4.29
MilitaryUnitResChangeLogs	220	10000	2.10
ResourceInspections	200	12000	2.29
Users	140	50	0.01
Missions	300	150	0.04
Resources	150	1000	0.14
MilitaryUnits	180	200	0.03

Ці обсяги не перевищують стандартні обмеження на розміщення в локальних або хмарних SQL Server, а обрана структура дозволяє масштабуватися без структурних змін навіть при зростанні кількості підрозділів і ресурсів у кілька разів.

3.2 Проектування бази знань інформаційної управляючої системи та/або засоби інтелектуального аналізу даних

У межах проектування та реалізації розробленої інформаційної управляючої системи не передбачалося впровадження окремої бази знань або компонентів інтелектуального аналізу даних. Такий підхід є обґрунтованим, виходячи зі специфіки предметної області та функціональних завдань системи, зосереджених на структурованому збереженні, обробці та візуалізації даних про оборонні ресурси.

Система оперує чітко визначеними сутностями — такими як матеріальні ресурси, локації зберігання, категорії обліку, статуси призначення тощо — і не включає задач, пов'язаних із нечіткою класифікацією, автоматичним виявленням закономірностей або автоматизованим ухваленням рішень. Усі процеси прийняття рішень залишаються на рівні користувача або відповідального оператора, що не вимагає наявності експертних правил, логічного виведення або онтологічної моделі знань.

Крім того, відсутність великого обсягу неструктурованих або семантично неоднорідних даних виключає потребу в застосуванні засобів інтелектуального аналізу даних, таких як кластеризація, класифікація, асоціативний пошук або нейромережеве прогнозування. Дані, що обробляються системою, здебільшого мають табличну структуру та належать до класу транзакційної інформації, що підлягає звичайній реляційній обробці.

Водночас, архітектура розробленої ІУС залишається відкритою для розширення. У разі виникнення потреби у реалізації модуля експертної підтримки рішень або впровадження засобів інтелектуального аналізу (наприклад, для прогнозування дефіциту ресурсів, виявлення аномалій у постачанні або моделювання логістичних сценаріїв), можлива інтеграція зовнішніх компонентів на основі CRISP-DM-підходу чи побудова бази знань відповідно до класичної або MDA-методології. Зокрема, перспективним може стати застосування технологій машинного навчання у поєднанні з експертними правилами в частині автоматизованого аналізу даних про переміщення або втрати ресурсів у різних територіальних підрозділах.

3.3. Програмне забезпечення

3.3.1 Проєктування програмного забезпечення

Проєктування програмного забезпечення інформаційної системи для обліку та управління оборонними ресурсами здійснювалося з урахуванням системних принципів побудови ІУС, включаючи структурованість, відкритість, масштабованість, адаптивність, надійність і безпечність. Основою для формування логіки функціонування виступили вимоги, визначені в технічному завданні, результати інфологічного моделювання та аналіз предметної області.

Бізнес-логіка та функціональне наповнення системи

Бізнес-логіка системи передбачає реалізацію таких основних сценаріїв:

- облік і категоризація матеріальних оборонних ресурсів;
- розподіл ресурсів між підрозділами військової структури;

- зберігання інформації про локації, їхній тип, призначення;
- контроль змін стану і кількості ресурсів у часі;
- інспектування об'єктів і формування звітів перевірок;
- взаємодія між підрозділами під час виконання місій;
- контроль доступу та розмежування прав користувачів залежно від ролі (адміністратор, менеджер ресурсів, логіст).

Реалізація цих сценаріїв вимагає побудови багаторівневої структури ПЗ, здатної обробляти запити в режимі реального часу, гарантувати цілісність даних і забезпечувати інтуїтивно зрозумілу взаємодію з користувачем.

Обґрунтування вибору моделі розроблення

В основі розробки лежить класична модель клієнт-серверної архітектури, де фронтенд (Angular) взаємодіє з сервером через RESTful API (ASP.NET Core). На стороні клієнта застосовано архітектурний шаблон MVVM (через Angular) з виокремленням логіки у компонентах і сервісах. Серверна частина структурована за принципами MVC, що забезпечує розділення логіки контролерів, бізнес-сервісів та репозиторіїв.

Такий підхід дозволив реалізувати гнучку й адаптивну систему, здатну до масштабування й розширення, з розмежуванням відповідальності між модулями.

Використані мови, середовища та технології

Під час розробки застосовано такі інструменти:

1. Серверна частина:

- Мова: C#
- Платформа: ASP.NET Core Web API (версія 8.0)
- ORM: Entity Framework Core (гібрид Code First + Scaffold)
- Аутентифікація: JWT + ASP.NET Core Identity (розширений AppUser)
- DI-контейнер, AutoMapper, конфігурація в Program.cs
- Запуск: Visual Studio 2022

2. Клієнтська частина:

- Фреймворк: Angular (v16)
- UI-бібліотека: Angular Material, кастомізовані компоненти
- Сервіси для HTTP-запитів, RxJS, Guards, Reactive Forms
- Запуск: Visual Studio Code

3. База даних:

- СУБД: Microsoft SQL Server
- Міграції: EF Core Migrations
- Генерація початкових моделей: Scaffold-DbContext

4. Інше:

- Контроль версій: Git + GitHub
- Візуалізація карти: Leaflet (через Angular wrapper) [37]
- Проєктування UML: Visual Paradigm 17.2 [38]

Архітектура та компоненти системи

У рамках серверної частини реалізовано понад десять Web API контролерів, серед яких: UsersController, ResourcesController, LocationsController, RolesController, InspectionsController, тощо. Вони оперують DTO-моделями, які відображають доменні сутності (Resource, Location, AppUser, Mission тощо), зокрема ResourceDto, LocationCreateDto, ResourceUpdateDto, UserDto, RoleDto.

Клієнтська частина містить понад 15 компонентів, зокрема: map, locations, resources, create-resource-dialog, sidebar, admin-panel, login, register-user. Застосовано механізм маршрутизації з guard'ами (authGuard, adminRedirect), а також реалізовано інтерцептор для автоматичної вставки JWT у запити.

Фільтрація, сортування та пагінація реалізовані **на стороні сервера**, що оптимізує продуктивність і дозволяє ефективно працювати з великими обсягами даних.

Переваги та властивості розробленої ІУС

Обрана структура ПЗ забезпечує такі властивості:

- Надійність — через валідацію запитів, контроль транзакцій, захист маршрутизаторів;
- Безпечність — завдяки JWT-аутентифікації та ролям доступу;
- Адаптивність і масштабованість — завдяки розділенню логіки на окремі модулі;
- Відкритість до розширення — можливість додавання нових модулів, ендпоінтів і компонентів без переробки ядра;
- Мобільність розгортання — система може запускатися локально, а за потреби легко деплоїться у хмарні середовища.

3.3.2 Вибір архітектури програмного забезпечення

У процесі розроблення програмного забезпечення інформаційної системи управління оборонними ресурсами архітектура стала фундаментом, що визначає логіку побудови, масштабованість, підтримуваність і подальшу еволюцію системи. Обрана архітектура формує каркас, у межах якого визначено канали обміну даними між компонентами, способи взаємодії користувача із системою, принципи обробки запитів і механізми зберігання інформації.

З урахуванням задач предметної області, вимог до надійності, відкритості та безпечності, було обрано модульну багаторівневу архітектуру з реалізацією класичної клієнт-серверної моделі. Такий підхід поєднує в собі розділення відповідальності між компонентами, простоту підтримки і гнучкість масштабування. Архітектурне рішення базується на комбінації двох основних шаблонів: MVC (Model–View–Controller) — для серверної частини та MVVM (Model–View–ViewModel) — для фронтенду.

Рівнева структура системи

Архітектура системи охоплює чотири логічні рівні:

1. Презентаційний рівень (Front-end)

Формує користувацький інтерфейс і реалізований у вигляді SPA-додатку на Angular 19.2.6. Застосовано Standalone-компоненти, що дозволяє уникнути надлишкової складності та забезпечити ізольованість логіки. Відображення, логіка і маршрутизація організовані за принципами MVVM:

компоненти реалізують View;

сервіси виконують роль ViewModel, взаємодіючи з API та реактивно обробляючи потоки подій через RxJS і Angular signals;

дані, що надходять із сервера, відображаються з використанням Angular Material UI-компонентів.

Присутні механізми захисту маршруту на клієнтській стороні (Route Guards), а також централізоване оновлення токенів через HTTP-інтерцептор [39-45].

2. Серверний рівень (Back-end/API)

Серверна логіка реалізована на платформі ASP.NET Core Web API. Архітектура організована за патерном MVC: контролери приймають запити, валідують параметри, викликають методи репозиторіїв та повертають DTO-об'єкти. Логіка не дублюється у сервісах, а делегується безпосередньо репозиторіям, що дозволяє зменшити кількість зайвих зв'язків у невеликому проєкті.

До складу API входять:

- аутентифікація за допомогою JWT, генерація токена при логіні;
- обмеження доступу через механізм ролей (Адміністратор, Менеджер ресурсів, Логіст);
- контроль доступу до методів API через [Authorize(Roles = "...")].

3. Рівень доступу до даних (Data Access Layer)

Для взаємодії з реляційною базою даних MSSQL Server використано Entity Framework Core [46-51]. Початкова схема була створена в СУБД, після чого за допомогою Scaffold-DbContext згенеровано моделі, які надалі модифікувалися через механізм Code First Migration — такий підхід класифікується як гібридний.

Доступ до даних інкапсулюється через:

- Generic-репозиторій (Repository<T>), який реалізує стандартні CRUD-операції;
- Інтерфейси репозиторіїв, які дозволяють наслідувати від IRepository<T> та визначати додаткові методи;
- DTO-класи, поділені за сценаріями (CreateDto, UpdateDto, InfoDto).

Архітектурна модель системи візуалізована на рисунку 3.3, який ілюструє логічне розділення компонентів на окремі рівні, а також шляхи взаємодії між Angular-додатком, серверною частиною ASP.NET Core та рівнем доступу до даних. Такий поділ забезпечує структурованість і слабке зв'язування модулів, що є основою для масштабованості та підтримуваності системи.

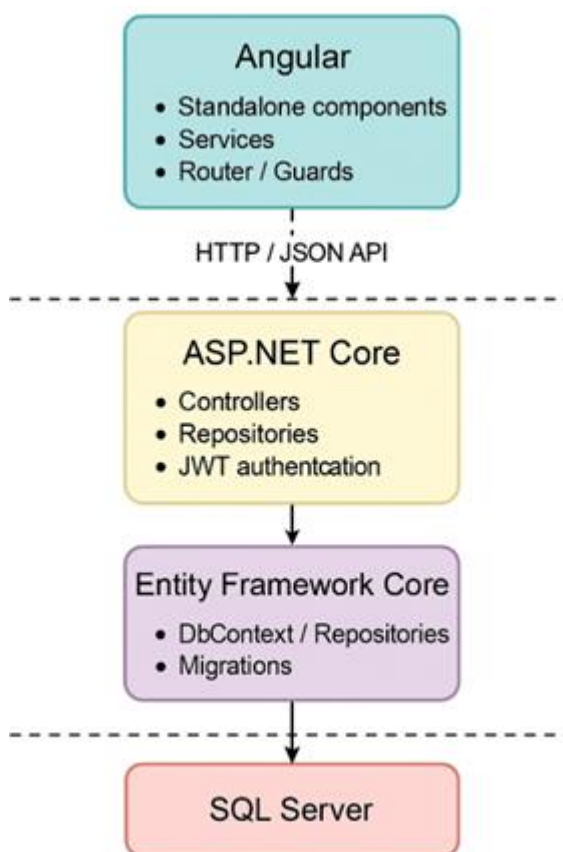


Рисунок 3.3 – Архітектурна структура системи

Джерело: створено автором

4. Інфраструктурний рівень

Містить конфігураційні файли (appsettings.json, середовищні конфігурації), а також логіку авторизації, DI-контейнер, маршрутизацію, JWT middleware. Хоча

централізований механізм обробки помилок не реалізовано, структура дозволяє легко додати global error handler у вигляді custom middleware.

Обґрунтування вибору

Модель побудови на основі MVC+MVVM з модульною структурою забезпечує:

- гнучкість — компоненти та API легко модифікувати і масштабувати;
- відокремленість відповідальності — презентаційна логіка, бізнес-логіка та доступ до даних поділені на рівні;
- безпеку — централізована автентифікація, role-based access, client-side guards;
- відкритість до інтеграції — RESTful API спрощує зв'язування з іншими системами, наприклад мобільними клієнтами або зовнішніми сервісами;
- супроводжуваність — завдяки узгодженим підходам до іменування, застосуванню AutoMapper та DI-контейнера, систему легко підтримувати та розширювати.

Можливості подальшого розвитку

Поточна архітектура закладає основу для таких перспектив:

- впровадження централізованого логування (ILogger, Serilog) або моніторингу;
- реалізація кешування (IMemoryCache, Redis);
- інтеграція сторонніх аналітичних сервісів;
- винесення функціональних модулів в окремі мікросервіси або об'єднання з іншими системами оборонного планування;
- застосування state management (NgRx) для складних сценаріїв на фронтенді;
- реалізація розширеного аудиту, історії дій користувачів та версійності змін.

3.3.3 Інструментальні засоби розробки для створення прикладного програмного забезпечення

Для повноцінного створення інформаційної управляючої системи, яка виконує облік, моніторинг та адміністрування оборонних ресурсів, було залучено комплекс сучасних інструментальних засобів. Їх добір здійснювався з урахуванням технічного завдання, вимог до безпеки, продуктивності та масштабованості.

Середовища розробки

Visual Studio 2022

Є основним середовищем для реалізації серверної частини системи. Вибір Visual Studio обумовлений широким спектром функцій, серед яких:

- повна підтримка ASP.NET Core 8.0;
- інтеграція з NuGet-пакетами, Git та Entity Framework Core;
- можливість запуску проекту через IIS Express;
- вбудований дебагер для аналізу HTTP-запитів, авторизації, DTO-моделей;
- менеджер міграцій EF Core для керування схемою бази даних;
- автогенерація Swagger-документації.

Visual Studio Code

Застосовувався для фронтенд-розробки. Це легке, кросплатформене IDE, яке дозволяє:

- швидко редагувати шаблони HTML, SCSS та TypeScript-код;
- використовувати Angular CLI та інтегровану термінальну оболонку;
- застосовувати розширення для ESLint, Angular Language Service, GitLens;
- організовувати проєкт за допомогою Workspace, Split Editors та Snippets.

SQL Server Management Studio (SSMS)

Цей інструмент забезпечував роботу з базою даних Microsoft SQL Server. З його допомогою:

- здійснювався аналіз зв'язків між таблицями;
- писалися складні SQL-запити для перевірки цілісності даних;
- переглядалися транзакції, індекси, статистики запитів;
- тестувалися тригери, зовнішні ключі, обмеження;
- виконувалося ручне резервне копіювання.

Visual Paradigm 17.2

Використано для проєктування структурних діаграм, включаючи:

- діаграми класів;
- діаграми прецедентів;
- діаграми компонентів;
- відображення взаємодії між сутностями (Resource, Location, MilitaryUnit) та контролерами API [35].

Серверні технології

ASP.NET Core 8.0

Складова, яка є основою серверної частини системи. Реалізовано модулі для:

- контролю доступу до ресурсів за ролями;
- генерації JWT-токенів із параметрами безпеки (алгоритм HMAC SHA-256, термін дії, ключ шифрування — див. додаток Б);
- створення RESTful API з маршрутами /api/resources, /api/locations, /api/users тощо;
- обробки HTTP-запитів через контролери (ResourcesController, LocationsController) — код контролерів наведено в додатку В.

ASP.NET Core Identity з розширенням AppUser

Імплементовано автентифікацію з додатковими полями (FirstName, LastName), а також зв'язками з таблицями UserRoles, ResourceInspections. У коді AppUser.cs (див. додаток Г) показано, як розширено базовий клас IdentityUser.

Також реалізовано захист API через [Authorize]-атрибути.

Entity Framework Core (EF Core)

Забезпечує ORM-зв'язок між класами C# та таблицями MSSQL.

Особливості реалізації:

- первинне створення моделей за допомогою Scaffold-DbContext;
- подальше внесення змін через Code First-міграції;
- підтримка зв'язків «один до багатьох» і «багато до багатьох»

(LocationResources, MilitaryUnitResources);

- конфігурація Fluent API (див. фрагмент OnModelCreating() у додатку Г).

AutoMapper

Застосовується для трансформації сутностей у DTO та навпаки. У MappingProfile.cs (додаток 3) наведено конфігурацію для автоматичного проектування Resource → ResourceDto, Location → LocationDto.

Фронтенд-технології

Angular 19.2.6

Забезпечує побудову SPA з реактивною логікою. У проєкті:

- реалізовано близько 15 Standalone-компонентів (map, locations, resources, admin-panel тощо);
- використано Reactive Forms, HttpClient, Guards для контролю доступу;
- збудовано власні сервіси для взаємодії з API (location.service.ts, resource.service.ts);
- логіка стану між компонентами реалізована через signals та RxJS Subjects.

Angular Material

UI-компоненти з високим рівнем кастомізації:

- mat-table для виведення ресурсів із серверною пагінацією та фільтрацією;
- mat-dialog для форм створення/редагування;
- mat-sidenav, mat-toolbar, mat-menu — для побудови навігаційної структури інтерфейсу.

Leaflet

Застосовується для візуалізації розміщення ресурсів на мапі. У коді `map.component.ts` створено інтерактивну карту з маркерами, що отримуються через API (`/api/resources/{id}/locations/markers-info`) [36].

Інтеграція та підтримка

Git та GitHub

Контроль версій здійснювався через:

- створення гілок для окремих функцій;
- рев'ю змін через Pull Request;
- аудит історії комітів для відстеження помилок;
- збереження резервних копій коду та документації.

Postman

Використовувався для тестування:

- запитів із JWT-аутентифікацією;
- сценаріїв реєстрації, логіну, редагування ресурсів;
- перевірки авторизації з різними ролями.

Таким чином, залучення вказаних інструментальних засобів забезпечило комплексну реалізацію усіх аспектів розробки — від фронтенду й бекенду до рівня бази даних та візуалізації геоданих. Їхнє узгоджене застосування сприяло формуванню структурованої, масштабованої та надійної інформаційної системи, що відповідає сучасним вимогам до ІУС оборонного призначення.

3.3.4 Тестування програмного забезпечення

Тестування програмного забезпечення є обов'язковим етапом забезпечення якості інформаційної системи та підтвердження її відповідності поставленим функціональним і нефункціональним вимогам. Метою тестування є виявлення дефектів, оцінка працездатності ключових функцій, перевірка коректності взаємодії між компонентами, а також забезпечення стабільності поведінки системи за типових і граничних сценаріїв її використання.

У процесі тестування інформаційної системи управління оборонними ресурсами основна увага була приділена ручному функціональному тестуванню, орієнтованому на перевірку працездатності модулів авторизації, обліку ресурсів, взаємодії з локаціями, форм інтерфейсу, маршрутизації та взаємодії з картою.

У таблиці 3.2 наведено приклади тест-кейсів, що відображають типові сценарії перевірки ключових функціональних модулів системи.

Таблиця 3.2 – Фрагмент тест-кейсів для перевірки функціональності системи

Test Case ID	Модуль / компонент	Умова тестування	Очікуваний результат	Фактичний результат	Статус
TC-01	Аутентифікація	Вхід із правильними обліковими даними	Користувача авторизовано, повернено JWT	Відповідає	Пройдено
TC-02	Аутентифікація	Вхід з неправильним паролем	Виведено повідомлення про помилку	Відповідає	Пройдено
TC-03	Ролі / доступ	Користувач без ролі "Адміністратор" відкриває маршрут /admin-panel	Перенаправлення на сторінку «403 Заборонено»	Відповідає	Пройдено
TC-04	Інтерфейс ресурсу	Створення ресурсу з порожнім полем «Назва»	Виведено повідомлення про обов'язкове поле	Відповідає	Пройдено
TC-05	Інтерактивна карта (Leaflet)	Завантаження мапи з пов'язаними ресурсами	Відображено всі маркери з координатами	Відповідає	Пройдено
TC-06	API /api/locations	Запит GET без JWT	Статус відповіді 401 Unauthorized	Відповідає	Пройдено
TC-07	Таблиця ресурсів	Видалення останнього запису у таблиці	Відображається повідомлення «Немає записів»	Виправлено	Пройдено
TC-08	Guard на Angular	Вхід з простроченим JWT	Перенаправлення на сторінку логіну	Відповідає	Пройдено
TC-09	API /api/resources	POST із валідними даними	Ресурс створено, повернено статус 201 Created	Відповідає	Пройдено

Джерело: розроблено автором

Основні об'єкти тестування

1. Користувацький інтерфейс (Angular)

Перевірено відображення основних компонентів: locations, resources, map, admin-panel, діалогових вікон створення/редагування ресурсів. Особлива увага приділена перевірці:

- реакції на некоректні вхідні дані (валідація форм);

- переходів між сторінками з урахуванням ролей;
- реакції інтерфейсу на зміну даних у реальному часі (через RxJS/signals);

- коректності відображення маркерів на карті за допомогою Leaflet.

2. API-рівень (ASP.NET Core)

Перевірено RESTful-ендпоінти:

- /api/resources, /api/locations, /api/users на відповідність CRUD-операціям;
- коректність відповідей на GET/POST/DELETE-запити з валідними та невалідними параметрами;
- реакцію API на запити без авторизації та з несанкціонованим доступом;
- роботу авторизаційного механізму через JWT (перевірка застарілих, відсутніх та підроблених токенів).

3. Валідація на сервері та клієнті

Усі форми перевірено на наявність обов'язкових полів, обмежень щодо форматів, довжини, типу даних. У разі помилок перевірено коректність виведення повідомлень.

4. Управління ролями та безпекою

Реалізовано ручне тестування обмежень доступу для ролей: Адміністратор, Менеджер ресурсів, Логіст. Перевірено:

- недоступність адміністративних функцій для неадміністратора;
- блокування доступу до певних API-ендпоінтів без JWT або з чужим JWT;
- обмеження навігації через Angular Guards.

Виявлені помилки та їх обробка

У ході тестування було виявлено декілька незначних помилок:

- некоректне відображення порожньої таблиці після видалення останнього запису;

- дублювання маркерів на карті після оновлення даних без оновлення сторінки;
- спрацьовування Guard'ів з затримкою при швидкому оновленні токена.

Усі помилки були усунені на етапі розробки завдяки ітеративному підходу з одночасною перевіркою працездатності функціоналу.

Тестування підтвердило відповідність реалізованого ПЗ функціональним вимогам. Усі ключові модулі працюють стабільно, інтерфейс реагує відповідно до сценаріїв взаємодії користувача, механізми авторизації працюють коректно. Наявна структура проєкту дозволяє легко інтегрувати автоматизовані тести на подальших етапах розвитку.

3.3.5 Керівництво (інструкція) користувача

Функціонування інформаційної системи обліку та управління оборонними ресурсами передбачає взаємодію з користувачем через багаторівневий інтерфейс, побудований відповідно до принципів інтуїтивної навігації, структурованості та розмежування доступу. У системі реалізовано авторизацію, контекстну взаємодію з основними об'єктами (ресурсами, локаціями, підрозділами) та інтерактивне відображення інформації за допомогою картографічного модуля.

Функціональні можливості користувачів відрізняються залежно від призначеної ролі у системі. У таблиці 3.3 наведено відповідність між основними функціями програмного забезпечення та доступними правами для кожної категорії користувачів.

Таблиця 3.3 – Функціональні можливості користувачів залежно від ролі у системі

Функціональність	Адміністратор	Менеджер ресурсів	Логіст
Вхід до системи	+	+	+
Перегляд ресурсів	+	+	+
Створення / редагування ресурсів		+	
Видалення ресурсів		+	
Призначення ресурсів підрозділам / місіям		+	

Продовження таблиці 3.3

Функціональність	Адміністратор	Менеджер ресурсів	Логіст
Призначення ресурсів на локації		+	
Перегляд інтерактивної карти ресурсів	+	+	+
Створення нових локацій		+	
Створення запитів на ресурси			+
Пошук підрозділів		+	+
Перегляд результатів на мапі	+	+	+
Реєстрація нових користувачів	+		
Доступ до адміністративної панелі	+		

Джерело: розроблено автором

Авторизація та початкова взаємодія

Після запуску системи користувач ідентифікується через форму авторизації, де вводяться email та пароль. Облікові записи створюються виключно адміністратором через інтерфейс керування користувачами, самостійна реєстрація не передбачена. Після успішної автентифікації користувача перенаправляє на початкову сторінку, яка визначається залежно від його ролі.

Інтерфейс адаптовано до українськомовного середовища, усі елементи меню, кнопки, сповіщення й повідомлення про помилки відображаються українською. Навігація реалізована у вигляді бічної панелі з пунктами, динамічно сформованими на основі доступних прав. Контроль доступу до маршрутів реалізовано за допомогою механізмів Router Guards на клієнтському рівні, а також відповідних перевірок на сервері.

Робота з ресурсами

У розділі «Ресурси» користувач отримує доступ до таблиці, що реалізована на основі Angular Material Table з підтримкою:

- фільтрації за типами і категоріями;
- серверної пагінації;
- сортування;
- масових дій над вибраними записами;
- відкриття перегляду деталей окремого ресурсу.

Менеджер ресурсів має право створювати нові об'єкти обліку за допомогою діалогової форми (create-resource-dialog), яка реалізує реактивну валідацію:

некоректні поля підсвічуються, система не дозволяє підтвердити створення до усунення помилок. Аналогічно реалізовано оновлення даних ресурсу. Усі дії супроводжуються системними сповіщеннями у вигляді toast-повідомлень.

Управління локаціями та просторове представлення

Локації є геопросторовими точками, де зберігаються оборонні ресурси. Менеджер ресурсів має можливість створювати нові локації із вказанням координат, типу й призначення. Після створення, локації автоматично з'являються на інтерактивній мапі, реалізованій з використанням бібліотеки Leaflet.

У розділі «Карта» усі локації відображаються як маркери. Користувач має змогу натискати на маркер, унаслідок чого з'являється вікно з короткою інформацією: назва локації, тип, наявні категорії ресурсів. Пряме редагування локацій з карти не передбачено; для цього слід скористатися відповідним розділом системи.

Пошук та призначення ресурсів

Реалізовано декілька сценаріїв пошуку:

- ресурсів за параметрами;
- підрозділів за назвами;
- відповідних зв'язків між ресурсами та місіями або локаціями.

Менеджер ресурсів має змогу призначати ресурси підрозділам чи місіям через відповідні діалогові інтерфейси. Для забезпечення прозорості взаємодії реалізовано відображення результатів пошуку на мапі.

Адміністративне управління

Адміністратор має доступ до розділу admin-panel, де реалізовано функціональність реєстрації користувачів. При створенні облікового запису вказуються ім'я, прізвище, електронна адреса та роль. Після збереження користувач має змогу здійснювати вхід до системи.

Повідомлення, винятки та сторінки стану

1. Усі дії системи супроводжуються відповідними повідомленнями (успішне збереження, помилки сервера, обмеження доступу).

2. У випадку некоректного введення валідаційні повідомлення виводяться під відповідними полями.

3. Перехід до недоступного ресурсу або некоректного маршруту спричиняє відображення службових сторінок 403 або 404, що містять пояснювальний текст.

Рекомендації з використання

1. Система оптимізована для роботи в сучасних десктопних браузерах (Google Chrome, Firefox, Microsoft Edge).

2. Мобільна версія не передбачена.

3. Рекомендується завершувати сесію через опцію «Вийти».

4. Зміни в системі застосовуються без потреби перезавантаження сторінки, завдяки реактивній архітектурі.

3.4. Технічне забезпечення

3.4.1 Обґрунтування вибору технічного забезпечення

Розроблення та експлуатація інформаційної системи для обліку та управління оборонними ресурсами потребує належного технічного забезпечення, здатного забезпечити стабільну роботу клієнтської та серверної частин програмного забезпечення, обробку запитів до бази даних, рендеринг картографічних компонентів та захист інформаційної взаємодії. У даному пункті наведено обґрунтування вибору технічного забезпечення, що використовувалося під час розробки, а також розглянуто можливі варіанти розгортання системи в умовах реального застосування.

Огляд технічного середовища предметної області

В установах оборонного сектору України технічне забезпечення зазвичай включає:

– автоматизовані робочі місця (АРМ) на базі ПК середнього рівня (Intel Core i5 / i7, 8–16 ГБ ОЗП);

- локальні сервери з обмеженою кількістю одночасних сесій, часто на базі Windows Server або Linux;
- внутрішні мережі з обмеженим доступом до Інтернету, у т.ч. ізольовані сегменти;
- інфраструктурні шлюзи або VPN, що забезпечують доступ до централізованих серверів із підпорядкованих підрозділів.

У зв'язку з цим доцільно орієнтуватися на рішення, які можуть функціонувати як у локальному середовищі, так і в умовах обмеженого мережевого доступу, з можливістю масштабування до хмарного формату в перспективі.

Практичне середовище розроблення

У межах кваліфікаційної магістерської роботи розроблення та тестування системи здійснювалося на персональному комп'ютері з наступними характеристиками:

- Процесор: Intel Xeon E5-2660 v2 @ 2.20GHz;
- Оперативна пам'ять: 16 ГБ;
- Накопичувач: SSD Samsung 860 EVO SATA, 250 ГБ;
- Операційна система: Windows 10 Home (64-bit);
- Сервер API: запуск через IIS Express;
- База даних: Microsoft SQL Server 2016 (локально, на тому самому ПК);
- Фронтенд: Angular 19.2.6, запуск через ng serve з ASP.NET сертифікатом;
- Інструменти: Visual Studio 2022, Visual Studio Code, SQL Server Management Studio 20;
- Браузер тестування: Google Chrome.

Система стабільно функціонувала при одночасному запуску всіх компонентів. Час оновлення Angular-проєкту при внесенні змін складав ≈ 1 секунду. Рендеринг Leaflet-мапи з десятками маркерів не створював затримок. Середнє споживання пам'яті коливалося у межах 2.5 ГБ.

Порівняльний аналіз варіантів розміщення

З метою обґрунтування архітектурних рішень здійснено порівняльний аналіз можливих моделей розгортання інформаційної системи. У таблиці 3.3 наведено основні варіанти технічного розміщення програмного забезпечення із зазначенням їх переваг та обмежень.

Таблиця 3.3 – Порівняльна характеристика моделей розгортання ІУС

Варіант	Переваги	Обмеження
Локальний сервер	Відсутність залежності від Інтернету, контроль безпеки	Високі вимоги до адміністрування, обмежена масштабованість
Централізований сервер у штабі	Можливість роботи через VPN, централізоване резервування	Залежність від каналів зв'язку, ризик перевантаження
Хмарна інфраструктура	Масштабованість, легка підтримка, швидке відновлення після збоїв	Потребує стабільного VPN-доступу, чутливість до кіберзагроз

Джерело: розроблено автором

З огляду на специфіку предметної області та структуру підпорядкованості, оптимальним є поєднання централізованого серверного ядра з розподіленим клієнтським доступом через захищені канали (локальна мережа, VPN).

Перспективи розвитку технічного середовища

Передбачено, що в подальшому система може:

- бути розгорнута на віртуальних платформах (Azure, AWS, Docker);
- масштабуватися через поділ API і БД на окремі вузли;
- інтегруватися з мобільними пристроями та пристроями збору геоданих;
- реалізувати резервне копіювання та моніторинг у режимі реального часу.

Застосоване технічне середовище дозволило повністю реалізувати функціональні вимоги до системи, підтвердило її працездатність в умовах, наближених до експлуатаційних, та забезпечило базу для масштабування без потреби повної перебудови інфраструктури. Вибір Хеоп-платформи, SSD-накопичувача і сучасних середовищ розробки виправданий як з погляду продуктивності, так і з огляду на реальні можливості ІТ-інфраструктури установ сектору оборони.

3.4.2 Ресурсні вимоги до технічного забезпечення

Розроблена інформаційна система є прикладом клієнт-серверного застосунку з розділенням на три логічні шари — презентаційний, прикладної логіки та доступу до даних. Відповідно, вимоги до технічного забезпечення формувалися з урахуванням ресурсного навантаження на кожен із цих компонентів та загальної продуктивності системи в умовах локального використання.

На етапі розроблення та тестування проєкт було реалізовано в середовищі одного фізичного комп'ютера, на якому були розгорнуті всі компоненти програмної архітектури:

1. Серверна частина — ASP.NET Core Web API, що працює в середовищі IIS Express. Для належного функціонування цього компонента потрібне ефективне багатопотокове обчислення, стабільна робота механізмів авторизації/аутентифікації (JWT), управління користувачами (ASP.NET Core Identity) та обробки REST-запитів.

2. Клієнтська частина — вебзастосунок на Angular (v19.2.6), що обслуговується через Angular CLI (ng serve). Додаток використовує Angular Material для побудови інтерфейсу, інтерактивну карту на базі Leaflet.js, а також функціональність route guards, пагінації, фільтрації та асинхронної взаємодії з API.

3. База даних — Microsoft SQL Server 2016, яка функціонує локально на тій самій машині. Вона зберігає структуровані дані, пов'язані з обліком оборонних ресурсів, географічним розташуванням об'єктів, історією змін, ролями користувачів, логістичними процесами тощо.

Тестування системи проводилося на наступній апаратній конфігурації:

Процесор: Intel Xeon E5-2660 v2 (10 фізичних ядер, 20 потоків, частота 2.2 ГГц). Завдяки підтримці великої кількості потоків процесор забезпечив ефективну паралельну обробку одночасних HTTP-запитів, що дозволило зберігати низький час відповіді навіть при навантаженні з боку Angular-інтерфейсу.

Оперативна пам'ять: 16 ГБ DDR3, чого було достатньо для стабільної роботи середовищ Visual Studio 2022, SQL Server Management Studio 20, Angular

CLI, браузера Google Chrome із відкритими вкладками системи та емулятором Leaflet-карти.

Накопичувач: SSD Samsung 860 EVO (SATA, 250 ГБ). Забезпечив необхідну швидкодію під час зчитування/запису великих обсягів даних у базу, завантаження Angular-компонентів і запуск серверних сервісів. Висока швидкість доступу до диску дозволила уникнути затримок при відкритті великих SQL-запитів, виконанні міграцій EF Core та рендерингу складних сторінок UI.

Операційна система: Windows 10 Home (x64). Незважаючи на те, що дана версія ОС не є серверною, вона дозволила коректно розгорнути IIS Express та інші компоненти, що використовуються в ході розробки та локального розгортання проєкту.

Браузер: Google Chrome — використано для тестування адаптивності інтерфейсу, роботи карти, виводу таблиць та повідомлень. Проведено тестування на різних розширеннях екрану з використанням Chrome DevTools.

Важливим фактором у визначенні технічних вимог також стали архітектурні рішення, реалізовані в проєкті. Зокрема:

- Використання ORM EF Core (гібрид Code First + Scaffold) із застосуванням міграцій;
- Аутентифікація через JWT-токени, що вимагає обчислювальних ресурсів при валідації запитів;
- Картографічні компоненти Leaflet, які споживають пам'ять та графічні ресурси при візуалізації великої кількості маркерів.

У разі розгортання розробленої ІУС в реальному середовищі (у межах військових або логістичних підрозділів), мінімальні ресурсні вимоги до робочої станції або сервера мають становити:

- ЦПУ: не нижче Intel Core i5 8-го покоління або AMD Ryzen 5 (4 ядра, 8 потоків);
- ОЗП: не менше 8 ГБ (рекомендовано 16 ГБ);
- Накопичувач: SSD від 256 ГБ;

- ОС: Windows 10 Pro / Windows Server 2016 або новіше;
- Мережа: стабільне з'єднання з локальним сервером або хмарним сховищем;
- Браузер: сучасна версія Chrome, Edge або Firefox.

Таким чином, сформульовані ресурсні вимоги повністю відповідають сучасним стандартам розроблення та локального використання інформаційних систем. За потреби масштабування або організації розподіленого доступу з різних підрозділів, доцільним буде винесення API-сервера на окрему машину з більшою кількістю оперативної пам'яті, розгортання БД на окремому SQL-сервері, а також налаштування HTTPS, брандмауерів та VPN-доступу.

3.5 Реалізація інформаційної системи управління оборонними ресурсами

Інформаційна система для обліку та управління оборонними ресурсами, розроблена у межах даної кваліфікаційної магістерської роботи, реалізована як модульна вебплатформа з клієнт-серверною архітектурою, що відповідає вимогам сучасних систем військового та логістичного призначення. Основна ідея реалізації полягає у створенні зручного, масштабованого та безпечного середовища для ефективного управління ресурсами у межах підрозділів із урахуванням їхньої географічної дислокації, належності до організаційної структури та функціонального призначення.

На етапі реалізації системи впроваджено ключові функціональні блоки: автентифікація та авторизація користувачів, управління ролями, перегляд і редагування даних про ресурси, картографічна інтеграція, а також модуль підрозділів, що відображає ієрархічну структуру організації.

Прототипування: візуалізація інтерфейсу

На етапі передпроектного аналізу було розроблено серію прототипів основних вікон системи, які дозволили сформулювати загальну візуальну концепцію, перевірити логіку користувацької взаємодії та узгодити стилістичні рішення (див.

рис. 3.4 – 3.8). Ці макети стали вихідною точкою для побудови кінцевих Angular-компонентів.

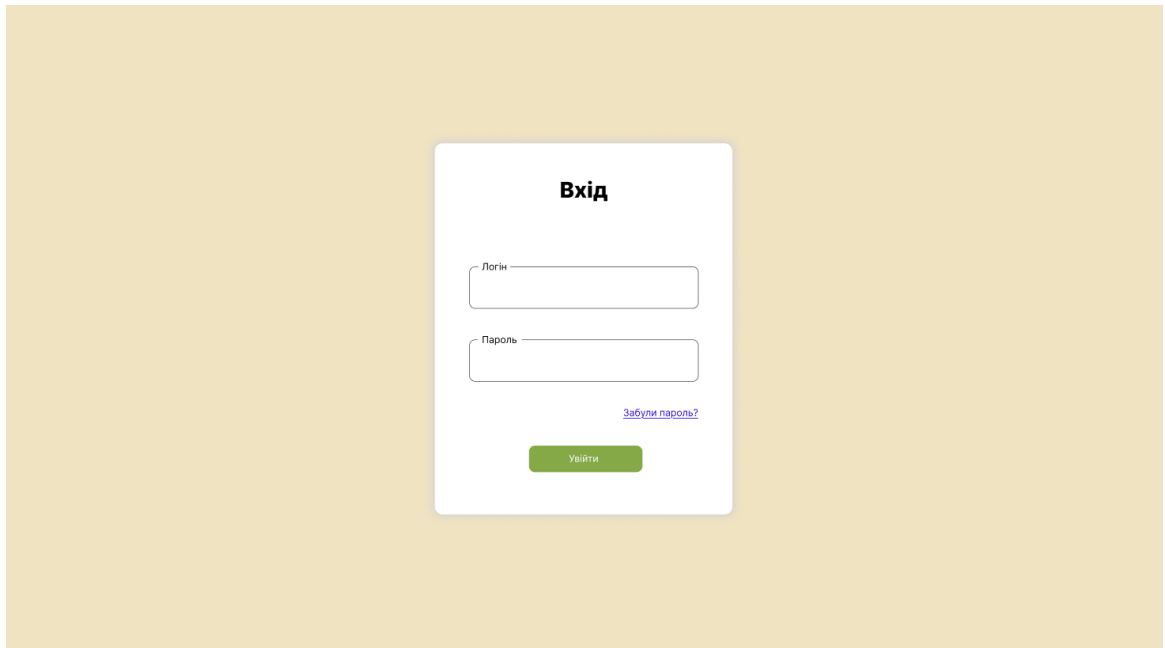


Рисунок 3.4 – Прототип вікна входу

Джерело: створено автором



Рисунок 3.5 – Прототип вікна з картою

Джерело: створено автором

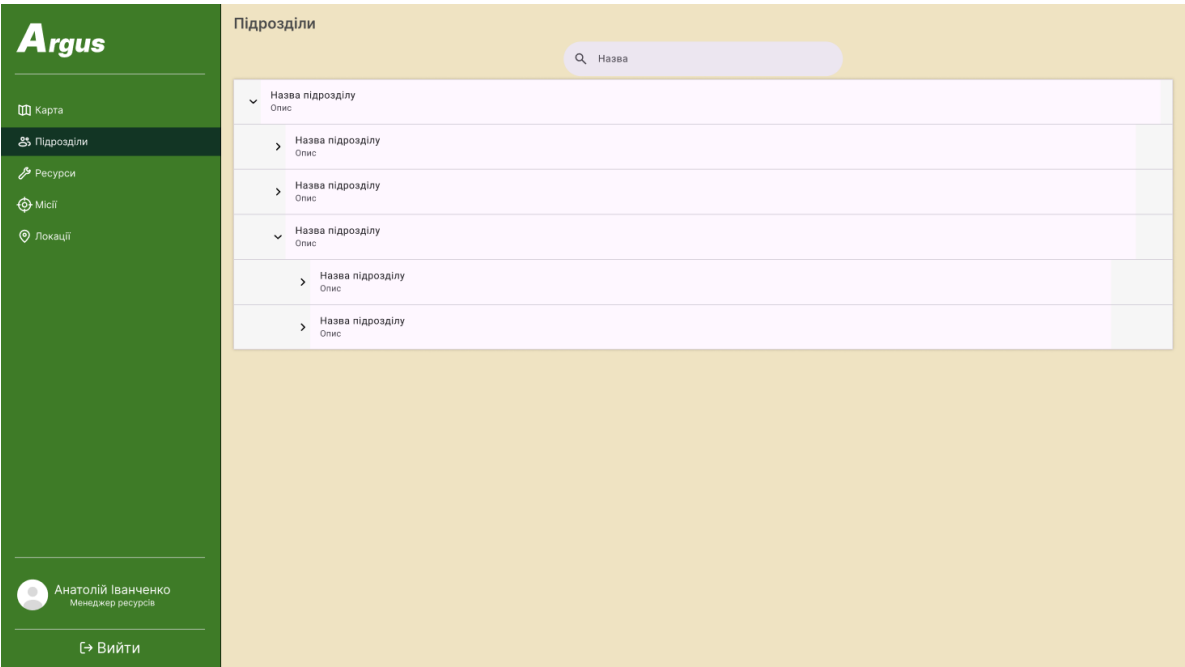


Рисунок 3.6 – Прототип вікна підрозділів

Джерело: створено автором

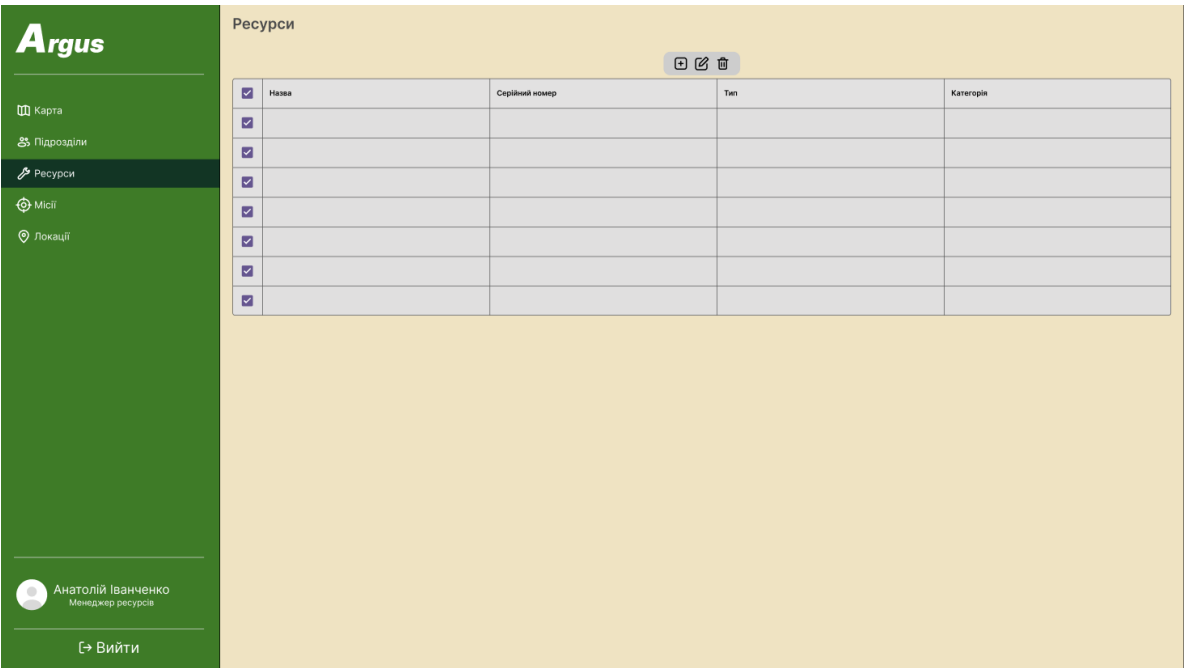


Рисунок 3.7 – Прототип вікна ресурсів

Джерело: створено автором

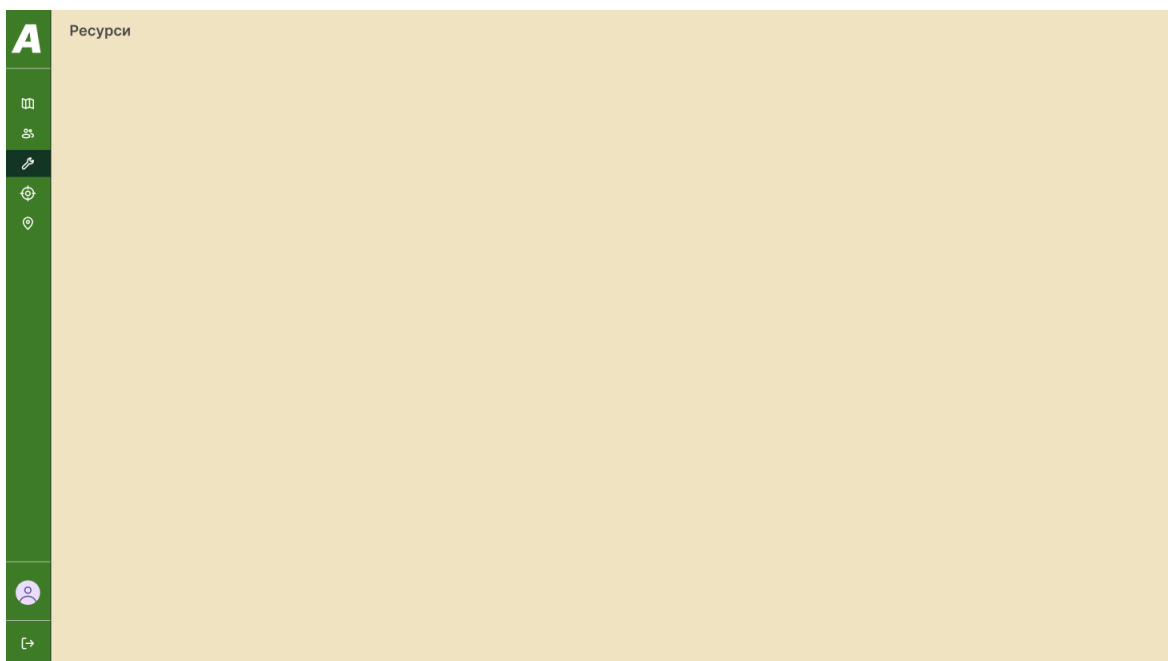


Рисунок 3.8 – Прототип вікна зі згорнутим сайдбаром

Джерело: створено автором

Прототипи дозволили мінімізувати ризики перевантаження інтерфейсу та підвищити ергономіку, зокрема для роботи у польових умовах з обмеженим доступом до великих екранів. Фінальні реалізації кожного з наведених вікон представлено у вигляді знімків робочої системи в Додатку Д.

Реалізовані модулі та технологічна основа

Функціональна реалізація системи ґрунтується на поділі архітектури за принципом «Separation of Concerns» та включає:

- Серверну частину, реалізовану на ASP.NET Core з JWT-аутентифікацією, механізмом Identity для управління користувачами, контролерами REST, застосуванням AutoMapper, DTO та валідацією вхідних даних;
- Фронтенд, реалізований на Angular v19.2.6, з використанням Angular Material, реактивних форм, модулів маршрутизації, захисту маршрутів, ладерів і кастомних пайпів;
- Сховище даних — SQL Server 2016, із реалізованою базою даних на основі поєднання Code First і Scaffold-підходів, із застосуванням міграцій, зв'язків «багато до багатьох» та індексації ключових таблиць.

Особливу увагу приділено безпеці: реалізовано перевірку JWT на кожен запит, окремий маршрут для refresh-токенів, логування подій входу, а також доступ за ролями на рівні як API, так і UI.

Перевірка та апробація результатів

Систему було протестовано в умовах емуляції багатокористувацького середовища. Проведено функціональне, інтеграційне та навантажувальне тестування з використанням Postman, браузерних інструментів Chrome DevTools та інструментів Angular CLI. Отримано стабільну швидкість завантаження сторінок, коректну валідацію запитів і підтримку одночасного доступу з кількох облікових записів.

ВИСНОВКИ

У результаті виконання кваліфікаційної магістерської роботи на тему «Розроблення інформаційної системи для обліку та управління оборонними ресурсами» реалізовано повноцінний комплекс заходів, спрямованих на проєктування, розроблення, тестування та впровадження інформаційної системи, яка відповідає сучасним вимогам автоматизації обліку ресурсів оборонного призначення. Сформульовані у вступі цілі та завдання повністю досягнуті, що свідчить про завершеність і практичну цінність розробки.

У теоретичній частині дослідження здійснено ґрунтовний аналіз предметної області управління оборонними ресурсами. Було визначено ключові поняття, процеси та об'єкти, пов'язані з логістичним обліком, оперативним розподілом і моніторингом військового майна, техніки, пального, медичних засобів тощо. Особливу увагу приділено аналізу чинних підходів до побудови інформаційних систем у військовій та оборонній галузях, а також вивченню досвіду стандартизації згідно з принципами НАТО (зокрема, стандартами STANAG). На основі цього було виявлено основні проблеми існуючих рішень — фрагментарність, обмежена масштабованість, відсутність централізованої інтеграції, низький рівень прозорості й безпеки.

Обґрунтовано вибір підходів та технологій для створення системи, зокрема — використання стеку ASP.NET Core (C#), Angular, MSSQL Server, що дозволяє реалізувати трирівневу архітектуру з чітким поділом на шари: Presentation, API, Data Access. Такий підхід забезпечує гнучкість у розробці, можливість масштабування, повторне використання компонентів і високу підтримуваність коду. У процесі моделювання було використано інструменти Visual Paradigm для побудови діаграм класів, компонентів, прецедентів, а також спроектовано повноцінну інфологічну та фізичну моделі бази даних. Особливу увагу приділено нормалізації структури, використанню ключів, індексів, зв'язків типу «багато до багатьох» та впровадженню механізму міграцій.

У частині розробки програмного забезпечення реалізовано клієнтську частину з використанням Angular та бібліотеки Angular Material. Здійснено маршрутизацію з guard-захистом, реалізовано інтерфейсну адаптацію під ролі користувачів (адміністратор, логіст, менеджер ресурсів). Фронтенд взаємодіє з API через REST-запити, із вбудованою валідацією, пагінацією та фільтрацією. Здійснено інтеграцію з картою Leaflet, що дозволяє географічно відображати розміщення підрозділів і наявних ресурсів. Серверну частину побудовано з використанням ASP.NET Core, Identity, JWT-аутентифікації, AutoMapper, FluentValidation. Організовано чіткий розподіл логіки по рівнях, застосовано шаблони Repository і Unit of Work, що гарантує низький рівень зв'язності.

У рамках реалізації здійснено повний цикл тестування: функціональне (на відповідність вимогам), інтеграційне (перевірка взаємодії модулів), навантажувальне (робота з великою кількістю ресурсів і паралельних сесій). Система показала стабільну роботу при одночасному доступі користувачів з різними ролями, без втрати продуктивності або безпеки. Проведено аналіз ресурсних характеристик і сформовано перелік технічних вимог до серверної інфраструктури, рекомендовано середовище розгортання на базі IIS Express у середовищі Windows 10 з використанням SQL Server.

Сформульовано рекомендації щодо впровадження системи в умовах оборонних відомств, територіальної оборони та військових частин. Визначено можливості подальшого розвитку системи — зокрема, включення мобільного клієнта, реалізація модуля аналітики (BI), інтероперабельність з ERP-рішеннями, інтеграція із зовнішніми джерелами даних через API, а також впровадження модулів ІАД на основі зібраної історії переміщення та споживання ресурсів.

Таким чином, кваліфікаційна магістерська робота є завершеним проєктом, що поєднує теоретичну новизну, практичну значущість і відповідає вимогам освітньо-професійної програми спеціальності 122 «Комп'ютерні науки». Отримані результати можуть бути використані як у реальному впровадженні в оборонних структурах, так і в навчальному процесі при підготовці фахівців з інформаційних управляючих систем і технологій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Міністерство оборони України. Міноборони разом із Генштабом ЗСУ розпочали підготовку до впровадження військової версії SAP в управлінні оборонними ресурсами [Електронний ресурс]. – Режим доступу: <https://mod.gov.ua/news/minoboroni-razom-iz-genshtabom-zsu-rozpochali-pidgotovku-do-vprovadzhennya-vijskovoyi-versiyi-sap-v-upravlinnya-oboronnimi-resursami>. – Дата звернення: 15.05.2025.
2. Про затвердження Положення про систему управління оборонними ресурсами : Наказ Міністерства оборони України від 14.04.2021 № 115 [Електронний ресурс]. – Режим доступу: <https://zakon.rada.gov.ua/laws/show/z0594-21> – Назва з екрана. – Дата звернення: 15.05.2025.
3. Про національну безпеку України : Закон України від 21.06.2018 № 2469-VIII [Електронний ресурс]. – Режим доступу: <https://zakon.rada.gov.ua/laws/show/2469-19> – Назва з екрана. – Дата звернення: 15.05.2025.
4. Про оборону України : Закон України від 06.12.1991 № 1932-XII [Електронний ресурс]. – Режим доступу: <https://zakon.rada.gov.ua/laws/show/1932-12> – Назва з екрана. – Дата звернення: 15.05.2025.
5. Про державне оборонне замовлення : Закон України від 03.03.1999 № 464-XIV [Електронний ресурс]. – Режим доступу: <https://zakon.rada.gov.ua/laws/show/464-14> – Назва з екрана. – Дата звернення: 15.05.2025.
6. Стратегія національної безпеки України, затверджена Указом Президента України від 14.09.2020 № 392/2020 [Електронний ресурс]. – Режим доступу: <https://zakon.rada.gov.ua/laws/show/392/2020> – Назва з екрана. – Дата звернення: 15.05.2025.
7. Стратегія розвитку оборонно-промислового комплексу України до 2030 року [Електронний ресурс]. – Режим доступу:

https://ukroboronprom.com.ua/storage/documents/report_eng.pdf – Назва з екрана. – Дата звернення: 15.05.2025.

8. Про затвердження Стратегічного оборонного бюлетеня України : Указ Президента України від 17.09.2021 № 473/2021 [Електронний ресурс]. – Режим доступу: <https://www.president.gov.ua/en/news/glava-derzhavi-zatverdiv-strategichnij-oboronnij-byuleten-uk-70713> – Назва з екрана. – Дата звернення: 15.05.2025.

9. Про затвердження Положення про організацію обліку оборонних ресурсів : Наказ Міністерства оборони України від 27.03.2019 № 141 [Електронний ресурс]. – Режим доступу: <https://zakon.rada.gov.ua/laws/show/z0384-19> – Назва з екрана. – Дата звернення: 15.05.2025.

10. Про затвердження Положення про автоматизовану систему управління оборонними ресурсами : Наказ Міністерства оборони України від 20.07.2020 № 245 [Електронний ресурс]. – Режим доступу: <https://zakon.rada.gov.ua/laws/show/z0750-20> – Назва з екрана. – Дата звернення: 15.05.2025.

11. Про затвердження Порядку планування оборонних ресурсів : Постанова Кабінету Міністрів України від 25.06.2020 № 500 [Електронний ресурс]. – Режим доступу: <https://zakon.rada.gov.ua/laws/show/500-2020-%D0%BF> – Назва з екрана. – Дата звернення: 15.05.2025.

12. Шматкова М.І. Управління оборонними ресурсами та забезпечення їх стійкості // Соціальний розвиток і безпека. – 2023. – №13(2). – С. 112–128. [Електронний ресурс]. – Режим доступу: <https://paperssds.eu/index.php/JSPSDS/article/view/515> – Назва з екрана. – Дата звернення: 15.05.2025.

13. Ковальчук В.І. Інформаційні системи управління в оборонній сфері: сучасний стан та перспективи розвитку // Військово-науковий вісник. – 2022. – №1. – С. 30–37. [Електронний ресурс]. – Режим доступу: <https://paperssds.eu/index.php/JSPSDS/article/view/697> – Назва з екрана. – Дата звернення: 15.05.2025.

14. Семененко О., Скуріневська Л. Показники та критерії оцінювання ефективності системи управління оборонними ресурсами сектору безпеки і оборони України // Наукові праці Центру воєнно-стратегічних досліджень. – 2023. – №13(2). – С. 45–52. [Електронний ресурс]. – Режим доступу: <https://paperssds.eu/index.php/JSPSDS/article/view/537> – Назва з екрана. – Дата звернення: 15.05.2025.

15. Міністерство оборони України. Міноборони завершило перший етап впровадження системи управління оборонними ресурсами в ЗСУ [Електронний ресурс]. – Режим доступу: <https://mod.gov.ua/news/minoboroni-zavershilo-pershij-etap-vprovadzhennya-sistemi-upravlinnya-oboronnimi-resursami-v-zsu>. – Дата звернення: 15.05.2025.

16. Kubra Ozcan. What is SAP (System Analysis Program Development)? [Електронний ресурс]. – Режим доступу: https://medium.com/@kubra.ozcan_98680/what-is-sap-system-analysis-program-development-6137d099bde7. – Дата звернення: 15.05.2025.

17. SAP. SAP for Defense and Security [Електронний ресурс]. – Режим доступу: <https://www.sap.com/industries/defense-security.html> – Назва з екрана. – Дата звернення: 15.05.2025.

18. NATO Allied Command Transformation. DELTA System CWIX [Електронний ресурс]. – Режим доступу: <https://www.act.nato.int/article/delta-system-cwix/>. – Дата звернення: 15.05.2025.

19. Приходько Р. Система Delta введена в експлуатацію для всіх підрозділів оборони України / Роман Приходько // Мілітарний. – 2024. – [Електронний ресурс]. – Режим доступу: <https://military.com/uk/news/systema-delta-vvedena-v-ekspluatatsiyu-dlya-usih-pidrozdiliv-oborony-ukrayiny/> – Назва з екрана. – Дата звернення: 15.05.2025.

20. Angular. Angular Material [Електронний ресурс]. – Режим доступу: <https://material.angular.io/>. – Дата звернення: 15.05.2025.

21. Microsoft. Best practices for RESTful web API design [Електронний ресурс] / Microsoft. – Режим доступу: <https://learn.microsoft.com/en->

[us/azure/architecture/best-practices/api-design](https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design) – Назва з екрана. – Дата звернення: 15.05.2025.

22. Swagger. Best Practices in API Design [Електронний ресурс] / Swagger. – Режим доступу: <https://swagger.io/resources/articles/best-practices-in-api-design/> – Назва з екрана. – Дата звернення: 15.05.2025.

23. Abdullah, S. Mastering REST API Design: Essential Best Practices, Do's and Don'ts for 2024 [Електронний ресурс] / Syed Abdullah. – Режим доступу: <https://medium.com/@syedabdullahrahman/mastering-rest-api-design-essential-best-practices-dos-and-donts-for-2024-dd41a2c59133> – Назва з екрана. – Дата звернення: 15.05.2025.

24. Microsoft Learn. Implementing the Repository and Unit of Work Patterns in an ASP.NET MVC Application [Електронний ресурс]. – Режим доступу: <https://learn.microsoft.com/en-us/aspnet/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application>. – Дата звернення: 15.05.2025.

25. Open Web Application Security Project (OWASP). JSON Web Token (JWT) Cheat Sheet for Java [Електронний ресурс]. – Режим доступу: https://cheatsheetseries.owasp.org/cheatsheets/JSON_Web_Token_for_Java_Cheat_Sheet.html – Назва з екрана. – Дата звернення: 15.05.2025.

26. JWT.io. JSON Web Tokens [Електронний ресурс]. – Режим доступу: <https://jwt.io/>. – Дата звернення: 15.05.2025.

27. Microsoft Learn. Introduction to Identity on ASP.NET Core [Електронний ресурс]. – Режим доступу: <https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-9.0> – Назва з екрана. – Дата звернення: 15.05.2025.

28. Patel R. A Beginner's Guide to ASP.NET Core Identity for Authentication and Authorization with .NET 8 [Електронний ресурс]. – Режим доступу: <https://medium.com/@ravipatel.it/a-beginners-guide-to-asp-net-core-identity-for-authentication-and-authorization-with-net-8-e6c8deb612f4> – Назва з екрана. – Дата звернення: 15.05.2025.

29. YogiHosting. How to Setup and Configure ASP.NET Core Identity [Електронний ресурс]. – Режим доступу: <https://www.yogihosting.com/aspnet-core-identity-setup/> – Назва з екрана. – Дата звернення: 15.05.2025.

30. Microsoft Learn. Configure JWT bearer authentication in ASP.NET Core [Електронний ресурс]. – Режим доступу: <https://learn.microsoft.com/en-us/aspnet/core/security/authentication/configure-jwt-bearer-authentication?view=aspnetcore-9.0> – Назва з екрана. – Дата звернення: 15.05.2025.

31. Getachew S. JWT Authentication in .NET 8: A Complete Guide for Secure and Scalable Applications [Електронний ресурс]. – Режим доступу: <https://medium.com/@solomongetachew112/jwt-authentication-in-net-8-a-complete-guide-for-secure-and-scalable-applications-6281e5e8667c> – Назва з екрана. – Дата звернення: 15.05.2025.

32. Getachew S. Getting Started with ASP.NET Core Identity: A Beginner's Guide to Best Practices with .NET 8 // Medium. – 2024. – 8 жовтня [Електронний ресурс]. – Режим доступу: <https://medium.com/@solomongetachew112/getting-started-with-asp-net-core-identity-a-beginners-guide-to-best-practices-with-net-8-04b17199c9d5> – Назва з екрана. – Дата звернення: 15.05.2025.

33. Telerik. ASP.NET Core Basics: Authentication and Authorization with JWT [Електронний ресурс]. – Режим доступу: <https://www.telerik.com/blogs/asp-net-core-basics-authentication-authorization-jwt> – Назва з екрана. – Дата звернення: 15.05.2025.

34. Telerik. New in .NET 8: ASP.NET Core Identity and How to Implement It [Електронний ресурс]. – Режим доступу: <https://www.telerik.com/blogs/new-net-8-aspnet-core-identity-how-implement> – Назва з екрана. – Дата звернення: 15.05.2025.

35. Copes F. JWT authentication: Best practices and when to use it // LogRocket. – 2024. – 12 вересня [Електронний ресурс]. – Режим доступу: <https://blog.logrocket.com/jwt-authentication-best-practices/> – Назва з екрана. – Дата звернення: 15.05.2025.

36. Curity. JWT Security Best Practices [Електронний ресурс]. – Режим доступу: <https://curity.io/resources/learn/jwt-best-practices/> – Назва з екрана. – Дата звернення: 15.05.2025.

37. Leaflet. An open-source JavaScript library for interactive maps [Електронний ресурс]. – Режим доступу: <https://leafletjs.com/> – Назва з екрана. – Дата звернення: 15.05.2025.

38. Visual Paradigm. What is Component Diagram? [Електронний ресурс]. – Режим доступу: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/>. – Дата звернення: 15.05.2025.

39. Tiwari S. Rest API Call In Angular Application [Електронний ресурс]. – Режим доступу: <https://medium.com/@sandeep15mca/rest-api-call-in-angular-application-289d907020ff> – Назва з екрана. – Дата звернення: 15.05.2025.

40. Angular. Angular Material [Електронний ресурс]. – Режим доступу: <https://material.angular.io/>. – Дата звернення: 15.05.2025.

41. Angular. Understanding communicating with backend services using HTTP [Електронний ресурс]. – Режим доступу: <https://v17.angular.io/guide/understanding-communicating-with-http> – Назва з екрана. – Дата звернення: 15.05.2025.

42. Angular. Get data from a server [Електронний ресурс]. – Режим доступу: <https://v17.angular.io/tutorial/tour-of-heroes/toh-pt6> – Назва з екрана. – Дата звернення: 15.05.2025.

43. Basiński M. Why Angular signals won't replace RxJS / Mateusz Basiński // Angular.love. – 2023. – 10 липня [Електронний ресурс]. – Режим доступу: <https://angular.love/why-angular-signals-wont-replace-rxjs> – Назва з екрана. – Дата звернення: 15.05.2025.

44. Ahmad A. Angular Modules vs. Standalone Components: A Detailed Comparison / Ausaf Ahmad // Medium. – 2025. – 24 січня [Електронний ресурс]. – Режим доступу: <https://medium.com/@ausaf.cs/angular-modules-vs-standalone-components-a-detailed-comparison-7b28c803cdd2> – Назва з екрана. – Дата звернення: 15.05.2025.

45. SitePoint. Angular and RxJS: Adding a REST API Back End [Электронный ресурс]. – Режим доступа: <https://www.sitepoint.com/angular-rxjs-create-api-service-rest-backend/> – Назва з екрана. – Дата звернення: 15.05.2025.
46. Microsoft Learn. SQL Server security best practices [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/sql/relational-databases/security/sql-server-security-best-practices?view=sql-server-ver16> – Назва з екрана. – Дата звернення: 15.05.2025.
47. Microsoft Learn. Monitor and Tune for Performance - SQL Server [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/sql/relational-databases/performance/monitor-and-tune-for-performance?view=sql-server-ver16> – Назва з екрана. – Дата звернення: 15.05.2025.
48. Simple Talk. SQL Server performance tuning: Nine best practices [Электронный ресурс]. – Режим доступа: <https://www.red-gate.com/simple-talk/databases/sql-server/database-administration-sql-server/sql-server-performance-tuning-nine-best-practices/> – Назва з екрана. – Дата звернення: 15.05.2025.
49. Microsoft. Introduction to Performance – EF Core [Электронный ресурс] / Microsoft. – Режим доступа: <https://learn.microsoft.com/en-us/ef/core/performance/> – Назва з екрана. – Дата звернення: 15.05.2025
50. Microsoft. Efficient Querying – EF Core [Электронный ресурс] / Microsoft. – Режим доступа: <https://learn.microsoft.com/en-us/ef/core/performance/efficient-querying> – Назва з екрана. – Дата звернення: 15.05.2025.
51. Getachew, S. 10 Essential Best Practices for Using Entity Framework Core in .NET 8 [Электронный ресурс] / Solomon Getachew. – Режим доступа: <https://medium.com/@solomongetachew112/10-essential-best-practices-for-using-entity-framework-core-in-net-8-3274d6143992> – Назва з екрана. – Дата звернення: 15.05.2025.

ДОДАТКИ

Додаток А

Код створення таблиць у БД

```
USE ArgusDB;  
GO
```

```
CREATE TABLE Users (  
    Id          int NOT NULL,  
    FirstName   varchar(50) NOT NULL,  
    LastName    varchar(50) NOT NULL,  
    Email       varchar(100) NOT NULL UNIQUE,  
    CONSTRAINT PK_Users  
        PRIMARY KEY (Id)  
);  
GO
```

```
CREATE TABLE Roles (  
    Id          smallint NOT NULL,  
    Name        varchar(50) NOT NULL,  
    Description  varchar(255) NULL,  
    CONSTRAINT PK_Roles  
        PRIMARY KEY (Id)  
);  
GO
```

```
CREATE TABLE Privileges (  
    Id          int NOT NULL,  
    Name        varchar(50) NOT NULL,  
    Description  varchar(255) NULL,  
    CONSTRAINT PK_Privileges  
        PRIMARY KEY (Id)  
);  
GO
```

```
CREATE TABLE RolePrivileges (  
    RoleId      smallint NOT NULL,  
    PrivilegeId int NOT NULL,  
    CONSTRAINT PK_RolePrivileges  
        PRIMARY KEY (RoleId, PrivilegeId),  
    CONSTRAINT FK_RolePrivileges_RoleId FOREIGN KEY (RoleId)  
        REFERENCES Roles(Id),  
    CONSTRAINT FK_RolePrivileges_PrivilegeId FOREIGN KEY (PrivilegeId)  
        REFERENCES Privileges(Id)  
);  
GO
```

```
CREATE TABLE UserRoles (  
    UserId int NOT NULL,  
    RoleId smallint NOT NULL,  
    CONSTRAINT PK_UserRoles  
        PRIMARY KEY (UserId, RoleId),  
    CONSTRAINT FK_UserRoles_UserId FOREIGN KEY (UserId)  
        REFERENCES Users(Id),  
    CONSTRAINT FK_UserRoles_RoleId FOREIGN KEY (RoleId)  
        REFERENCES Roles(Id)  
);  
GO
```

```
CREATE TABLE ResourceCategories(  
    Id SMALLINT NOT NULL,
```

```

        Name VARCHAR(50) NOT NULL,
        CONSTRAINT PK_ResourceCategories
            PRIMARY KEY (Id)
    );
GO

CREATE TABLE ResourceTypes(
    Id SMALLINT IDENTITY(1,1) NOT NULL,
    Name VARCHAR(50) NOT NULL,
    CONSTRAINT PK_ResourceTypes
        PRIMARY KEY (Id)
);
GO

CREATE TABLE ResourceCategoryTypes(
    Id SMALLINT IDENTITY(1,1) NOT NULL,
    TypeId SMALLINT NOT NULL,
    CategoryId SMALLINT NOT NULL,
    CONSTRAINT PK_ResourceCategoryTypes
        PRIMARY KEY (Id),
    CONSTRAINT FK_ResourceCategoryTypes_TypeId FOREIGN KEY (TypeId)
        REFERENCES ResourceTypes(Id),
    CONSTRAINT FK_ResourceCategoryTypes_CategoryId FOREIGN KEY (CategoryId)
        REFERENCES ResourceCategories(Id)
);
GO

CREATE TABLE Resources(
    Id BIGINT IDENTITY(1,1) NOT NULL,
    Name VARCHAR(50) NOT NULL,
    ResourceTypeId SMALLINT NOT NULL,
    Position geography NULL,
    SerialNumber BIGINT NULL,
    CONSTRAINT PK_Resources
        PRIMARY KEY (Id),
    CONSTRAINT AK_SerialNumber
        UNIQUE(SerialNumber),
    CONSTRAINT FK_Resources_ResourceTypes FOREIGN KEY (ResourceTypeId)
        REFERENCES ResourceTypes(Id)
);
GO

CREATE TABLE LocationTypes(
    Id SMALLINT IDENTITY(1,1) NOT NULL,
    Name VARCHAR(50) NOT NULL,
    Description VARCHAR(255) NULL,
    CONSTRAINT PK_LocationTypes
        PRIMARY KEY (Id)
);
GO

CREATE TABLE Locations(
    Id BIGINT IDENTITY(1,1) NOT NULL,
    LocationTypeId SMALLINT NOT NULL,
    Position geography NULL,
    IsActive BIT NOT NULL DEFAULT 1,
    CONSTRAINT PK_Locations
        PRIMARY KEY (Id),
    CONSTRAINT FK_Locations_LocationTypeId FOREIGN KEY (LocationTypeId)
        REFERENCES LocationTypes(Id)
);
GO

```

```

CREATE TABLE LocationResources(
    Id BIGINT IDENTITY(1,1) NOT NULL,
    LocationId BIGINT NOT NULL,
    ResourceId BIGINT NOT NULL,
    Quantity INT NULL DEFAULT 1,
    CONSTRAINT PK_LocationResources
        PRIMARY KEY (Id),
    CONSTRAINT FK_LocationResources_LocationId FOREIGN KEY (LocationId)
        REFERENCES Locations,
    CONSTRAINT FK_LocationResources_ResourceId FOREIGN KEY (ResourceId)
        REFERENCES Resources
);
GO

CREATE TABLE MilitaryUnitStatuses(
    Id SMALLINT IDENTITY(1,1) NOT NULL,
    Name VARCHAR(50) NOT NULL,
    Description VARCHAR(255) NULL,
    CONSTRAINT PK_MilitaryUnitStatuses
        PRIMARY KEY (Id)
);
GO

CREATE TABLE MilitaryUnitTypes(
    Id SMALLINT IDENTITY(1,1) NOT NULL,
    Name VARCHAR(50) NOT NULL,
    CONSTRAINT PK_MilitaryUnitTypes
        PRIMARY KEY (Id)
);
GO

CREATE TABLE MilitaryUnits(
    Id INT IDENTITY(1,1) NOT NULL,
    Name VARCHAR(50) NOT NULL,
    Commander VARCHAR(100) NULL,
    ParentUnitId INT NOT NULL,
    UnitTypeId SMALLINT NOT NULL,
    StatusId SMALLINT NOT NULL,
    CONSTRAINT PK_MilitaryUnits
        PRIMARY KEY (Id),
    CONSTRAINT FK_MilitaryUnits_ParentUnitId FOREIGN KEY (ParentUnitId)
        REFERENCES MilitaryUnits(Id),
    CONSTRAINT FK_MilitaryUnits_UnitTypeId FOREIGN KEY (UnitTypeId)
        REFERENCES MilitaryUnitTypes(Id),
    CONSTRAINT FK_MilitaryUnits_StatusId FOREIGN KEY (StatusId)
        REFERENCES MilitaryUnitStatuses(Id)
);
GO

CREATE TABLE MilitaryUnitResources(
    Id BIGINT IDENTITY(1,1) NOT NULL,
    UnitId INT NOT NULL,
    ResourceId BIGINT NOT NULL,
    Quantity INT NULL DEFAULT 1,
    CONSTRAINT PK_MilitaryUnitResources
        PRIMARY KEY (Id),
    CONSTRAINT FK_MilitaryUnitResources_UnitId FOREIGN KEY (UnitId)
        REFERENCES MilitaryUnits(Id),
    CONSTRAINT FK_MilitaryUnitResources_ResourceId FOREIGN KEY (ResourceId)
        REFERENCES Resources(Id)
);
GO

```

```

CREATE TABLE MilitaryUnitResChangeLogs (
    Id          bigint IDENTITY(1,1) NOT NULL,
    LogDate     datetime NOT NULL,
    AuthorId    int NOT NULL,
    UnitId      int NOT NULL,
    CONSTRAINT PK_MilitaryUnitResChangeLogs
        PRIMARY KEY (Id),
    CONSTRAINT FK_MilitaryUnitResChangeLogs_UnitId FOREIGN KEY (UnitId)
        REFERENCES MilitaryUnits(Id),
    CONSTRAINT FK_MilitaryUnitResChangeLogs_AuthorId FOREIGN KEY (AuthorId)
        REFERENCES Users(Id)
);
GO

CREATE TABLE MilitaryUnitResChanges (
    ChangeLogId BIGINT NOT NULL,
    ResourceId   BIGINT NOT NULL,
    CONSTRAINT PK_MilitaryUnitResChanges
        PRIMARY KEY (ChangeLogId, ResourceId),
    CONSTRAINT FK_MilitaryUnitResChanges_ChangeLogId FOREIGN KEY (ChangeLogId)
        REFERENCES MilitaryUnitResChangeLogs(Id),
    CONSTRAINT FK_MilitaryUnitResChanges_ResourceId FOREIGN KEY (ResourceId)
        REFERENCES Resources(Id)
);
GO

CREATE TABLE MissionTypes(
    Id SMALLINT IDENTITY(1,1) NOT NULL,
    Name VARCHAR(50) NOT NULL,
    CONSTRAINT PK_MissionTypes
        PRIMARY KEY (Id)
);
GO

CREATE TABLE Missions(
    Id BIGINT IDENTITY(1,1) NOT NULL,
    Name VARCHAR(255) NOT NULL,
    Description TEXT NULL,
    TypeId SMALLINT NOT NULL,
    StartDate DATETIME NOT NULL,
    EndDate DATETIME NULL,
    Goals TEXT NULL,
    ParentMissionId BIGINT NOT NULL,
    CONSTRAINT PK_Missions
        PRIMARY KEY (Id),
    CONSTRAINT FK_Missions_ParentMissionId FOREIGN KEY (ParentMissionId)
        REFERENCES Missions(Id),
    CONSTRAINT FK_Missions_TypeId FOREIGN KEY (TypeId)
        REFERENCES MissionTypes(Id)
);
GO

CREATE TABLE MissionUnitResources(
    Id BIGINT IDENTITY(1,1) NOT NULL,
    MissionId BIGINT NOT NULL,
    UnitId INT NOT NULL,
    ResourceId BIGINT NOT NULL,
    InitialQuantity INT NOT NULL DEFAULT 1,
    FinalQuantity INT NULL,
    CONSTRAINT PK_MissionUnitResources
        PRIMARY KEY (Id),
    CONSTRAINT FK_MissionUnitResources_MissionId FOREIGN KEY (MissionId)
        REFERENCES Missions(Id),
    CONSTRAINT FK_MissionUnitResources_UnitId FOREIGN KEY (UnitId)

```

```

        REFERENCES MilitaryUnits(Id),
    CONSTRAINT FK_MissionUnitResources_ResourceId FOREIGN KEY (ResourceId)
        REFERENCES Resources(Id)
);
GO

CREATE TABLE ResourceInspections (
    Id          bigint IDENTITY(1,1) NOT NULL,
    InspectionDate date NOT NULL,
    AuthorId     int NOT NULL,
    LocationId    bigint NOT NULL,
    CONSTRAINT PK_ResourceInspections
        PRIMARY KEY (Id),
    CONSTRAINT FK_ResourceInspections_AuthorId FOREIGN KEY (AuthorId)
        REFERENCES Users(Id),
    CONSTRAINT FK_ResourceInspections_LocationId FOREIGN KEY (LocationId)
        REFERENCES Locations(Id)
);
GO

CREATE TABLE ResourceInspectionDetails (
    InspectionId bigint NOT NULL,
    ResourceId    bigint NOT NULL,
    OldQuantity   int NULL,
    NewQuantity   int NULL,
    CONSTRAINT PK_ResourceInspectionDetails
        PRIMARY KEY CLUSTERED (InspectionId, ResourceId),
    CONSTRAINT FK_ResourceInspectionDetails_InspectionId FOREIGN KEY (InspectionId)
        REFERENCES ResourceInspections(Id),
    CONSTRAINT FK_ResourceInspectionDetails_ResourceId FOREIGN KEY (ResourceId)
        REFERENCES Resources(Id)
);
GO

```

Клас налаштувань Identity IdentityExtensions.cs:

```
public static class IdentityExtensions
{
    public static IServiceCollection AddIdentityHandlersAndStores(this IServiceCollection
services)
    {
        services.AddIdentityApiEndpoints<AppUser>()
            .AddRoles<IdentityRole>()
            .AddEntityFrameworkStores<ArgusDbContext>();

        return services;
    }

    public static IServiceCollection ConfigureIdentityOptions(this IServiceCollection
services)
    {
        services.Configure<IdentityOptions>(options =>
        {
            options.Password.RequireDigit = true;
            options.Password.RequireLowercase = true;
            options.Password.RequireUppercase = true;
            options.User.RequireUniqueEmail = true;
        });

        return services;
    }

    // Auth = Authentication + Authorization
    public static IServiceCollection AddIdentityAuth(this IServiceCollection services,
IConfiguration config)
    {
        services.
            AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
            .AddJwtBearer(option =>
            {
                option.SaveToken = false;
                option.TokenValidationParameters = new TokenValidationParameters
                {
                    ValidateIssuerSigningKey = true,
                    IssuerSigningKey = new SymmetricSecurityKey(
                        Encoding.UTF8.GetBytes(config["AppSettings:JWTSecret"]!))
                    ,
                    ValidateIssuer = false,
                    ValidateAudience = false,
                    ValidateLifetime = true,
                    ClockSkew = TimeSpan.Zero
                };
            });

        services.AddAuthorization(options =>
        {
            options.FallbackPolicy = new AuthorizationPolicyBuilder()
                .AddAuthenticationSchemes(JwtBearerDefaults.AuthenticationScheme)
                .RequireAuthenticatedUser()
                .Build();
        });

        return services;
    }
}
```

```

    }

    public static WebApplication AddIdentityAuthMiddlewares(this WebApplication app)
    {
        app.UseAuthentication();
        app.UseAuthorization();

        return app;
    }
}

```

Метод аутентифікації зі створенням JWT:

```

[AllowAnonymous]
private static async Task<IResult> SignIn(
    UserManager<AppUser> userManager,
    [FromBody] LoginModel loginModel,
    IOptions<AppSettings> appSettings
)
{
    var user = await userManager.FindByEmailAsync(loginModel.Email);
    if (user != null && await userManager.CheckPasswordAsync(user, loginModel.Password))
    {
        var roles = await userManager.GetRolesAsync(user);
        var signInKey = new SymmetricSecurityKey(
            Encoding.UTF8.GetBytes(appSettings.Value.JWTSecret)
        );
        var defaultExpiresTime = 60;
        bool isExpiresTimeValid = int.TryParse(appSettings.Value.ExpiresIn, out int
expiresIn);

        ClaimsIdentity claims = new([
            new Claim("userID", user.Id.ToString()),
            new Claim(ClaimTypes.Role, roles.First()),
        ]);
        var tokenDescriptor = new SecurityTokenDescriptor
        {
            Subject = claims,
            Expires = DateTime.UtcNow.AddMinutes(isExpiresTimeValid ? expiresIn :
defaultExpiresTime),
            SigningCredentials = new SigningCredentials(
                signInKey,
                SecurityAlgorithms.HmacSha256Signature
            )
        };
        var tokenHandler = new JwtSecurityTokenHandler();
        var securityToken = tokenHandler.CreateToken(tokenDescriptor);
        var token = tokenHandler.WriteToken(securityToken);
        return Results.Ok(new { token });
    } else
    {
        return Results.BadRequest(new { messages = "Email or password is incorrect." });
    }
}

```

LocationsApiController.cs:

```
namespace Argus.Server.Controllers
{
    [Authorize(Roles = "Resource manager, Logistician")]
    [Route("api/locations")]
    [ApiController]
    public class LocationsApiController : ControllerBase
    {
        private readonly ILocationRepository _locationRepository;
        private readonly ILocationResourcesRepository _locationResourcesRepository;
        private readonly IResourceRepository _resourceRepository;
        private readonly IMapper _mapper;
        protected APIResponse _response;
        public LocationsApiController(IMapper mapper, ILocationRepository
locationRepository, ILocationResourcesRepository locationResourcesRepository,
IResourceRepository resourceRepository)
        {
            _mapper = mapper;
            _locationRepository = locationRepository;
            _locationResourcesRepository = locationResourcesRepository;
            _resourceRepository = resourceRepository;
            _response = new APIResponse()
            {
                StatusCode = HttpStatusCode.OK,
                IsSuccess = true
            };
        }

        // GET: api/locations
        [HttpGet]
        [ProducesResponseType(StatusCodes.Status200OK)]
        public async Task<ActionResult<APIResponse>> Get([FromQuery] DataSourceRequest
request)
        {
            var locations = await _locationRepository.GetAllAsync(request,
includeProperties: "LocationType");
            var locationsDto = _mapper.Map<List<LocationDto>>(locations);
            var totalCount = await _locationRepository.CountAsync(request);

            _response.Result = new DataSourceResult<LocationDto>()
            {
                Data = locationsDto,
                TotalCount = totalCount
            };
            _response.StatusCode = HttpStatusCode.OK;
            _response.IsSuccess = true;

            return Ok(_response);
        }

        // GET: api/locations/{locationId}
        [HttpGet("{locationId}")]
        [ProducesResponseType(StatusCodes.Status200OK)]
        [ProducesResponseType(StatusCodes.Status404NotFound)]
        public async Task<ActionResult> Get(long locationId)
        {
            var location = await _locationRepository.GetAsync(l => l.Id == locationId);
            var locationDto = _mapper.Map<LocationDto>(location);
        }
    }
}
```

```

        _response.Result = locationDto;
        return Ok(_response);
    }

    // GET: api/locations/{locationId}/resources
    [HttpGet("{locationId}/resources")]
    [ProducesResponseType(StatusCodes.Status200OK)]
    public async Task<ActionResult<APIResponse>> GetResourcesForLocation([FromQuery]
DataSourceRequest request, long locationId)
    {
        IEnumerable<LocationResourceDto> locationResources = await
_locationResourcesRepository.GetLocationResources(request, locationId);
        int totalCount = await
_locationResourcesRepository.CountByLocationIdAsync(request, locationId);

        _response.Result = new DataSourceResult<LocationResourceDto>
        {
            Data = locationResources.ToList(),
            TotalCount = totalCount
        };
        _response.StatusCode = HttpStatusCode.OK;

        return Ok(_response);
    }

    // GET: api/locations/{locationId}/resources/add
    [HttpGet("{locationId}/resources/add")]
    [ProducesResponseType(StatusCodes.Status200OK)]
    public async Task<ActionResult<APIResponse>>
GetResourcesForAddingToLocation([FromQuery] DataSourceRequest request, long locationId)
    {
        IEnumerable<Resource> resources = await
_resourceRepository.GetResourcesForAddingToLocation(request, locationId);
        var resourcesDto = _mapper.Map<List<ResourceDto>>(resources);
        int totalCount = await
_resourceRepository.CountResourcesForAddingToLocation(request, locationId);
        _response.Result = new DataSourceResult<ResourceDto>
        {
            Data = resourcesDto,
            TotalCount = totalCount
        };
        _response.StatusCode = HttpStatusCode.OK;
        return Ok(_response);
    }

    // POST api/locations/
    [HttpPost]
    [ProducesResponseType(StatusCodes.Status200OK)]
    [ProducesResponseType(StatusCodes.Status400BadRequest)]
    [ProducesResponseType(StatusCodes.Status500InternalServerError)]
    public async Task<ActionResult<APIResponse>> CreateLocation([FromBody]
LocationCreateDto locationDto)
    {
        try
        {
            if (locationDto == null)
            {
                _response.StatusCode = HttpStatusCode.BadRequest;
                _response.IsSuccess = false;
                _response.ErrorMessages = new List<string> { "Location data is null"
};
                return BadRequest(_response);
            }
            var location = _mapper.Map<Location>(locationDto);

```

```

        await _locationRepository.CreateAsync(location);

        _response.Result = location;
        _response.StatusCode = HttpStatusCode.Created;
        return CreatedAtAction(nameof(Get), new { id = location.Id }, _response);
    }
    catch (Exception ex)
    {
        _response.IsSuccess = false;
        _response.ErrorMessages = [ex.ToString()];
        return StatusCode(500, _response);
    }
}

[Authorize(Roles = "Resource manager")]
// POST api/locations/{locationId}/resources
[HttpPost("{locationId}/resources")]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[ProducesResponseType(StatusCodes.Status500InternalServerError)]
public async Task<ActionResult<APIResponse>> AddResources(long locationId,
[FromBody] LocationResourceDto[] resources)
{
    try
    {
        await _locationResourcesRepository.AddResourcesAsync(locationId,
resources);
    }
    catch (Exception ex)
    {
        _response.StatusCode = HttpStatusCode.InternalServerError;
        _response.IsSuccess = false;
        _response.ErrorMessages = [ex.Message];

        return BadRequest(_response);
    }

    return NoContent();
}

// DELETE api/locations/
[HttpDelete]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
[ProducesResponseType(StatusCodes.Status500InternalServerError)]
public async Task<ActionResult> DeleteLocations([FromBody] long[] ids)
{
    var errors = new List<string>();

    foreach (var id in ids)
    {
        var location = await _locationRepository.GetAsync(r => r.Id == id);

        if (location != null)
        {
            await _locationRepository.RemoveAsync(location);
        } else
        {
            errors.Add($"Location with id \"{id}\" not found.");
        }
    }

    if (errors.Count > 0)
    {

```

```

        if (errors.Count == ids.Length)
        {
            _response.StatusCode = HttpStatusCode.NotFound;
            _response.IsSuccess = false;
            _response.ErrorMessages = errors;
        } else
        {
            _response.StatusCode = HttpStatusCode.MultiStatus;
            _response.IsSuccess = true;
        }

        return NotFound(_response);
    }

    return NoContent();
}

// DELETE api/locations/{locationId}/resources
[Authorize(Roles = "Resource manager")]
[HttpDelete("{locationId}/resources")]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
[ProducesResponseType(StatusCodes.Status500InternalServerError)]
public async Task<ActionResult<APIResponse>> DeleteResourcesFromLocation(long
locationId, [FromBody] long[] resourceIds)
{
    try
    {
        await _locationResourcesRepository.RemoveByLocationIdAsync(locationId,
resourceIds);
    }
    catch (Exception ex)
    {
        var isNotFound = ex is NotFoundException;
        var status = isNotFound ? HttpStatusCode.NotFound :
HttpStatusCode.InternalServerError;

        _response.StatusCode = status;
        _response.IsSuccess = false;
        _response.ErrorMessages = [ex.Message];

        return isNotFound
            ? NotFound(_response)
            : BadRequest(_response);
    }

    return NoContent();
}
}
}

```

LocationTypesAPIController.cs:

```

namespace Argus.Server.Controllers
{
    [Authorize(Roles = "Resource manager, Logistician")]
    [Route("api/locationTypes")]
    [ApiController]
    public class LocationTypesAPIController : ControllerBase
    {
        private readonly ILocationTypesRepository _locationTypesRepository;
        private readonly IMapper _mapper;
        private APIResponse _response;
    }
}

```

```

        public LocationTypesApiController(ILocationTypesRepository
locationTypesRepository, IMapper mapper)
        {
            _locationTypesRepository = locationTypesRepository;
            _mapper = mapper;
            _response = new APIResponse()
            {
                StatusCode = HttpStatusCode.OK,
                IsSuccess = true
            };
        }

        [HttpGet]
        public async Task<ActionResult<APIResponse>> Get()
        {
            try
            {
                var locationTypes = await _locationTypesRepository.GetAllAsync();
                var loationTypesDto = _mapper.Map<List<LocationTypeDto>>(locationTypes);

                _response.Result = locationTypes;
                return Ok(_response);
            }
            catch (Exception ex)
            {
                _response.StatusCode = HttpStatusCode.InternalServerError;
                _response.ErrorMessages = [ex.Message];
                return BadRequest(_response);
            }
        }
    }
}

```

ResourcesApiController.cs:

```

namespace Argus.Server.Controllers
{
    [Authorize(Roles = "Resource manager, Logistician")]
    [ApiController]
    [Route("api/Resources")]
    public class ResourcesApiController : ControllerBase
    {
        private readonly IResourceRepository _resourceRepository;
        private readonly ILocationRepository _locationRepository;
        private readonly ILocationResourcesRepository _locationResourcesRepository;
        private readonly IMapper _mapper;
        protected APIResponse _response;

        public ResourcesApiController(IMapper mapper, IResourceRepository
resourceRepository, ILocationResourcesRepository locationResourcesRepository,
ILocationRepository locationRepository)
        {
            _mapper = mapper;
            _resourceRepository = resourceRepository;
            _locationResourcesRepository = locationResourcesRepository;
            _locationRepository = locationRepository;
            _response = new APIResponse();
        }

        // GET: api/resources
        [HttpGet]
        [ProducesResponseType(StatusCodes.Status200OK)]
        public async Task<ActionResult<APIResponse>> Get([FromQuery] DataSourceRequest
request)

```

```

    {
        IEnumerable<Resource> resources = await
        _resourceRepository.GetAllAsync(request, includeProperties: "ResourceType");
        var resourcesDto = _mapper.Map<List<ResourceDto>>>(resources);
        int totalCount = await _resourceRepository.CountAsync(request);

        _response.Result = new DataSourceResult<ResourceDto>
        {
            Data = resourcesDto,
            TotalCount = totalCount
        };
        _response.StatusCode = HttpStatusCode.OK;

        return Ok(_response);
    }

    // GET: api/resources/{resourceId}
    [HttpGet("{resourceId}")]
    [ProducesResponseType(StatusCodes.Status200OK)]
    [ProducesResponseType(StatusCodes.Status400BadRequest)]
    [ProducesResponseType(StatusCodes.Status500InternalServerError)]
    public async Task<ActionResult<APIResponse>> Get(long resourceId)
    {
        try
        {
            var resource = await _resourceRepository.GetResourceInfo(resourceId);

            if (resource == null)
            {
                throw new NotFoundException($"Resource with ID {resourceId} not
found!");
            }

            _response.Result = resource;
        }
        catch (Exception ex)
        {
            var isNotFound = ex is NotFoundException;
            var status = isNotFound ? HttpStatusCode.NotFound :
HttpStatusCode.InternalServerError;

            _response.StatusCode = status;
            _response.IsSuccess = false;
            _response.ErrorMessages = [ex.Message];

            return isNotFound
                ? NotFound(_response)
                : BadRequest(_response);
        }

        return Ok(_response);
    }

    // GET: api/resources/{resourceId}/locations
    [HttpGet("{resourceId}/locations")]
    [ProducesResponseType(StatusCodes.Status200OK)]
    public async Task<ActionResult<APIResponse>> GetResourceLocations(long
resourceId)
    {
        IEnumerable<Location> locations = await
        _locationResourcesRepository.GetResourceLocations(resourceId);
        var locationsDto = _mapper.Map<List<LocationDto>>>(locations);

        _response.Result = locationsDto;
    }

```

```

        return Ok(_response);
    }

    // GET: api/resources/resource/{resourceId}/locations/markers-info
    [HttpGet("resource/{resourceId}/locations/markers-info")]
    [ProducesResponseType(StatusCodes.Status200OK)]
    public async Task<ActionResult<APIResponse>>
GetMarkersInfoForResourceLocations(long resourceId)
    {
        _response.Result = await _locationRepository.GetMarkersInfoForLocations(l =>
l.LocationResources.Any(lr => lr.ResourceId == resourceId));
        return Ok(_response);
    }

    // POST: api/resources/resource/
    [Authorize(Roles = "Resource manager")]
    [HttpPost]
    [ProducesResponseType(StatusCodes.Status200OK)]
    [ProducesResponseType(StatusCodes.Status400BadRequest)]
    [ProducesResponseType(StatusCodes.Status500InternalServerError)]
    public async Task<ActionResult<APIResponse>> CreateResource([FromBody]
ResourceCreatedDto resourceDto)
    {
        try
        {
            if (resourceDto == null)
            {
                _response.StatusCode = HttpStatusCode.BadRequest;
                _response.IsSuccess = false;
                _response.ErrorMessages = new List<string> { "Resource data is null"
};

                return BadRequest(_response);
            }
            var resource = _mapper.Map<Resource>(resourceDto);

            await _resourceRepository.CreateAsync(resource);

            _response.Result = resource;
            _response.StatusCode = HttpStatusCode.Created;
            return CreatedAtAction(nameof(Get), new { id = resource.Id }, _response);
        }
        catch (Exception ex)
        {
            _response.IsSuccess = false;
            _response.ErrorMessages = new List<string>() { ex.ToString() };
            return StatusCode(500, _response);
        }
    }

    [Authorize(Roles = "Resource manager")]
    [HttpPut("{id:long}", Name = "UpdateResource")]
    [ProducesResponseType(StatusCodes.Status204NoContent)]
    [ProducesResponseType(StatusCodes.Status400BadRequest)]
    [ProducesResponseType(StatusCodes.Status404NotFound)]
    public async Task<ActionResult<APIResponse>> UpdateResource(long id, [FromBody]
ResourceUpdatedDto resourceDto)
    {
        if (resourceDto == null || id != resourceDto.Id)
        {
            _response.StatusCode = HttpStatusCode.BadRequest;
            _response.IsSuccess = false;
            _response.ErrorMessages = new List<string> { "Resource data is null or ID
mismatch" };

```

```

        return BadRequest(_response);
    }
    var resource = await _resourceRepository.GetAsync(r => r.Id == id);
    if (resource == null)
    {
        _response.StatusCode = HttpStatusCode.NotFound;
        _response.IsSuccess = false;
        _response.ErrorMessages = new List<string> { "Resource not found" };
        return NotFound(_response);
    }

    resource.Name = resourceDto.Name;
    resource.ResourceTypeId = resourceDto.ResourceTypeId;
    resource.SerialNumber = resourceDto.SerialNumber;

    await _resourceRepository.UpdateAsync(resource);

    _response.Result = resource;
    _response.StatusCode = HttpStatusCode.OK;
    return Ok(_response);
}

[Authorize(Roles = "Resource manager")]
[HttpDelete(Name = "DeleteResources")]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
public async Task<ActionResult<APIResponse>> DeleteResources([FromBody] long[]
ids)
{
    var errors = new List<string>();

    foreach (var id in ids)
    {
        var resource = await _resourceRepository.GetAsync(r => r.Id == id);

        if (resource != null)
        {
            await _resourceRepository.RemoveAsync(resource);
        }
        else
        {
            errors.Add($"Resource with id \"{id}\" not found.");
        }
    }

    if (errors.Count > 0)
    {
        if (errors.Count == ids.Length)
        {
            _response.StatusCode = HttpStatusCode.NotFound;
            _response.IsSuccess = false;
            _response.ErrorMessages = errors;
        }
        else
        {
            _response.StatusCode = HttpStatusCode.MultiStatus;
            _response.IsSuccess = true;
        }

        return NotFound(_response);
    }

    return NoContent();
}

```

```

    }
}

```

ResourceTypesApiController.cs:

```

namespace Argus.Server.Controllers
{
    [Authorize(Roles = "Resource manager")]
    [Route("api/resourceTypes")]
    [ApiController]
    public class ResourceTypesApiController : ControllerBase
    {
        private readonly IResourceTypeRepository _resourceTypesRepository;
        private readonly IMapper _mapper;
        protected APIResponse _response;

        public ResourceTypesApiController(IResourceTypeRepository
resourceTypesRepository, IMapper mapper)
        {
            _resourceTypesRepository = resourceTypesRepository;
            _mapper = mapper;
            _response = new APIResponse();
        }

        // GET: api/resourceTypes
        [HttpGet]
        public async Task<ActionResult<APIResponse>> GetResourceTypes()
        {
            var resourceTypes = await _resourceTypesRepository.GetAllAsync();
            var resourceTypesDto = _mapper.Map<List<ResourceTypeDto>>(resourceTypes);

            _response.Result = resourceTypesDto;
            _response.StatusCode = HttpStatusCode.OK;

            return Ok(_response);
        }
    }
}

```

RolesApiController.cs:

```

namespace Argus.Server.Controllers
{
    [Authorize(Roles = "Admin")]
    [Route("api/roles")]
    [ApiController]
    public class RolesApiController : ControllerBase
    {
        private readonly IRoleRepository _roleRepository;
        private readonly IMapper _mapper;

        public RolesApiController(IRoleRepository roleRepository, IMapper mapper)
        {
            _roleRepository = roleRepository;
            _mapper = mapper;
        }

        [AllowAnonymous]
        // GET: api/roles
        [HttpGet]
        [ProducesResponseType(StatusCodes.Status200OK)]
        public async Task<ActionResult> GetAll()
        {

```

```

        try
        {
            var roles = await _roleRepository.GetAllAsync();
            var rolesDto = _mapper.Map<List<RoleDto>>(roles);
            return Ok(rolesDto);
        }
        catch (Exception ex)
        {
            return BadRequest(ex.Message);
        }
    }
}

```

UsersAPIController.cs:

```

namespace Argus.Server.Controllers
{
    [Authorize(Roles = "Admin")]
    [Route("api/users")]
    [ApiController]
    public class UsersAPIController : ControllerBase
    {
        private readonly IUserRepository _userRepository;
        private readonly UserManager<AppUser> _userManager;
        private readonly IMapper _mapper;
        private APIResponse _response;

        public UsersAPIController(IUserRepository userRepository, UserManager<AppUser>
userManager, IMapper mapper)
        {
            _userRepository = userRepository;
            _userManager = userManager;
            _mapper = mapper;
            _response = new APIResponse()
            {
                StatusCode = HttpStatusCode.OK,
                IsSuccess = true
            };
        }

        [HttpGet]
        public async Task<ActionResult<UserDto>> GetAll([FromQuery] DataSourceRequest
request)
        {
            try
            {
                IEnumerable<UserDto> usersDto = await
_userRepository.GetAllAsync(request);
                var totalCount = await _userRepository.CountAsync();

                _response.Result = new DataSourceResult<UserDto>
                {
                    Data = usersDto.ToList(),
                    TotalCount = totalCount
                };

                return Ok(_response);
            }
            catch (Exception ex)
            {
                _response.IsSuccess = false;
            }
        }
    }
}

```

```
        _response.ErrorMessage = [ex.Message];  
        return BadRequest(_response);  
    }  
}  
}
```

Розширений клас моделі IdentityUser, AppUser.cs:

```
public partial class AppUser : IdentityUser
{
    [PersonalData]
    public string FirstName { get; set; } = null!;
    [PersonalData]
    public string LastName { get; set; } = null!;
    public virtual ICollection<AppUserRole> UserRoles { get; set; } = new
List<AppUserRole>();
    public virtual ICollection<MilitaryUnitResChangeLog> MilitaryUnitResChangeLogs { get;
set; } = new List<MilitaryUnitResChangeLog>();
    public virtual ICollection<ResourceInspection> ResourceInspections { get; set; } =
new List<ResourceInspection>();
}
```

```
public partial class ArgusDbContext :
    IdentityDbContext<
        AppUser,
        IdentityRole,
        string,
        IdentityUserClaim<string>,
        AppUserRole,
        IdentityUserLogin<string>,
        IdentityRoleClaim<string>,
        IdentityUserToken<string>>
    {
        public ArgusDbContext()
        {
        }

        public ArgusDbContext(DbContextOptions<ArgusDbContext> options)
            : base(options)
        {
        }

        public virtual DbSet<Location> Locations { get; set; }
        public virtual DbSet<LocationResource> LocationResources { get; set; }
        public virtual DbSet<LocationType> LocationTypes { get; set; }
        public virtual DbSet<MilitaryUnit> MilitaryUnits { get; set; }
        public virtual DbSet<MilitaryUnitResChangeLog> MilitaryUnitResChangeLogs { get; set; }
    }

    public virtual DbSet<MilitaryUnitResource> MilitaryUnitResources { get; set; }
    public virtual DbSet<MilitaryUnitStatus> MilitaryUnitStatuses { get; set; }
    public virtual DbSet<MilitaryUnitType> MilitaryUnitTypes { get; set; }
    public virtual DbSet<Mission> Missions { get; set; }
    public virtual DbSet<MissionType> MissionTypes { get; set; }
    public virtual DbSet<MissionUnitResource> MissionUnitResources { get; set; }
    public virtual DbSet<Resource> Resources { get; set; }
    public virtual DbSet<ResourceCategory> ResourceCategories { get; set; }
    public virtual DbSet<ResourceCategoryType> ResourceCategoryTypes { get; set; }
    public virtual DbSet<ResourceInspection> ResourceInspections { get; set; }
    public virtual DbSet<ResourceInspectionDetail> ResourceInspectionDetails { get; set; }
}

    public virtual DbSet<ResourceType> ResourceTypes { get; set; }
    public virtual DbSet<AppUser> AppUsers { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
```

```

{
    base.OnModelCreating(modelBuilder);

    modelBuilder.Entity<Location>(entity =>
    {
        entity.Property(e => e.IsActive).HasDefaultValue(true);

        entity.HasOne(d => d.LocationType).WithMany(p => p.Locations)
            .HasForeignKey(d => d.LocationTypeId)
            .OnDelete(DeleteBehavior.ClientSetNull)
            .HasConstraintName("FK_Locations_LocationTypeId");
    });

    modelBuilder.Entity<LocationResource>(entity =>
    {
        entity.Property(e => e.Quantity).HasDefaultValue(1);

        entity.HasOne(d => d.Location).WithMany(p => p.LocationResources)
            .HasForeignKey(d => d.LocationId)
            .OnDelete(DeleteBehavior.Cascade)
            .HasConstraintName("FK_LocationResources_LocationId");

        entity.HasOne(d => d.Resource).WithMany(p => p.LocationResources)
            .HasForeignKey(d => d.ResourceId)
            .OnDelete(DeleteBehavior.Cascade)
            .HasConstraintName("FK_LocationResources_ResourceId");
    });

    modelBuilder.Entity<LocationType>(entity =>
    {
        entity.Property(e => e.Description)
            .HasMaxLength(255)
            .IsUnicode(false);
        entity.Property(e => e.Name)
            .HasMaxLength(50)
            .IsUnicode(false);
    });

    modelBuilder.Entity<MilitaryUnit>(entity =>
    {
        entity.Property(e => e.Commander)
            .HasMaxLength(100)
            .IsUnicode(false);
        entity.Property(e => e.Name)
            .HasMaxLength(50)
            .IsUnicode(false);

        entity.HasOne(d => d.ParentUnit).WithMany(p => p.InverseParentUnit)
            .HasForeignKey(d => d.ParentUnitId)
            .OnDelete(DeleteBehavior.ClientSetNull)
            .HasConstraintName("FK_MilitaryUnits_ParentUnitId");

        entity.HasOne(d => d.Status).WithMany(p => p.MilitaryUnits)
            .HasForeignKey(d => d.StatusId)
            .OnDelete(DeleteBehavior.ClientSetNull)
            .HasConstraintName("FK_MilitaryUnits_StatusId");

        entity.HasOne(d => d.UnitType).WithMany(p => p.MilitaryUnits)
            .HasForeignKey(d => d.UnitTypeId)
            .OnDelete(DeleteBehavior.ClientSetNull)
            .HasConstraintName("FK_MilitaryUnits_UnitTypeId");
    });

    modelBuilder.Entity<MilitaryUnitResChangeLog>(entity =>

```

```

{
    entity.Property(e => e.LogDate).HasColumnType("datetime");

    entity.HasOne(d => d.Author).WithMany(p => p.MilitaryUnitResChangeLogs)
        .HasForeignKey(d => d.AuthorId)
        .OnDelete(DeleteBehavior.ClientSetNull)
        .HasConstraintName("FK_MilitaryUnitResChangeLogs_AuthorId");

    entity.HasOne(d => d.Unit).WithMany(p => p.MilitaryUnitResChangeLogs)
        .HasForeignKey(d => d.UnitId)
        .OnDelete(DeleteBehavior.ClientSetNull)
        .HasConstraintName("FK_MilitaryUnitResChangeLogs_UnitId");

    entity.HasMany(d => d.Resources).WithMany(p => p.ChangeLogs)
        .UsingEntity<Dictionary<string, object>>(
            "MilitaryUnitResChange",
            r => r.HasOne<Resource>().WithMany()
                .HasForeignKey("ResourceId")
                .OnDelete(DeleteBehavior.ClientSetNull)
                .HasConstraintName("FK_MilitaryUnitResChanges_ResourceId"),
            l => l.HasOne<MilitaryUnitResChangeLog>().WithMany()
                .HasForeignKey("ChangeLogId")
                .OnDelete(DeleteBehavior.ClientSetNull)
                .HasConstraintName("FK_MilitaryUnitResChanges_ChangeLogId"),
            j =>
            {
                j.HasKey("ChangeLogId", "ResourceId");
                j.ToTable("MilitaryUnitResChanges");
            }
        );
});

modelBuilder.Entity<MilitaryUnitResource>(entity =>
{
    entity.Property(e => e.Quantity).HasDefaultValue(1);

    entity.HasOne(d => d.Resource).WithMany(p => p.MilitaryUnitResources)
        .HasForeignKey(d => d.ResourceId)
        .OnDelete(DeleteBehavior.ClientSetNull)
        .HasConstraintName("FK_MilitaryUnitResources_ResourceId");

    entity.HasOne(d => d.Unit).WithMany(p => p.MilitaryUnitResources)
        .HasForeignKey(d => d.UnitId)
        .OnDelete(DeleteBehavior.ClientSetNull)
        .HasConstraintName("FK_MilitaryUnitResources_UnitId");
});

modelBuilder.Entity<MilitaryUnitStatus>(entity =>
{
    entity.Property(e => e.Description)
        .HasMaxLength(255)
        .IsUnicode(false);
    entity.Property(e => e.Name)
        .HasMaxLength(50)
        .IsUnicode(false);
});

modelBuilder.Entity<MilitaryUnitType>(entity =>
{
    entity.Property(e => e.Name)
        .HasMaxLength(50)
        .IsUnicode(false);
});

modelBuilder.Entity<Mission>(entity =>

```

```

{
    entity.Property(e => e.Description).HasColumnType("text");
    entity.Property(e => e.EndDate).HasColumnType("datetime");
    entity.Property(e => e.Goals).HasColumnType("text");
    entity.Property(e => e.Name)
        .HasMaxLength(255)
        .IsUnicode(false);
    entity.Property(e => e.StartDate).HasColumnType("datetime");

    entity.HasOne(d => d.ParentMission).WithMany(p => p.InverseParentMission)
        .HasForeignKey(d => d.ParentMissionId)
        .OnDelete(DeleteBehavior.ClientSetNull)
        .HasConstraintName("FK_Missions_ParentMissionId");

    entity.HasOne(d => d.Type).WithMany(p => p.Missions)
        .HasForeignKey(d => d.TypeId)
        .OnDelete(DeleteBehavior.ClientSetNull)
        .HasConstraintName("FK_Missions_TypeId");
});

modelBuilder.Entity<MissionType>(entity =>
{
    entity.Property(e => e.Name)
        .HasMaxLength(50)
        .IsUnicode(false);
});

modelBuilder.Entity<MissionUnitResource>(entity =>
{
    entity.Property(e => e.InitialQuantity).HasDefaultValue(1);

    entity.HasOne(d => d.Mission).WithMany(p => p.MissionUnitResources)
        .HasForeignKey(d => d.MissionId)
        .OnDelete(DeleteBehavior.ClientSetNull)
        .HasConstraintName("FK_MissionUnitResources_MissionId");

    entity.HasOne(d => d.Resource).WithMany(p => p.MissionUnitResources)
        .HasForeignKey(d => d.ResourceId)
        .OnDelete(DeleteBehavior.ClientSetNull)
        .HasConstraintName("FK_MissionUnitResources_ResourceId");

    entity.HasOne(d => d.Unit).WithMany(p => p.MissionUnitResources)
        .HasForeignKey(d => d.UnitId)
        .OnDelete(DeleteBehavior.ClientSetNull)
        .HasConstraintName("FK_MissionUnitResources_UnitId");
});

modelBuilder.Entity<Resource>(entity =>
{
    entity.HasIndex(e => e.SerialNumber, "AK_SerialNumber").IsUnique();

    entity.Property(e => e.Name)
        .HasMaxLength(50)
        .IsUnicode(false);

    entity.HasOne(d => d.ResourceType).WithMany(p => p.Resources)
        .HasForeignKey(d => d.ResourceTypeId)
        .OnDelete(DeleteBehavior.ClientSetNull)
        .HasConstraintName("FK_Resources_ResourceTypes");
});

modelBuilder.Entity<ResourceCategory>(entity =>
{
    entity.Property(e => e.Id).ValueGeneratedNever();
}

```

```

        entity.Property(e => e.Name)
            .HasMaxLength(50)
            .IsUnicode(false);
        entity.Property(e => e.IconPath)
            .HasMaxLength(255)
            .IsUnicode(false);
    });

    modelBuilder.Entity<ResourceCategoryType>(entity =>
    {
        entity.HasOne(d => d.Category).WithMany(p => p.ResourceCategoryTypes)
            .HasForeignKey(d => d.CategoryId)
            .OnDelete(DeleteBehavior.ClientSetNull)
            .HasConstraintName("FK_ResourceCategoryTypes_CategoryId");

        entity.HasOne(d => d.Type).WithMany(p => p.ResourceCategoryTypes)
            .HasForeignKey(d => d.TypeId)
            .OnDelete(DeleteBehavior.ClientSetNull)
            .HasConstraintName("FK_ResourceCategoryTypes_TypeId");
    });

    modelBuilder.Entity<ResourceInspection>(entity =>
    {
        entity.HasOne(d => d.Author).WithMany(p => p.ResourceInspections)
            .HasForeignKey(d => d.AuthorId)
            .OnDelete(DeleteBehavior.ClientSetNull)
            .HasConstraintName("FK_ResourceInspections_AuthorId");

        entity.HasOne(d => d.Location).WithMany(p => p.ResourceInspections)
            .HasForeignKey(d => d.LocationId)
            .OnDelete(DeleteBehavior.ClientSetNull)
            .HasConstraintName("FK_ResourceInspections_LocationId");
    });

    modelBuilder.Entity<ResourceInspectionDetail>(entity =>
    {
        entity.HasKey(e => new { e.InspectionId, e.ResourceId });

        entity.HasOne(d => d.Inspection).WithMany(p => p.ResourceInspectionDetails)
            .HasForeignKey(d => d.InspectionId)
            .OnDelete(DeleteBehavior.ClientSetNull)
            .HasConstraintName("FK_ResourceInspectionDetails_InspectionId");

        entity.HasOne(d => d.Resource).WithMany(p => p.ResourceInspectionDetails)
            .HasForeignKey(d => d.ResourceId)
            .OnDelete(DeleteBehavior.ClientSetNull)
            .HasConstraintName("FK_ResourceInspectionDetails_ResourceId");
    });

    modelBuilder.Entity<ResourceType>(entity =>
    {
        entity.Property(e => e.Name)
            .HasMaxLength(50)
            .IsUnicode(false);
    });

    modelBuilder.Entity<AppUser>(entity =>
    {
        entity.Property(e => e.FirstName)
            .HasMaxLength(50)
            .IsUnicode(false);
        entity.Property(e => e.LastName)
            .HasMaxLength(50)
            .IsUnicode(false);
    });

```

```

        entity.HasMany(u => u.UserRoles)
            .WithOne()
            .HasForeignKey(ur => ur.UserId)
            .IsRequired();
    });

    modelBuilder.Entity<AppUserRole>(b =>
    {
        b.HasKey(ar => new { ar.UserId, ar.RoleId });

        b
            .HasOne(ar => ar.User)
            .WithMany(u => u.UserRoles)
            .HasForeignKey(ar => ar.UserId)
            .IsRequired();

        b
            .HasOne(ar => ar.Role)
            .WithMany()
            .HasForeignKey(ar => ar.RoleId)
            .IsRequired();
    });

    OnModelCreatingPartial(modelBuilder);
}

partial void OnModelCreatingPartial(ModelBuilder modelBuilder);
}

```

Додаток Д

Інтерфейс системи

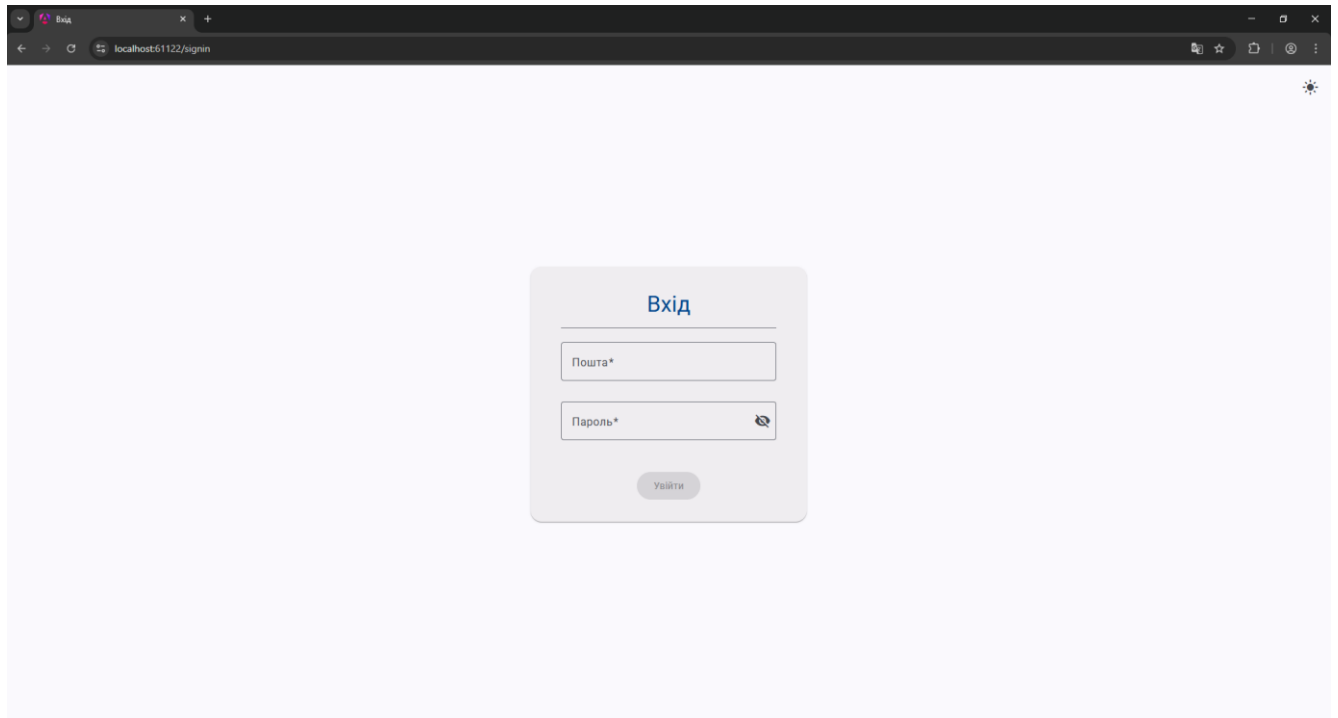


Рисунок Д-1 – Сторінка входу

Джерело: розроблено автором

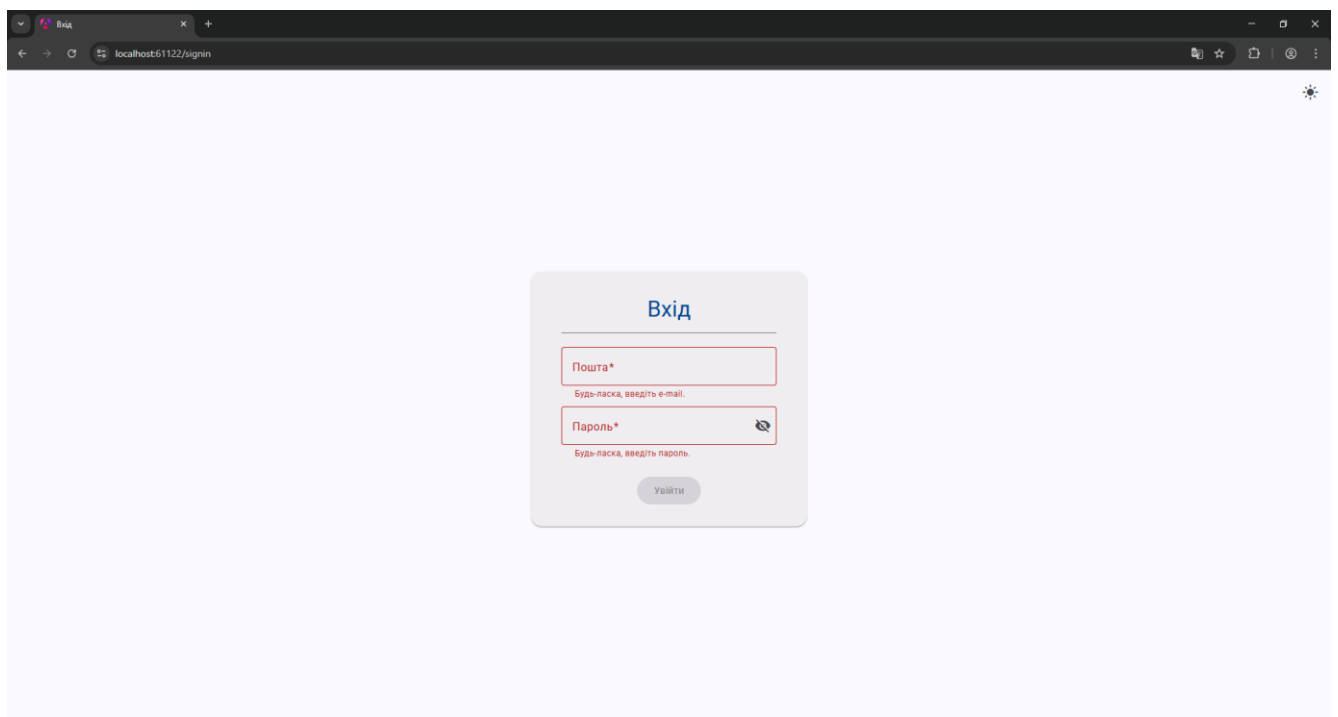


Рисунок Д-2 – Сторінка входу з помилковими даними

Джерело: розроблено автором

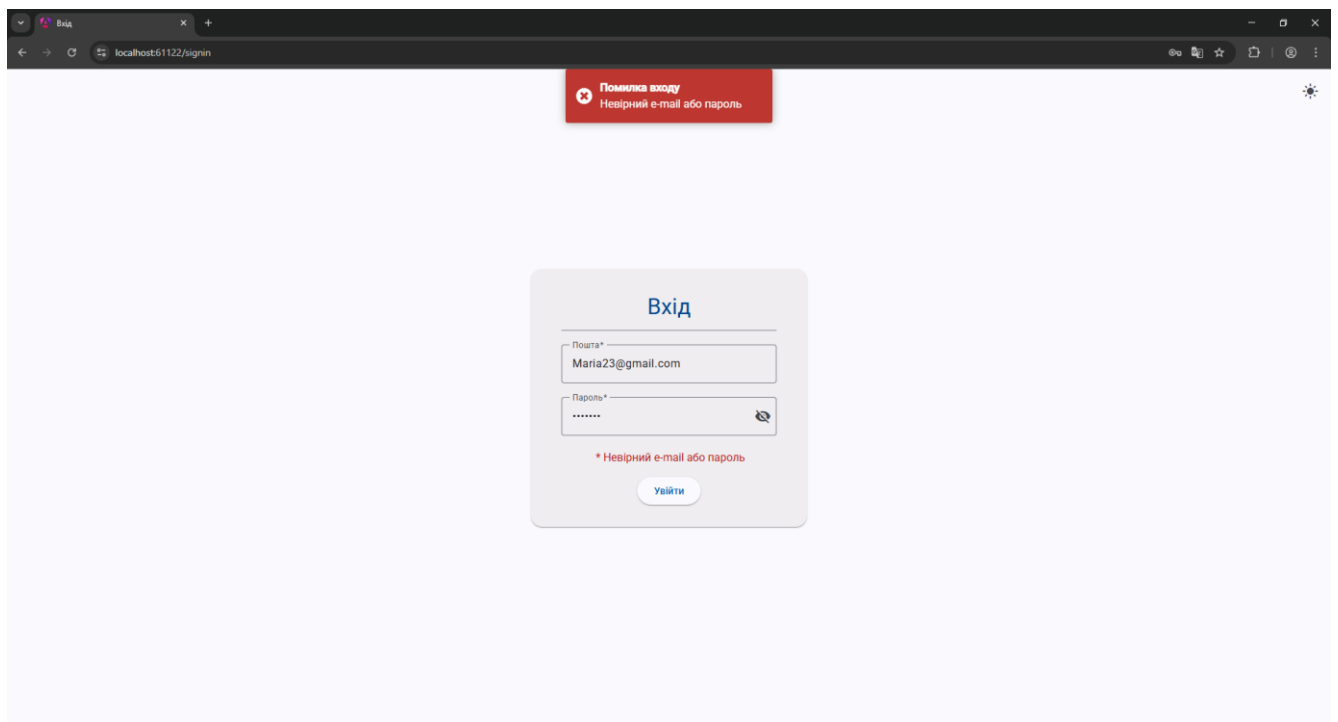


Рисунок Д-3 – Сторінка входу з помилкою про неправильний пароль чи логін

Джерело: розроблено автором

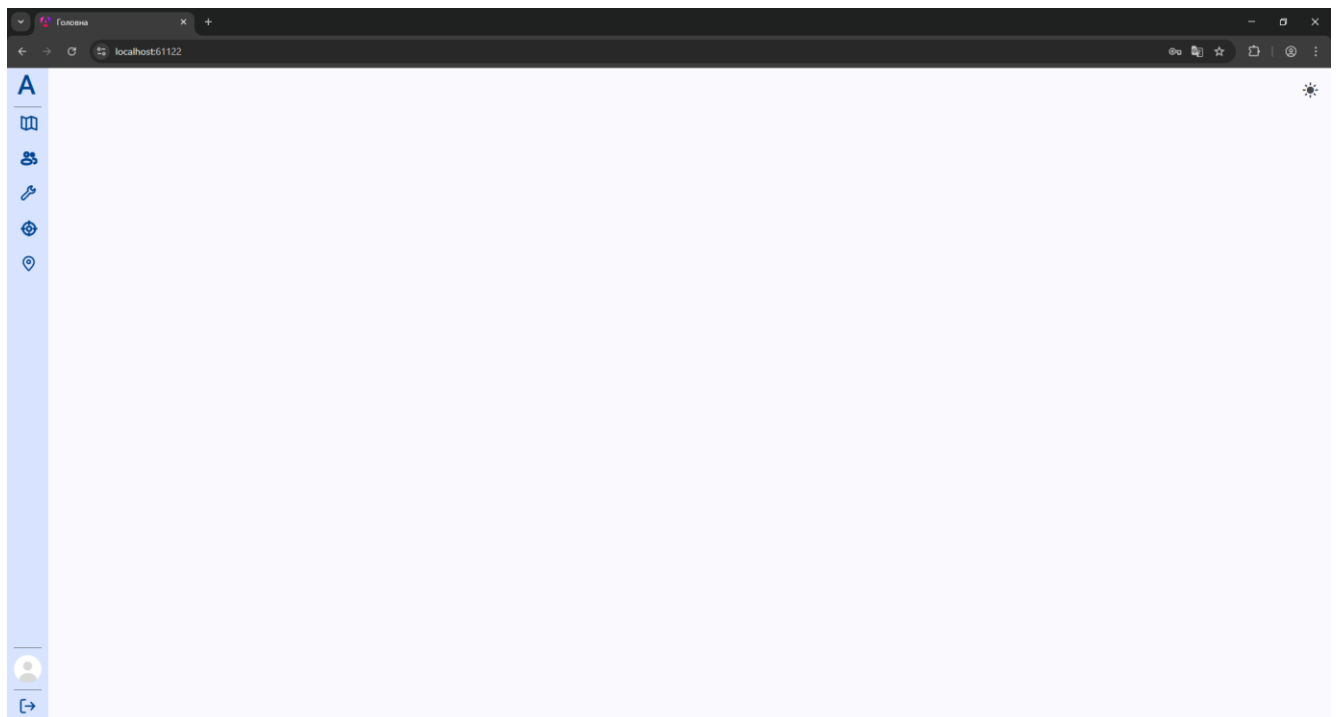


Рисунок Д-4 – Головна сторінка (світла тема)

Джерело: розроблено автором

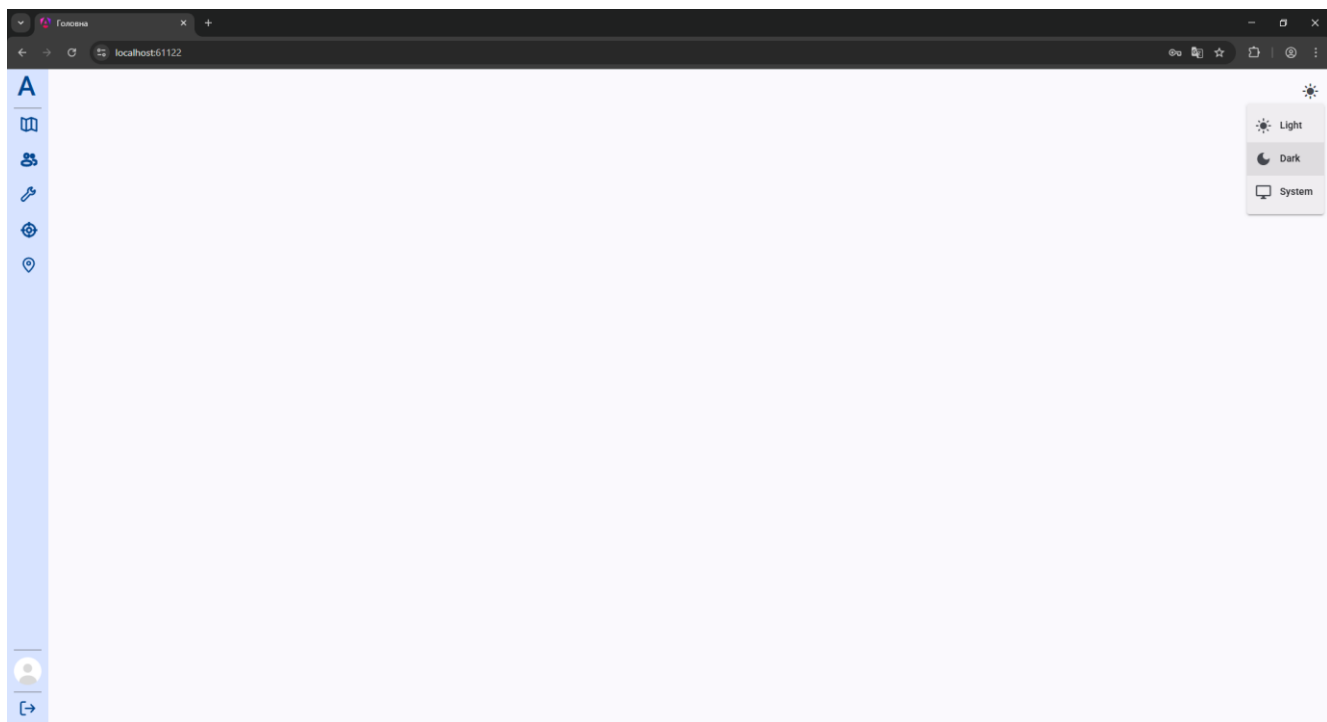


Рисунок Д-5 – Зміна теми на головні сторінці

Джерело: розроблено автором

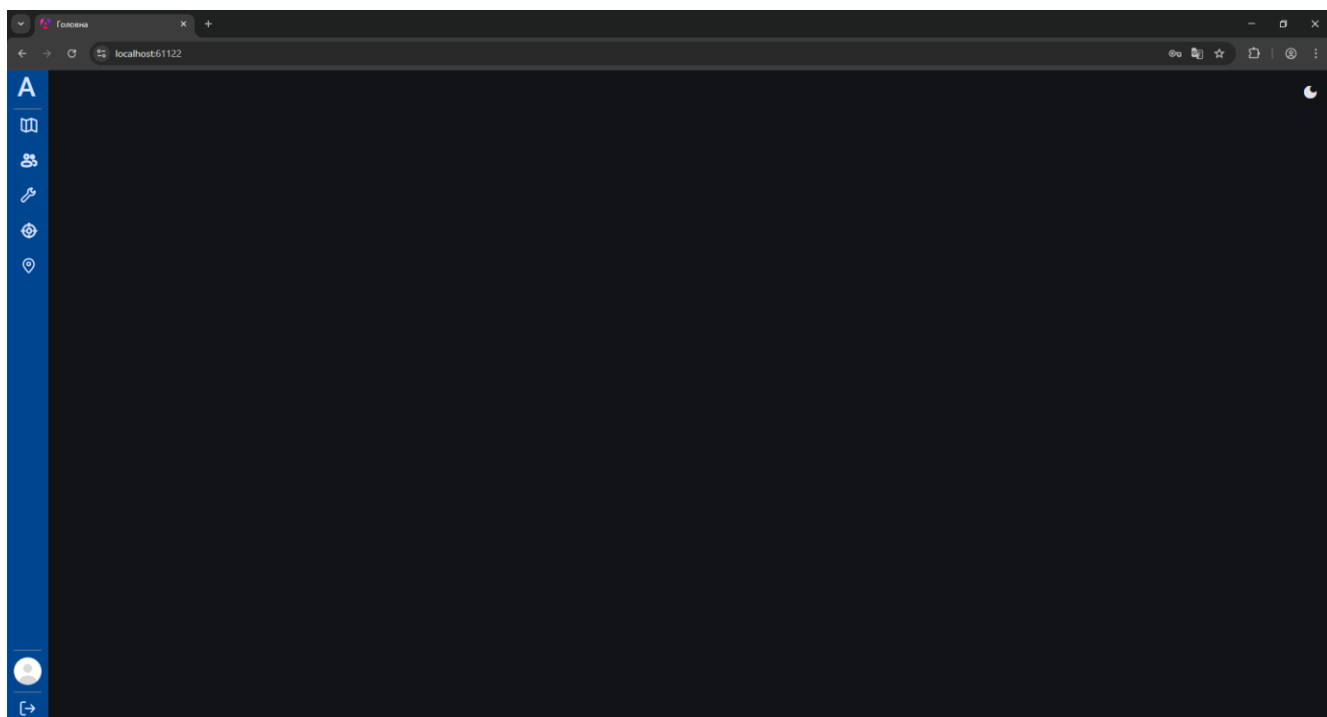


Рисунок Д-6 – Головна сторінка (темна тема)

Джерело: розроблено автором

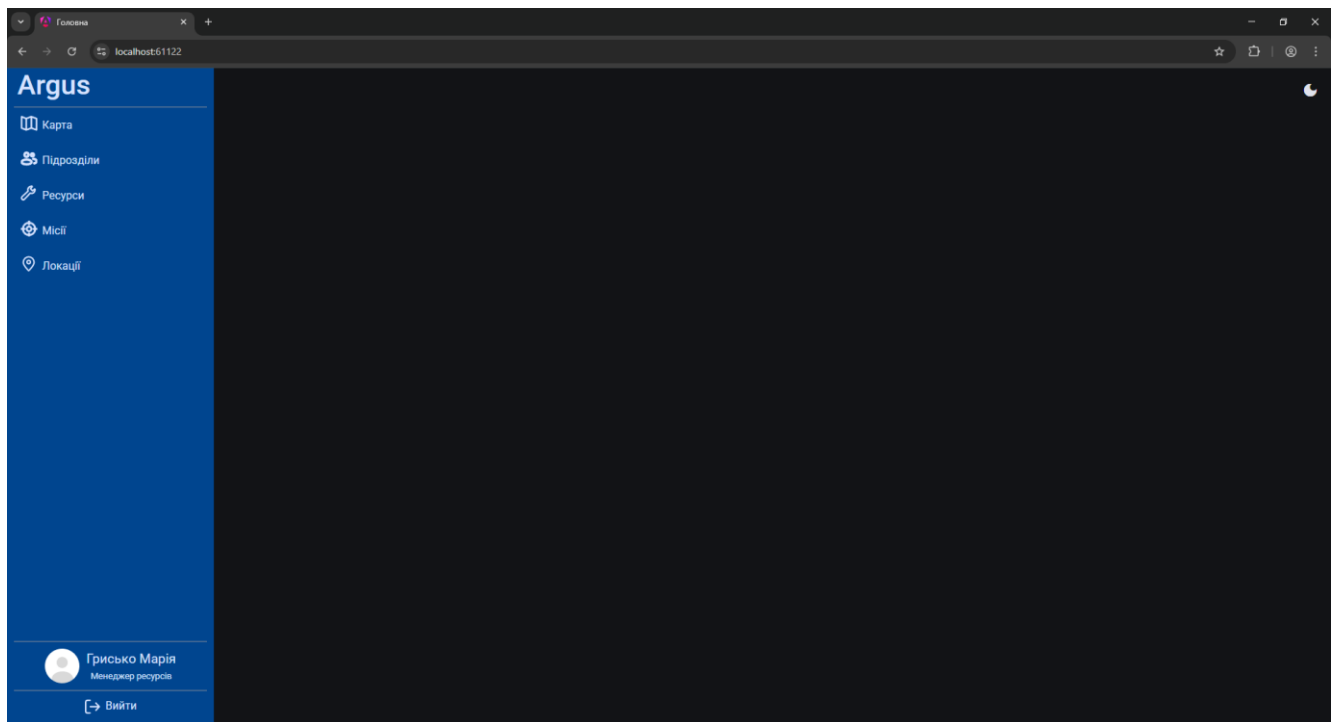


Рисунок Д-7 – Сайдбар (темна тема)

Джерело: розроблено автором

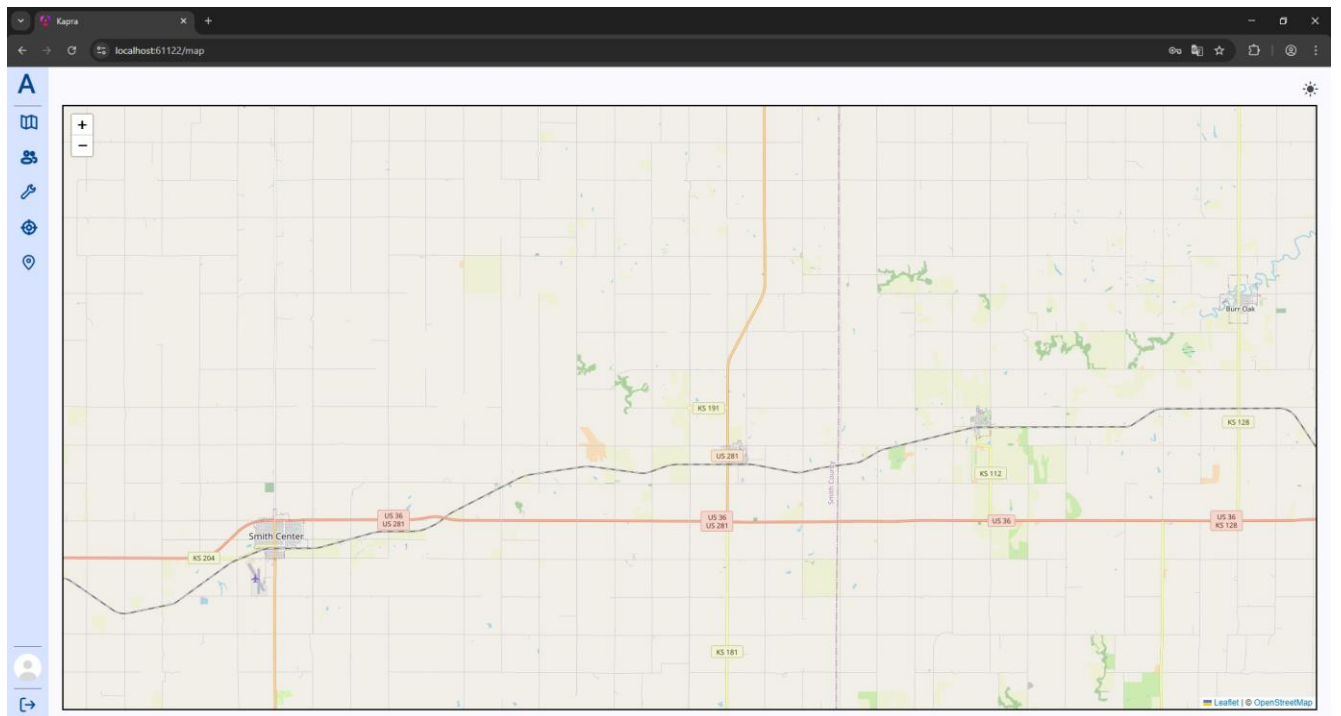


Рисунок Д-8 – Сторінка з картою

Джерело: розроблено автором

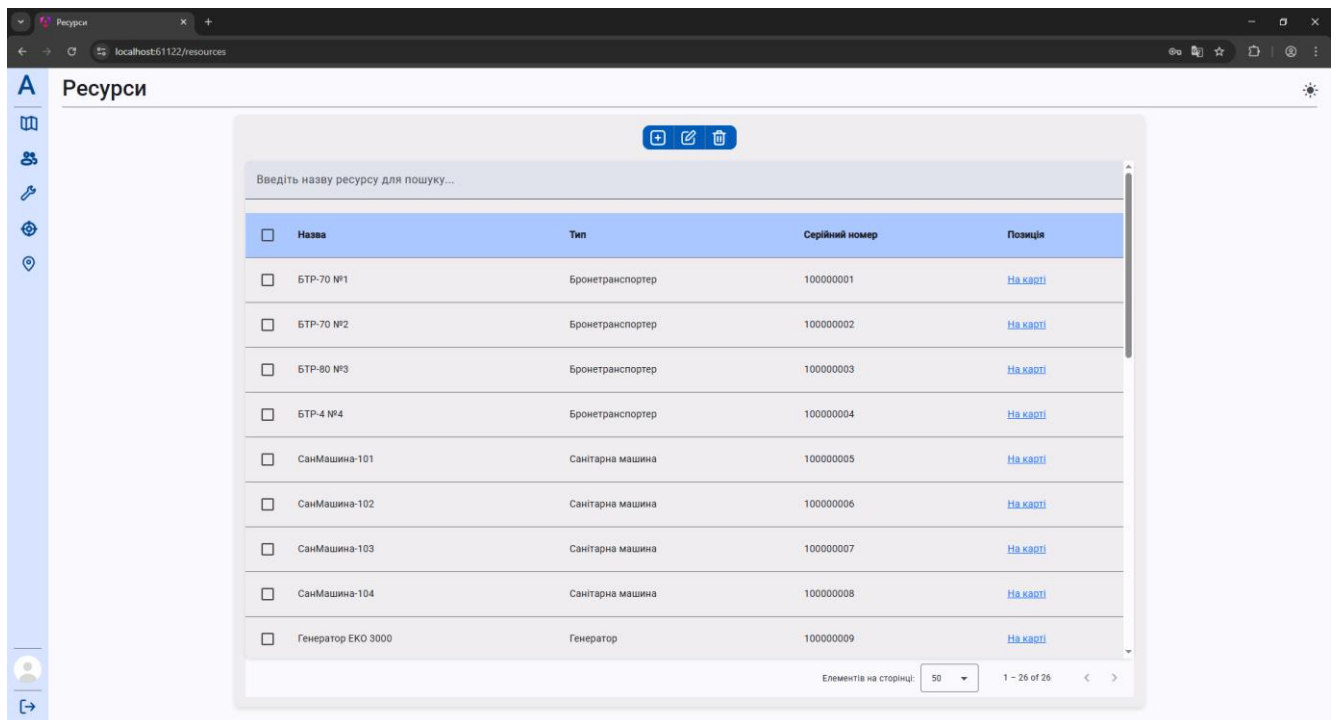


Рисунок Д-9 – Сторінка управління ресурсами

Джерело: розроблено автором

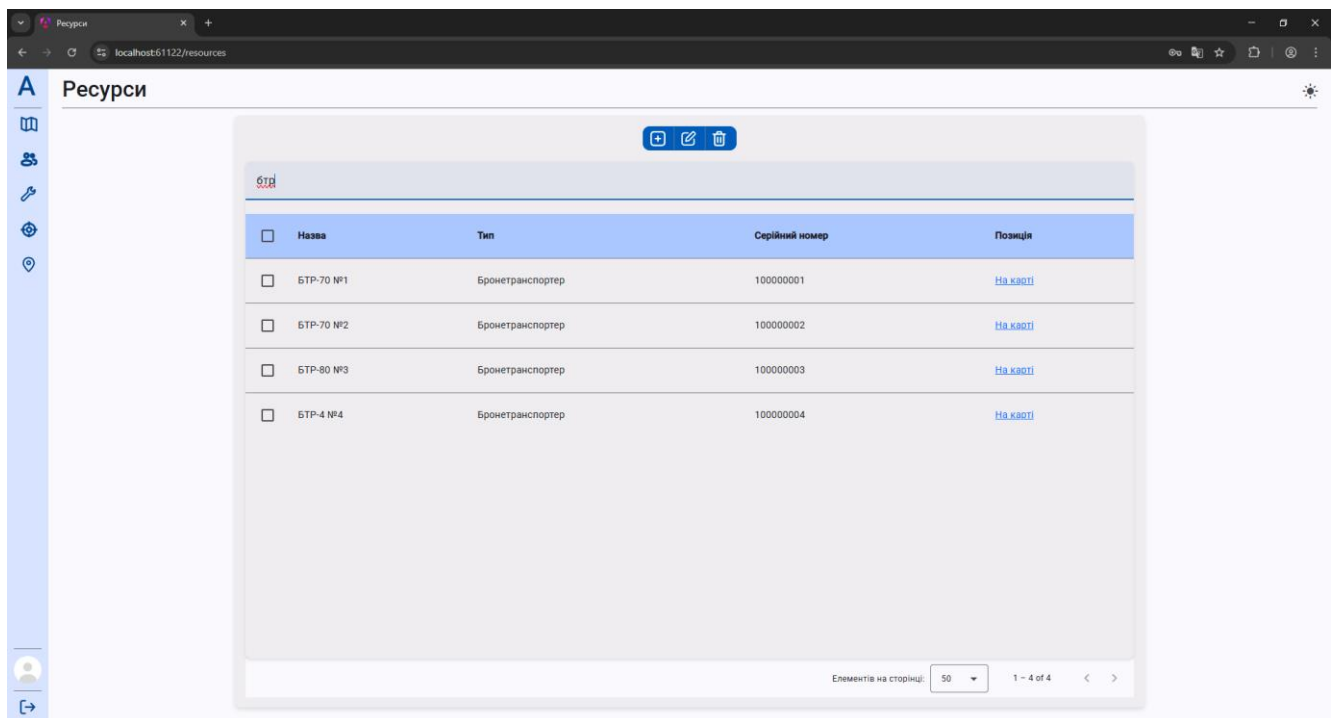


Рисунок Д-10 – Робота фільтрації на сторінці управління ресурсами

Джерело: розроблено автором

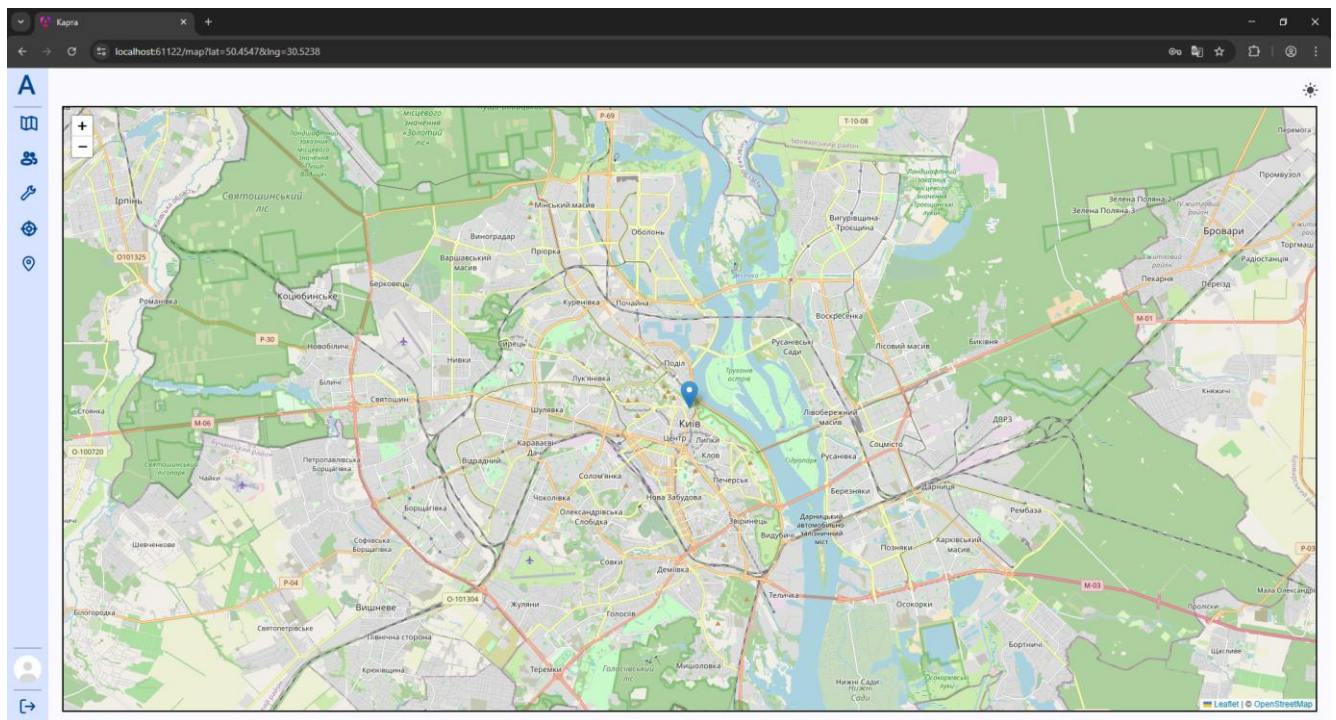


Рисунок Д-11 – Відображення ресурсу на карті

Джерело: розроблено автором

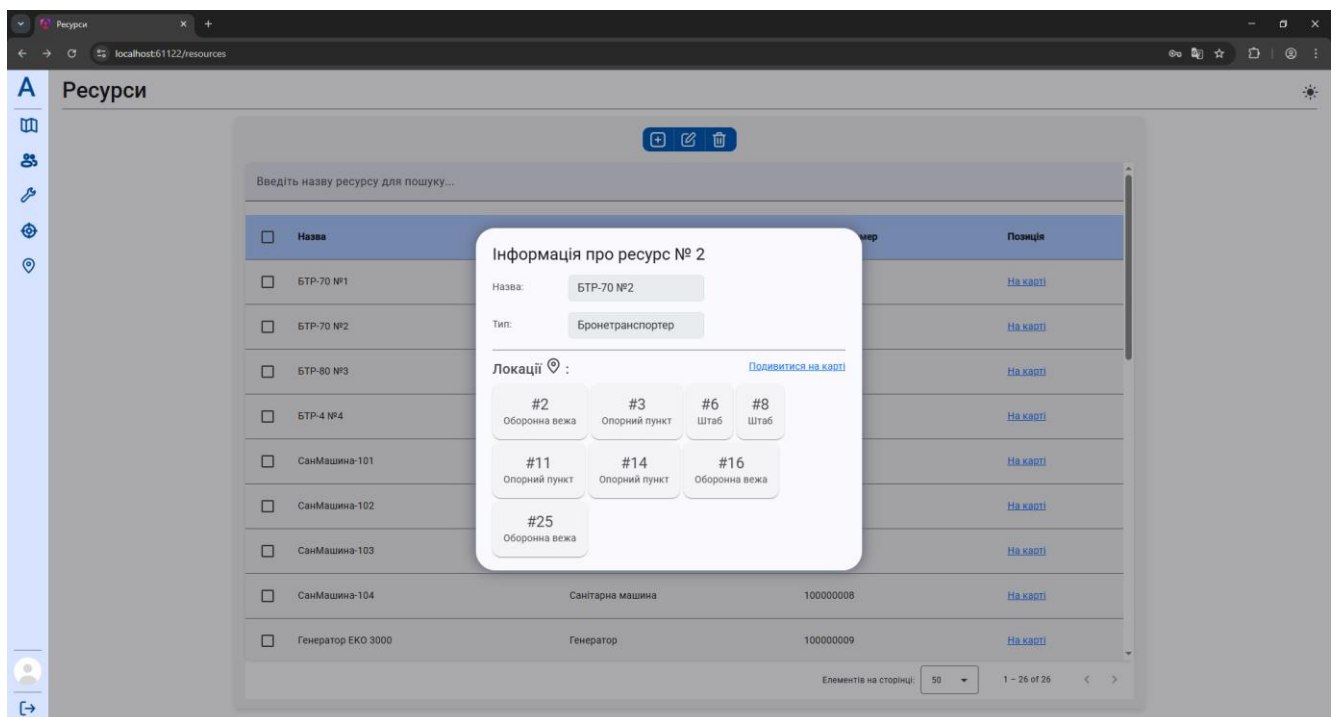


Рисунок Д-12 – Перегляд інформації по обраному ресурсу

Джерело: розроблено автором

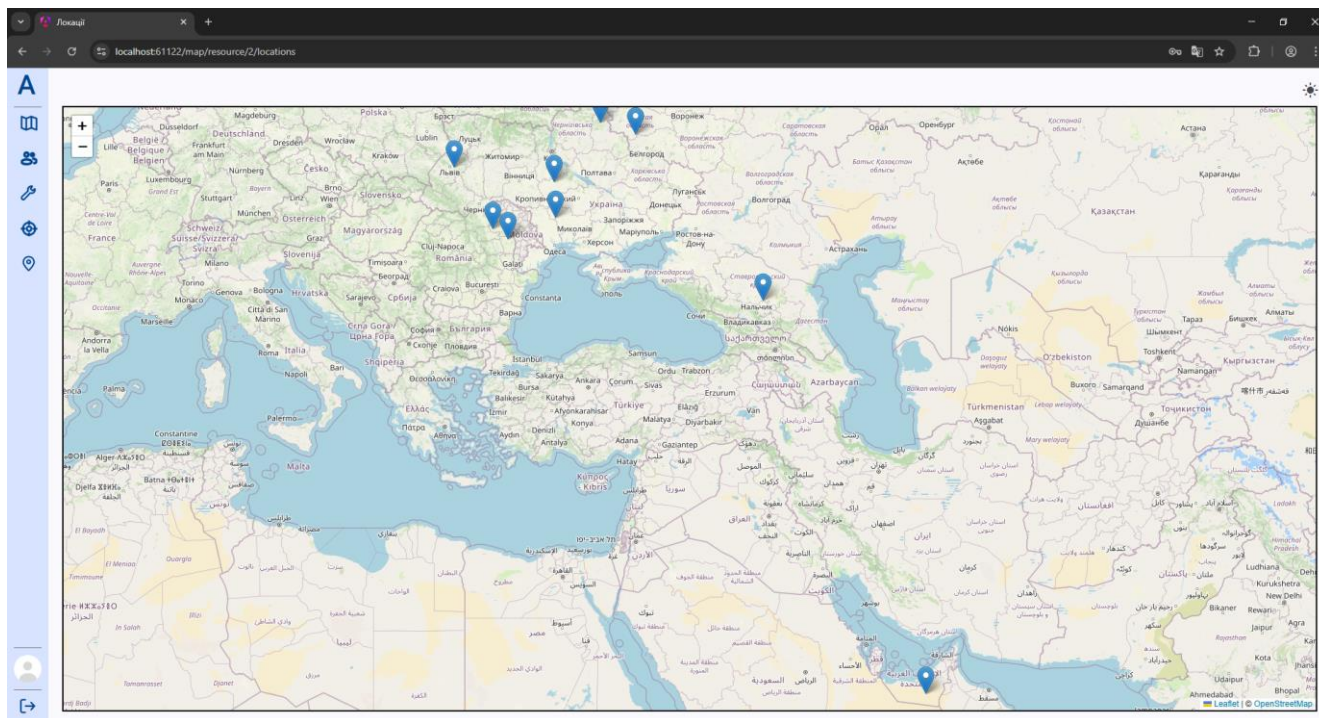


Рисунок Д-13 – Перегляд локацій, на яких є обраний ресурс

Джерело: розроблено автором

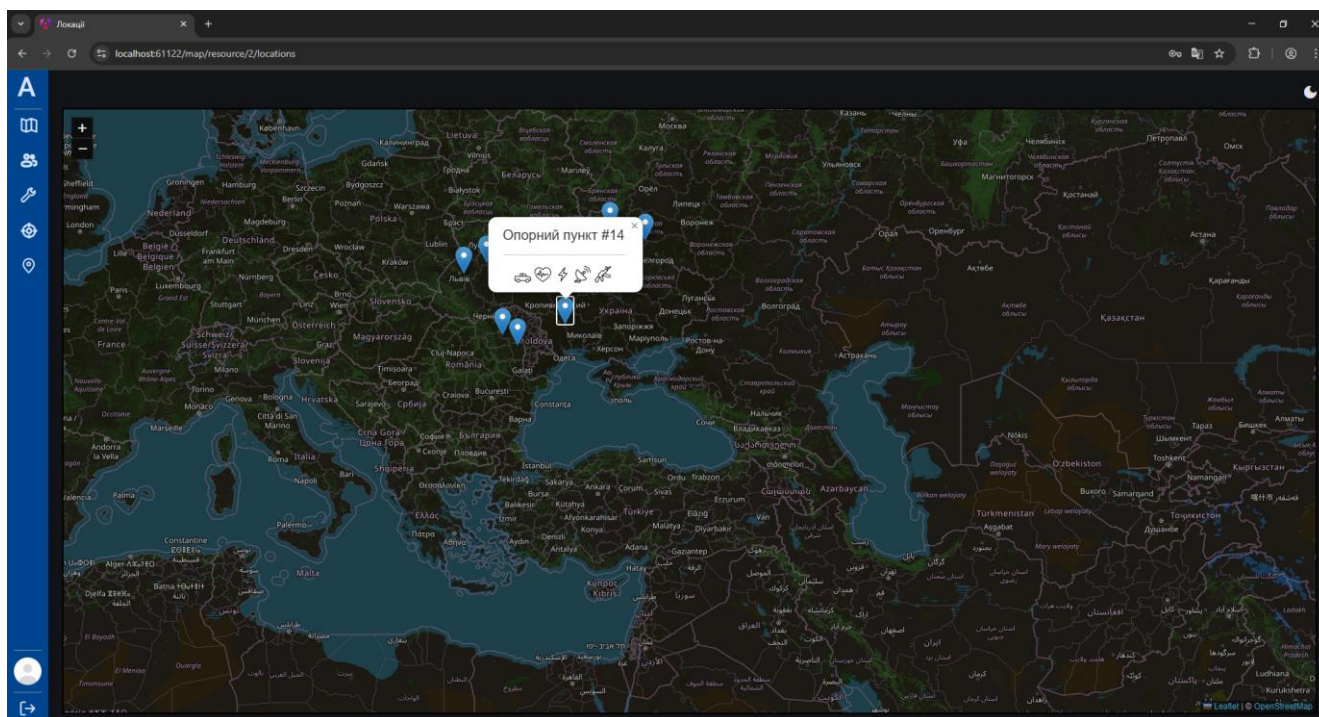


Рисунок Д-14 – Перегляд короткої інформації про локацію на карті

Джерело: розроблено автором

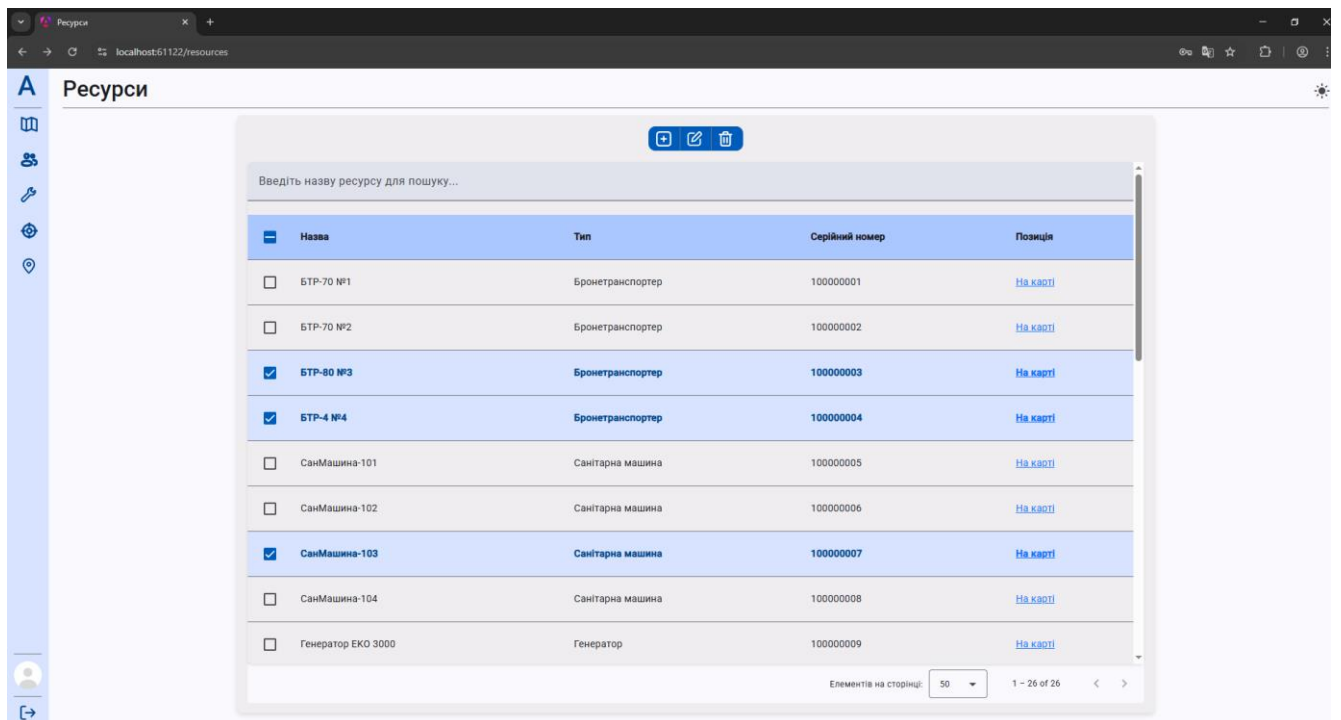


Рисунок Д-15 – Виділення ресурсів

Джерело: розроблено автором

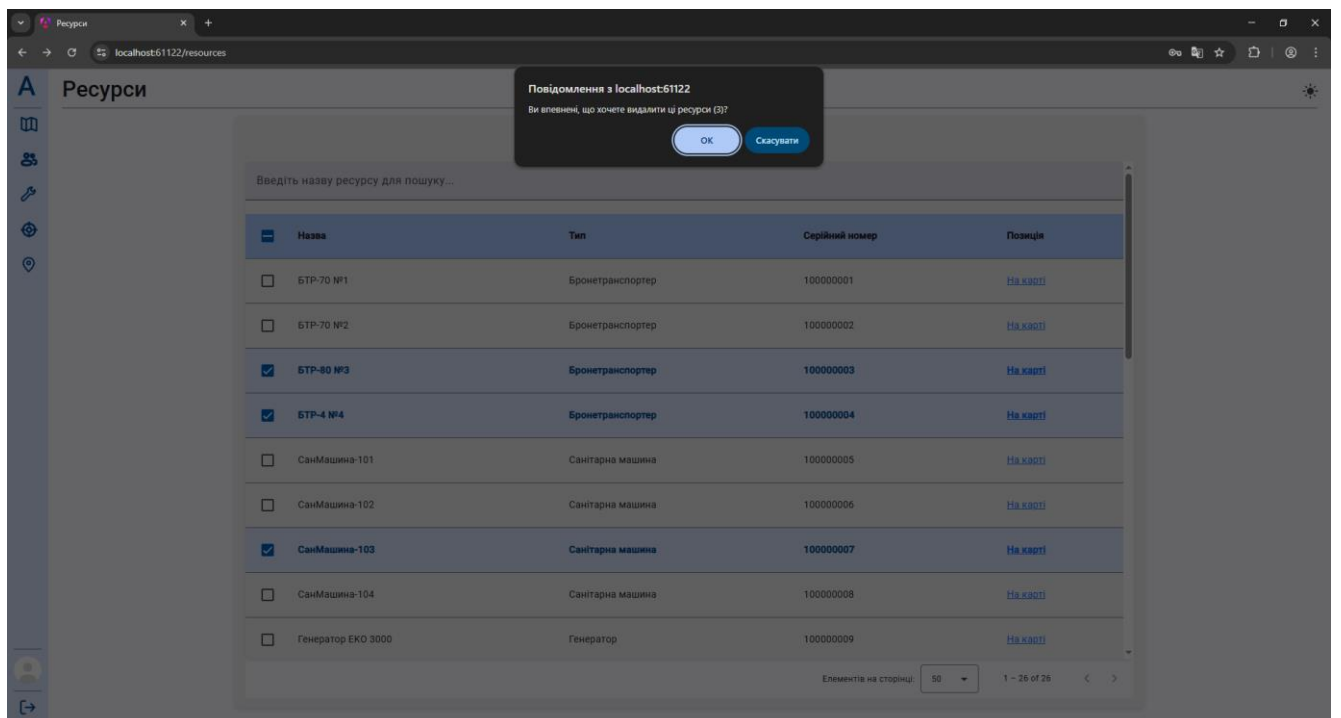


Рисунок Д-16 – Видалення ресурсів

Джерело: розроблено автором

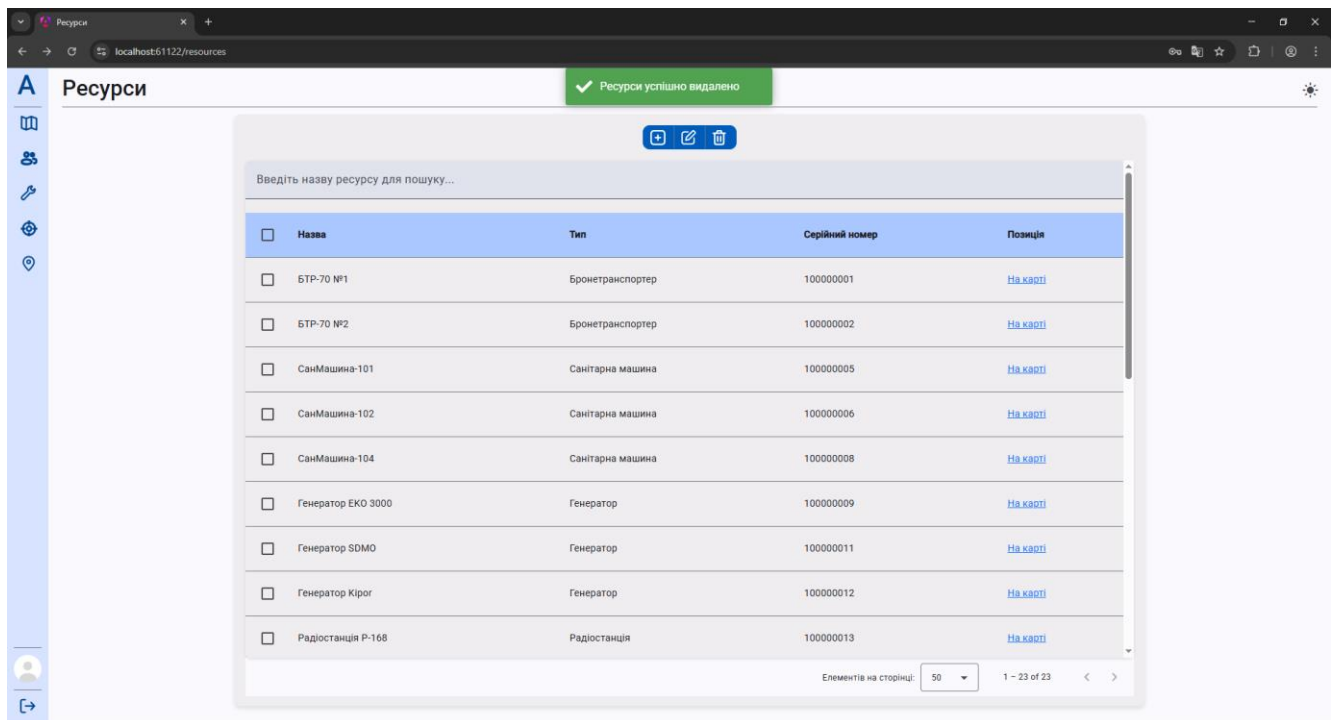


Рисунок Д-17 – Повідомлення про успішне видалення ресурсів

Джерело: розроблено автором

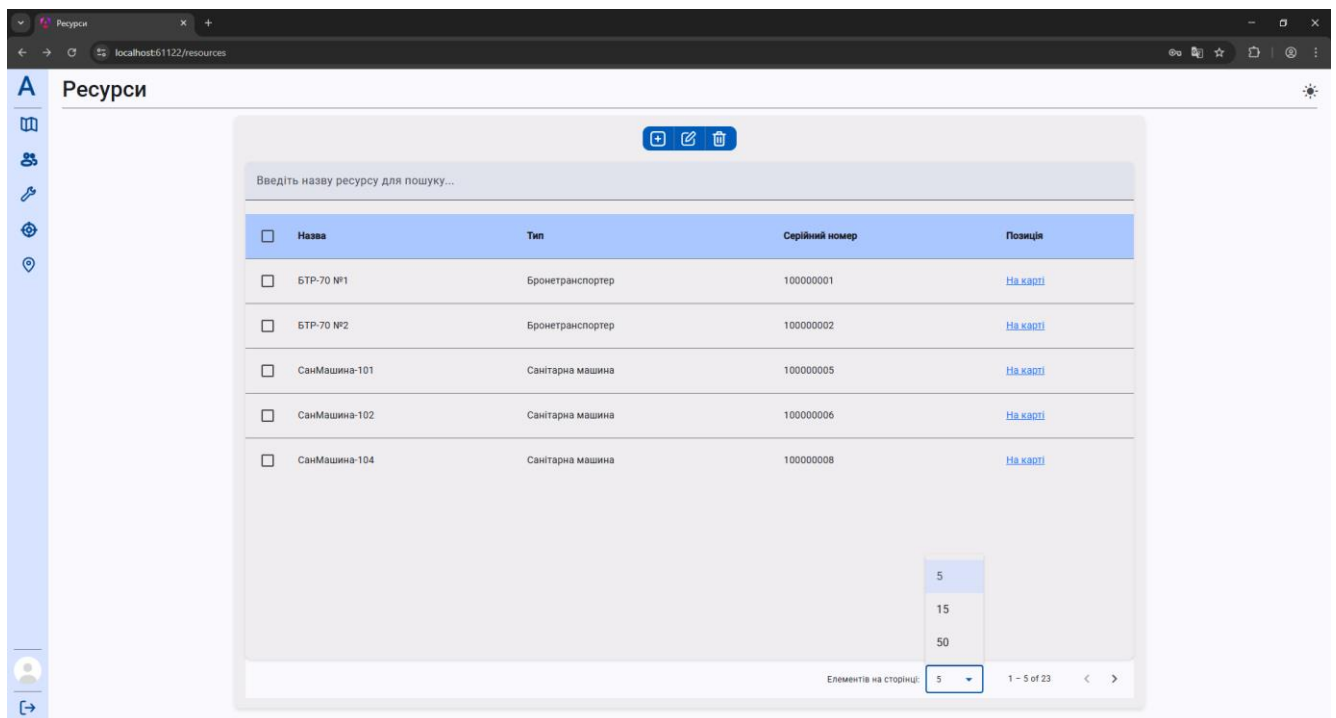


Рисунок Д-18 – Зміна розміру сторінок, пагінація

Джерело: розроблено автором

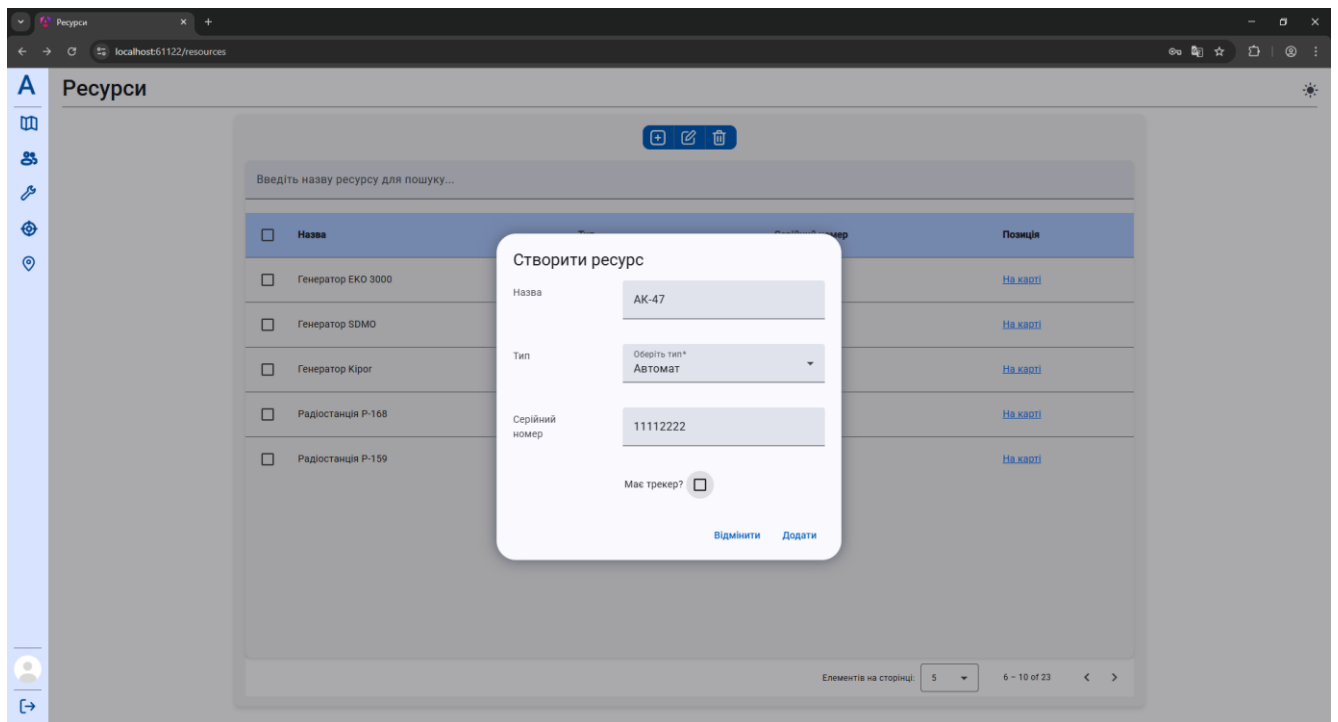


Рисунок Д-19 – Створення ресурсу

Джерело: розроблено автором

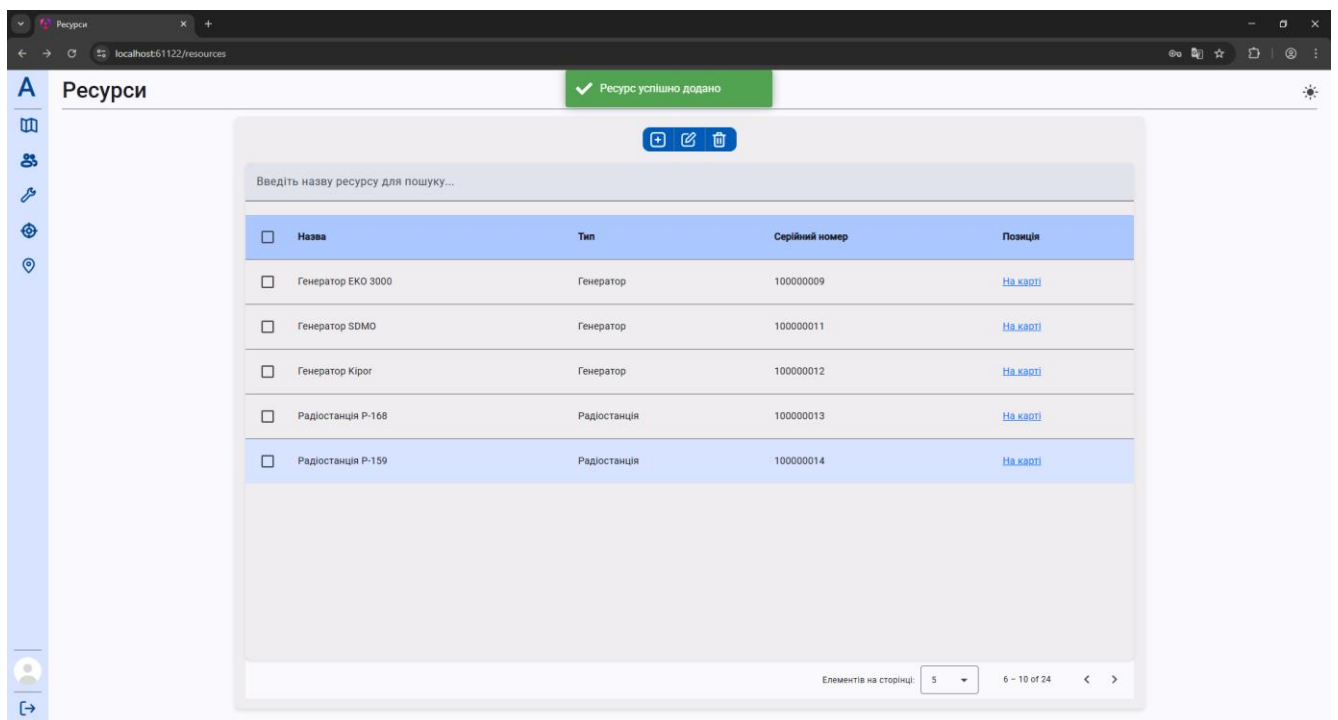


Рисунок Д-20 – Повідомлення про успішно доданий ресурс

Джерело: розроблено автором

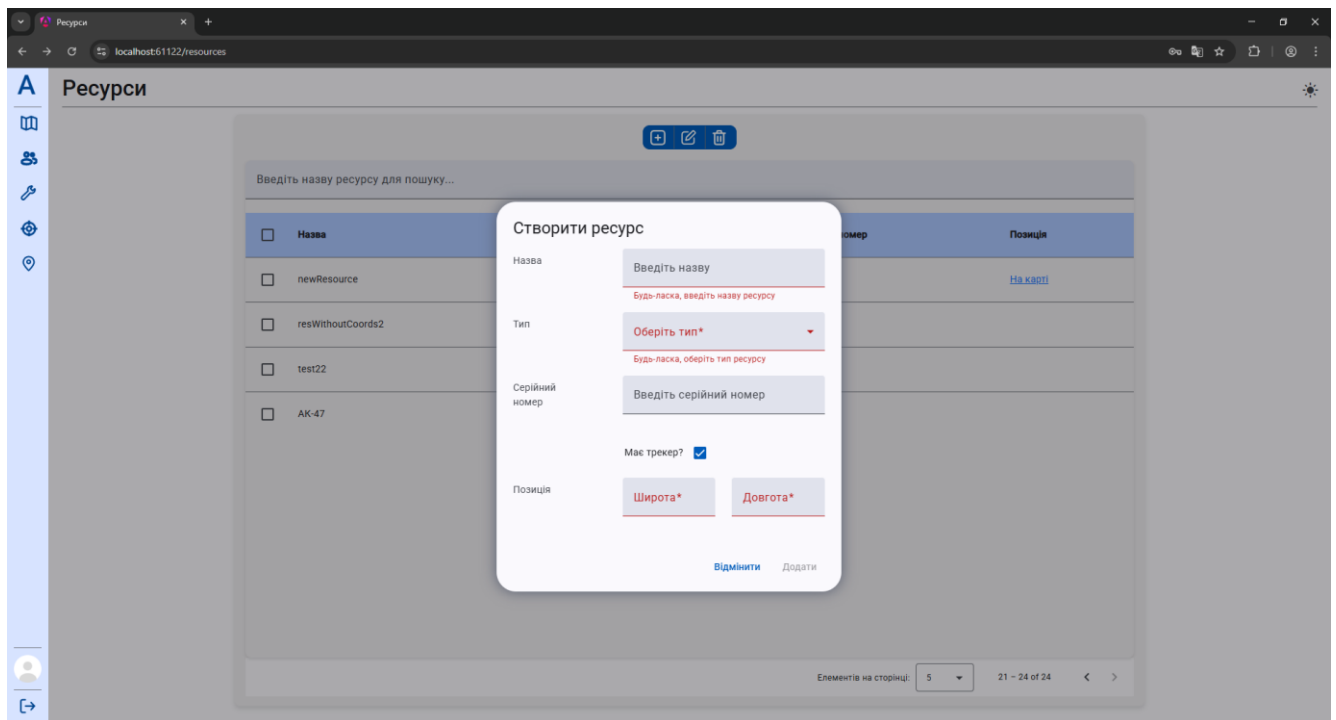


Рисунок Д-21 – Форма для створення ресурсу з помилковими даними

Джерело: розроблено автором

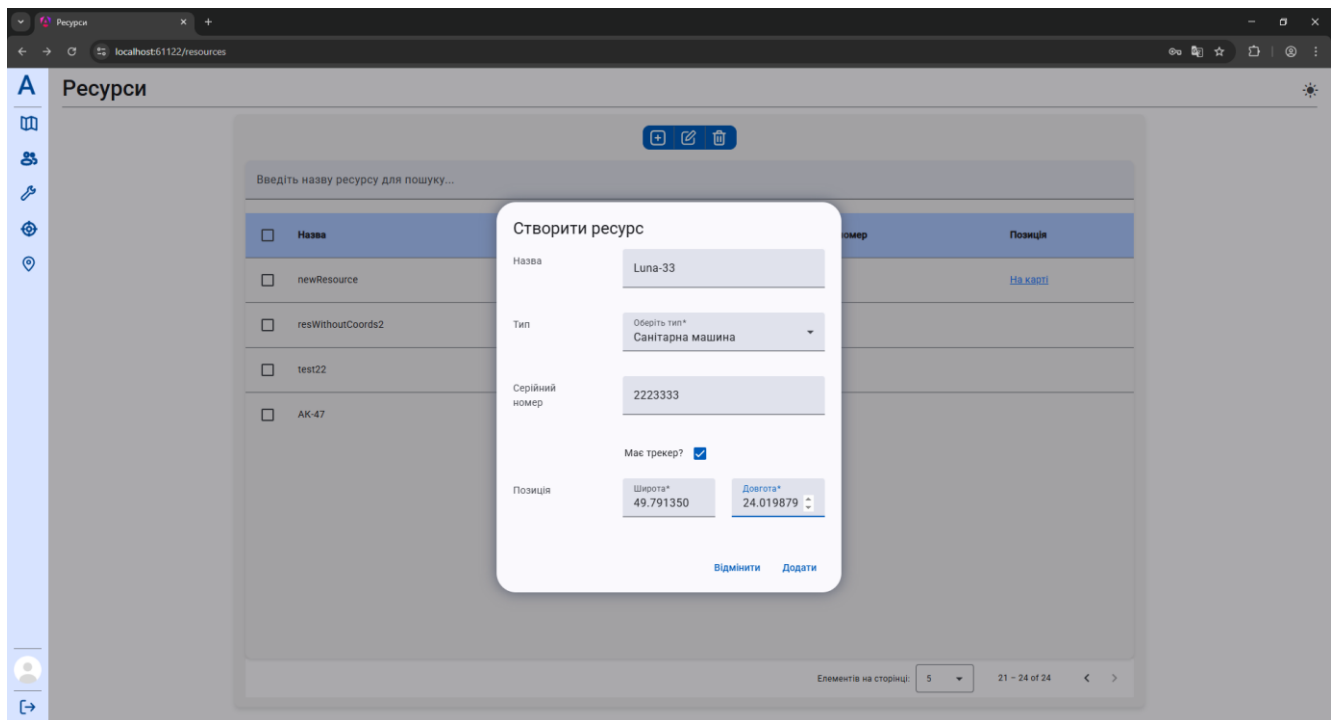


Рисунок Д-22 – Форма для створення ресурсу з координатами

Джерело: розроблено автором

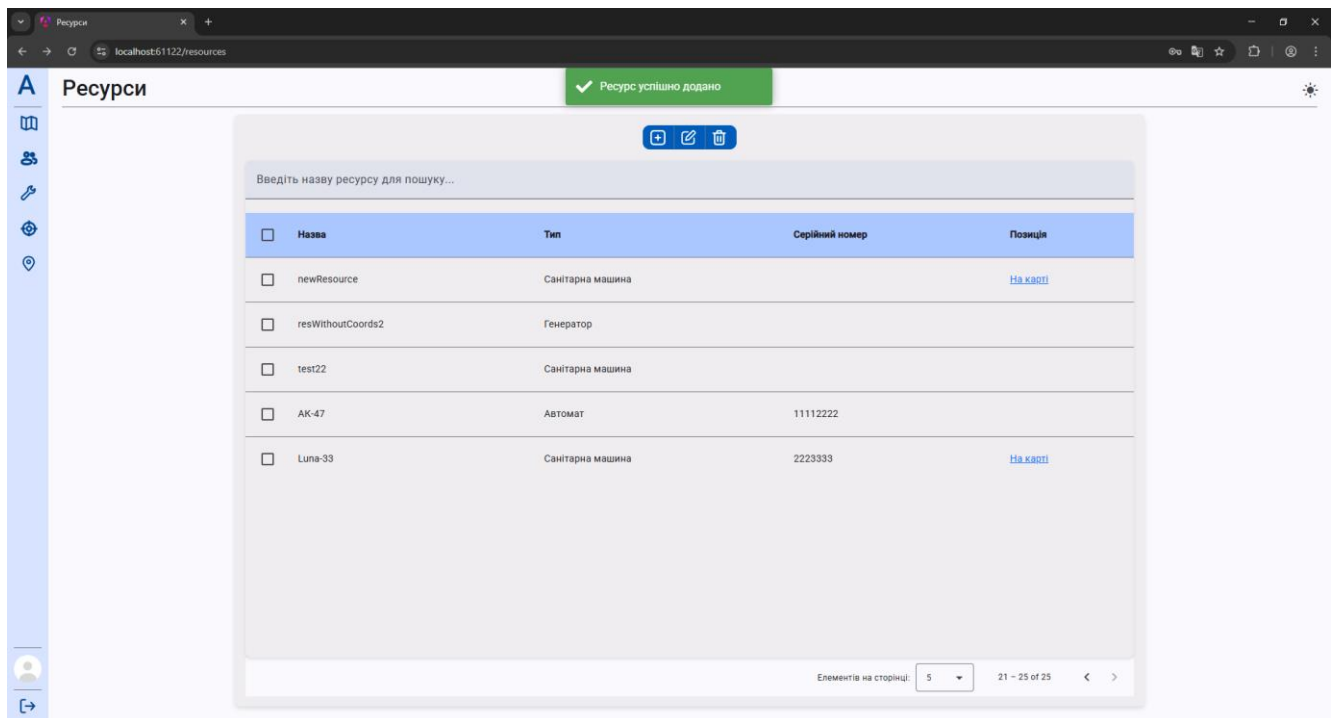


Рисунок Д-23 – Повідомлення про успішно створений ресурс.

Джерело: розроблено автором

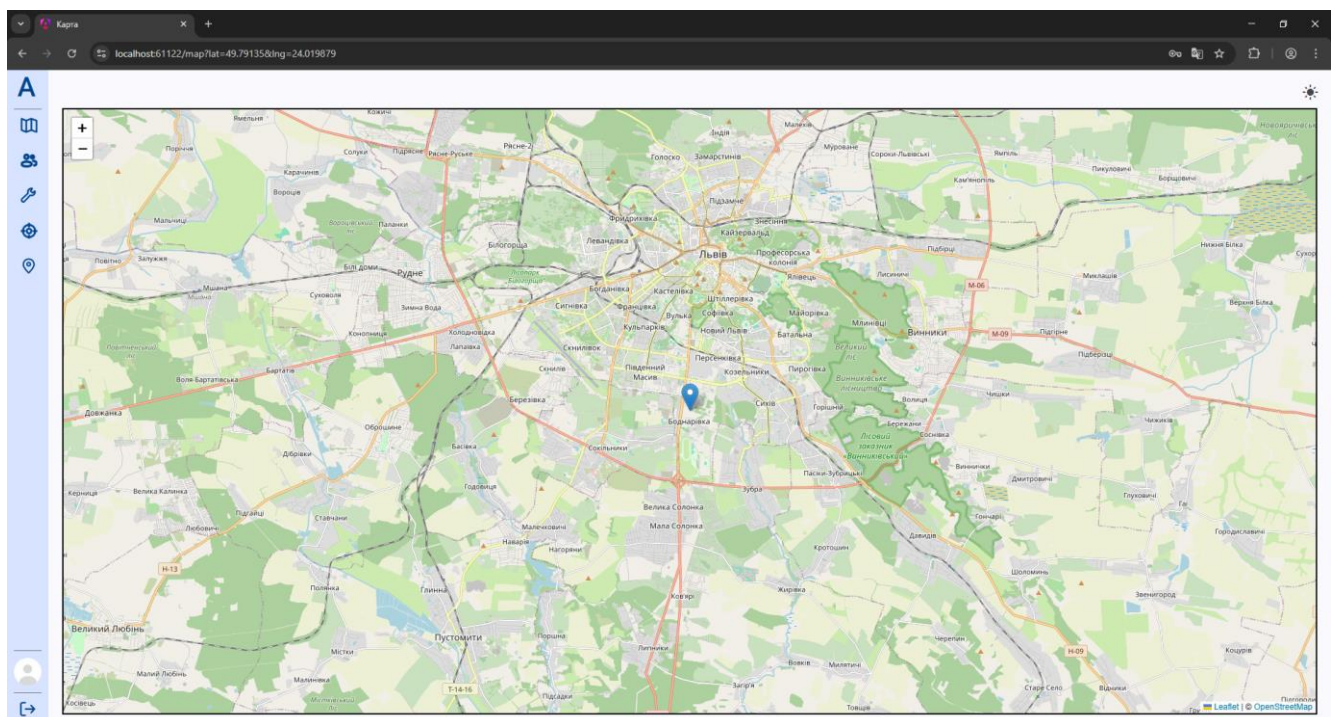


Рисунок Д-24 – Відображення новоствореного ресурсу на карті

Джерело: розроблено автором

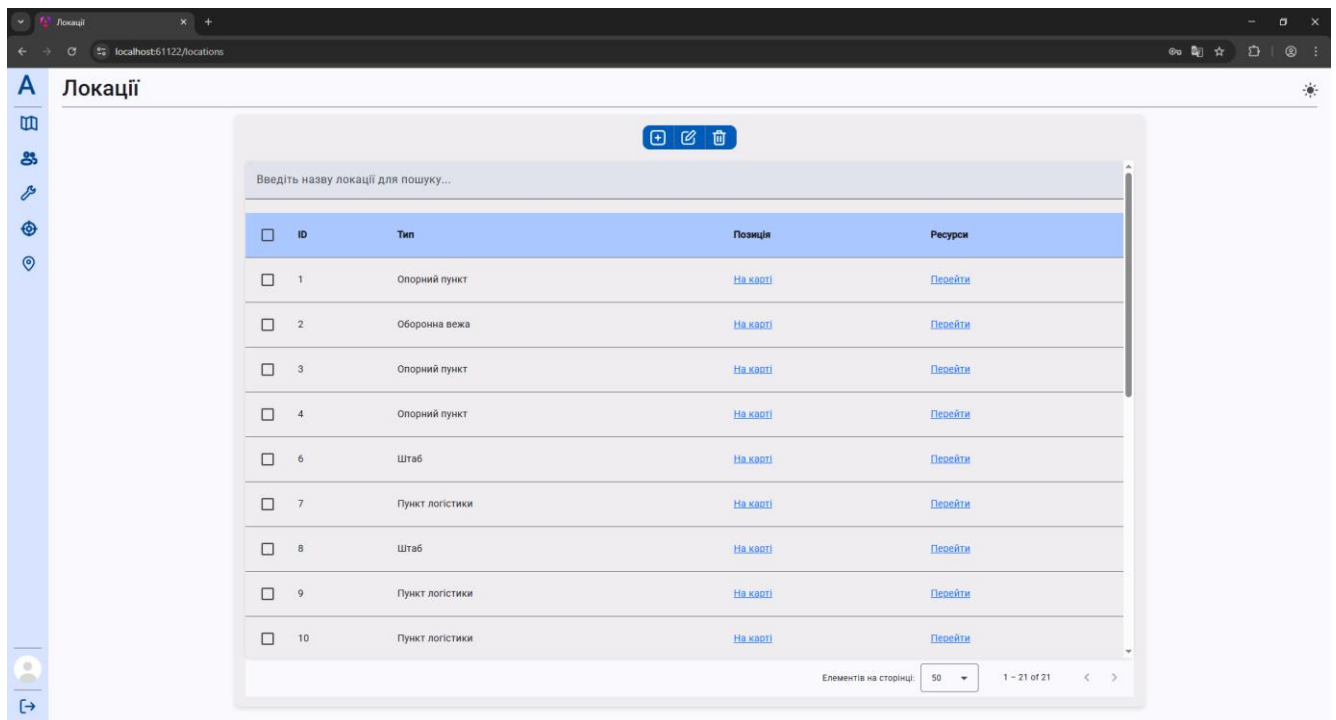


Рисунок Д-25 – Сторінка локацій

Джерело: розроблено автором

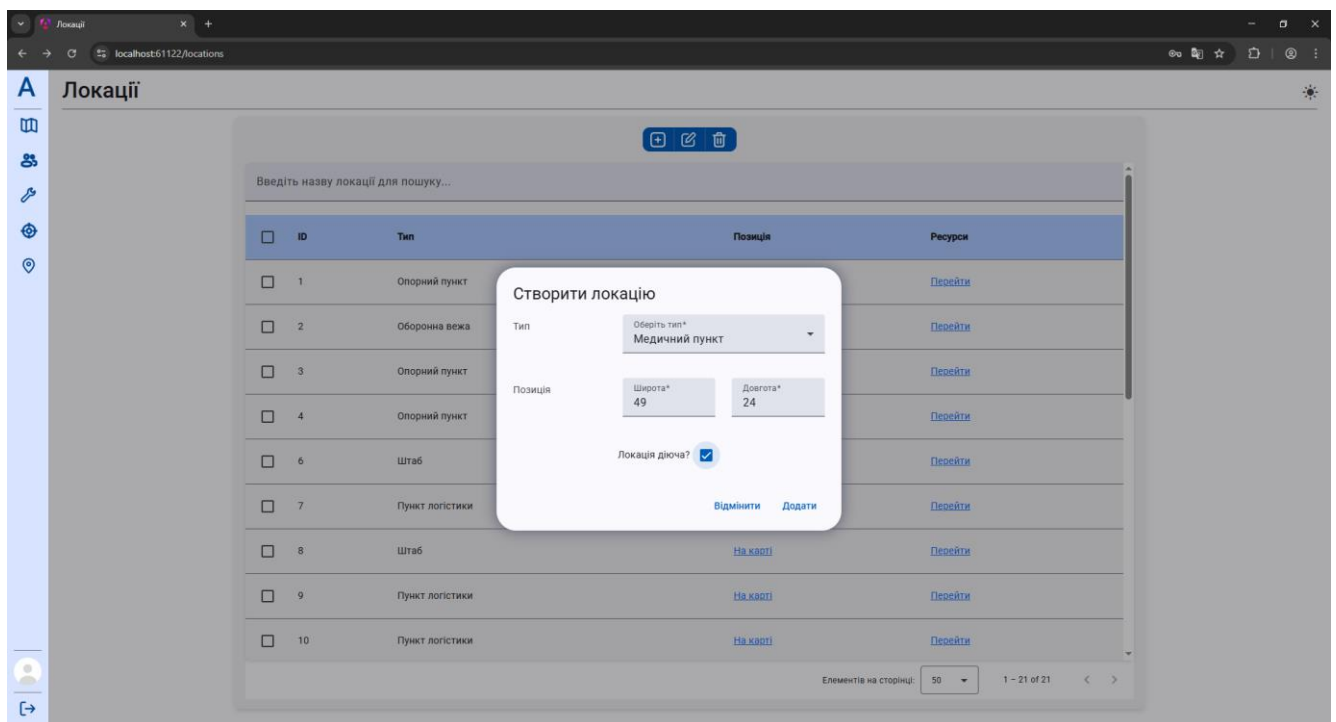


Рисунок Д-26 – Форма створення нової локації

Джерело: розроблено автором

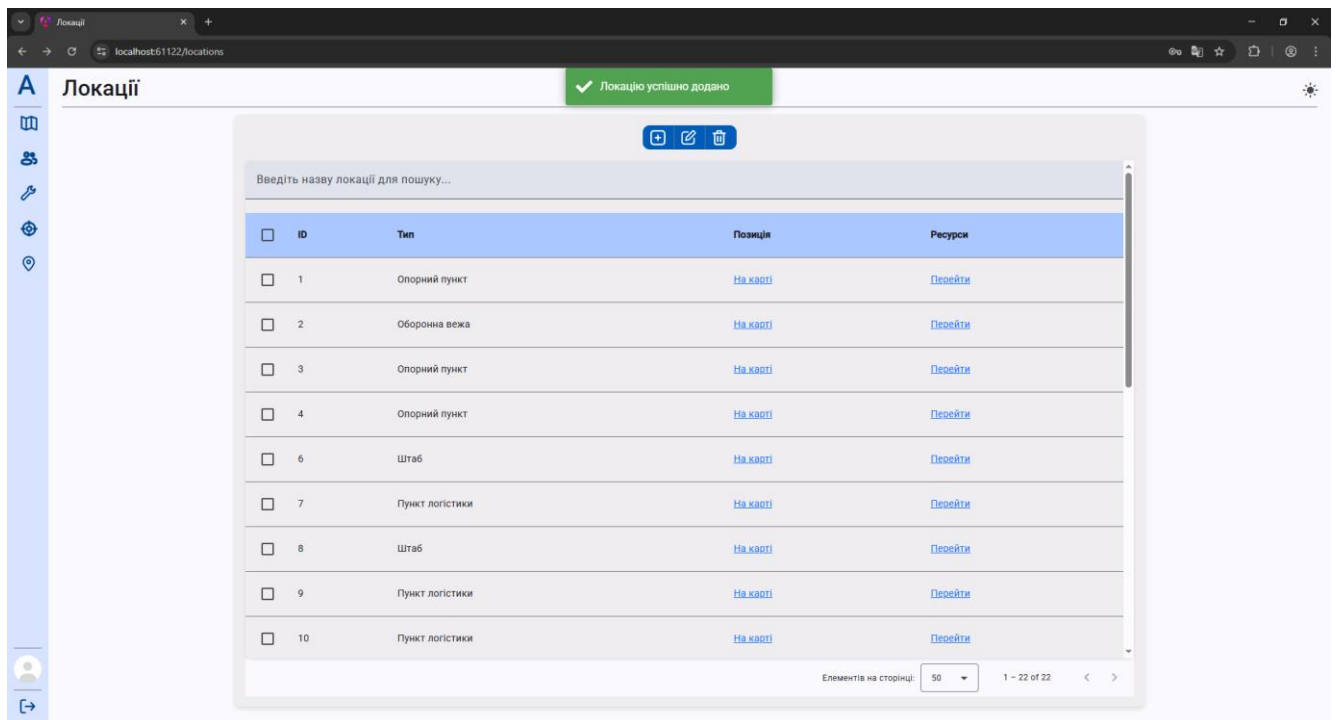


Рисунок Д-27 – Повідомлення про успішне створення локації

Джерело: розроблено автором

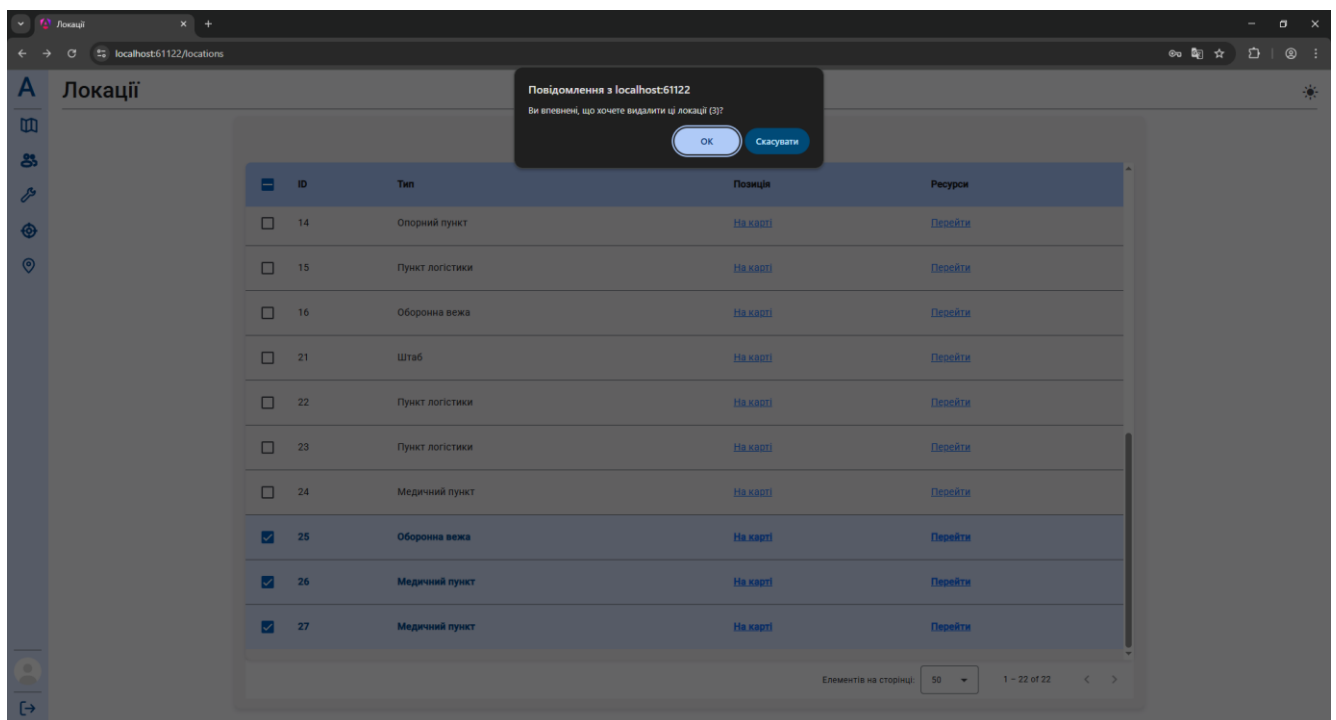


Рисунок Д-28 – Видалення локацій

Джерело: розроблено автором

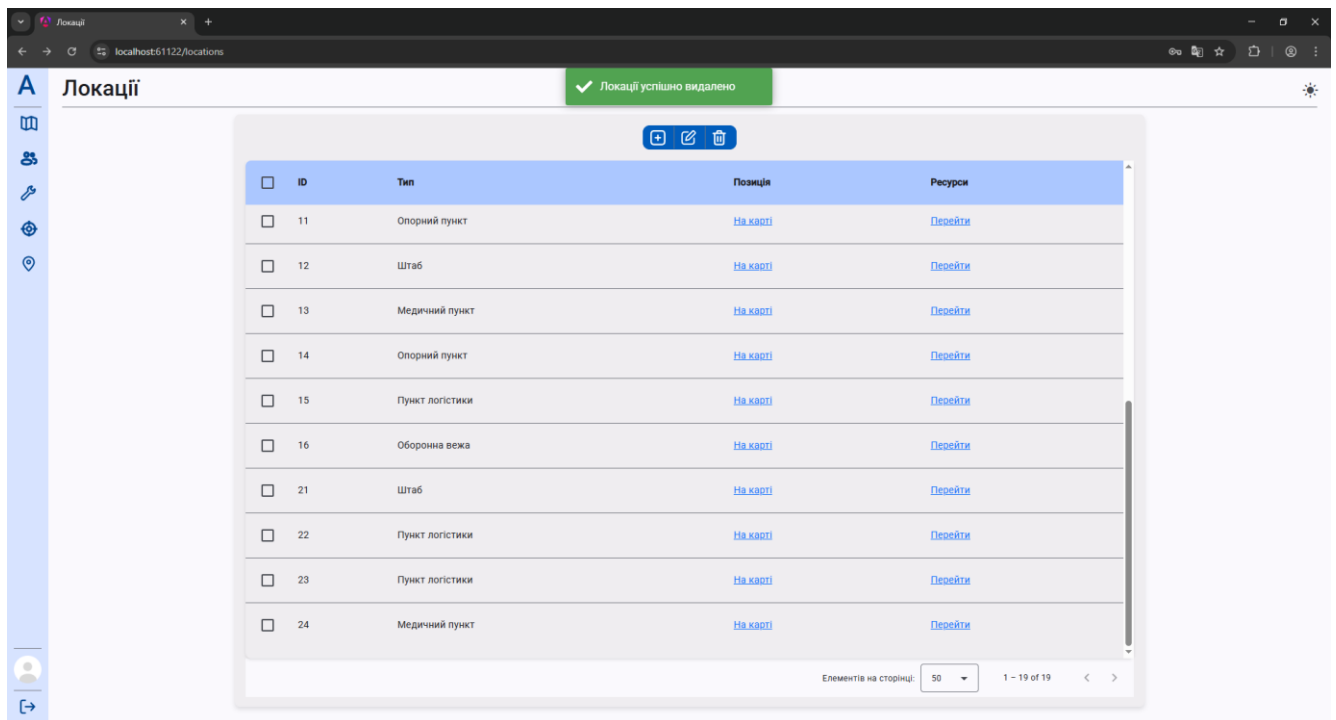


Рисунок Д-29 – Повідомлення про успішне видалення локацій

Джерело: розроблено автором

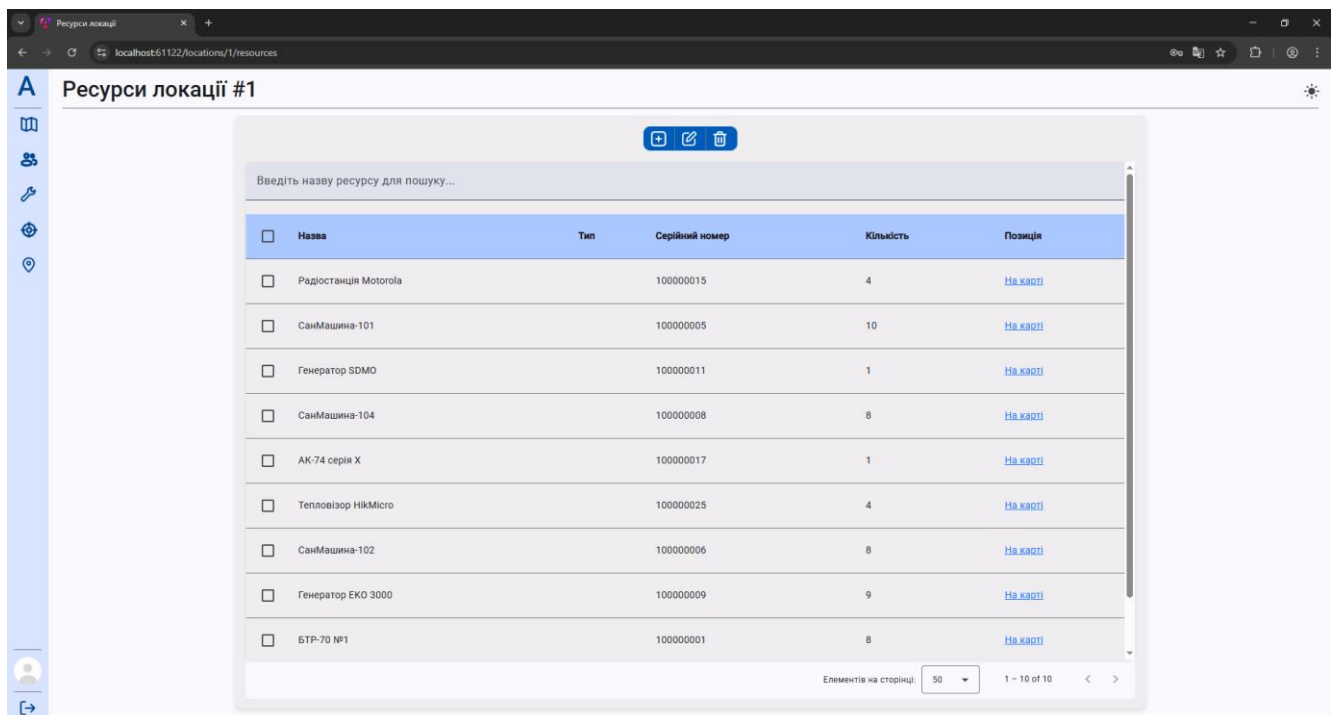


Рисунок Д-30 – Перегляд ресурсів локації

Джерело: розроблено автором

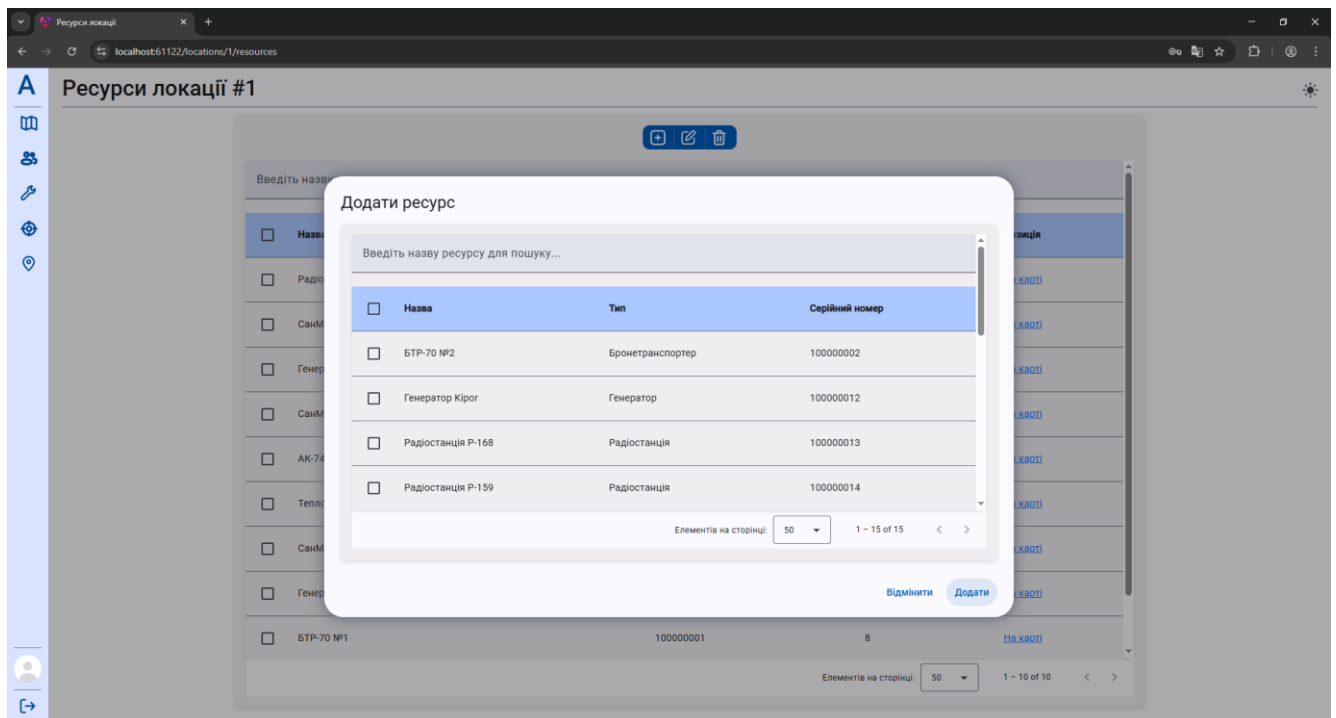


Рисунок Д-31 – Форма додавання нових ресурсів до локації

Джерело: розроблено автором

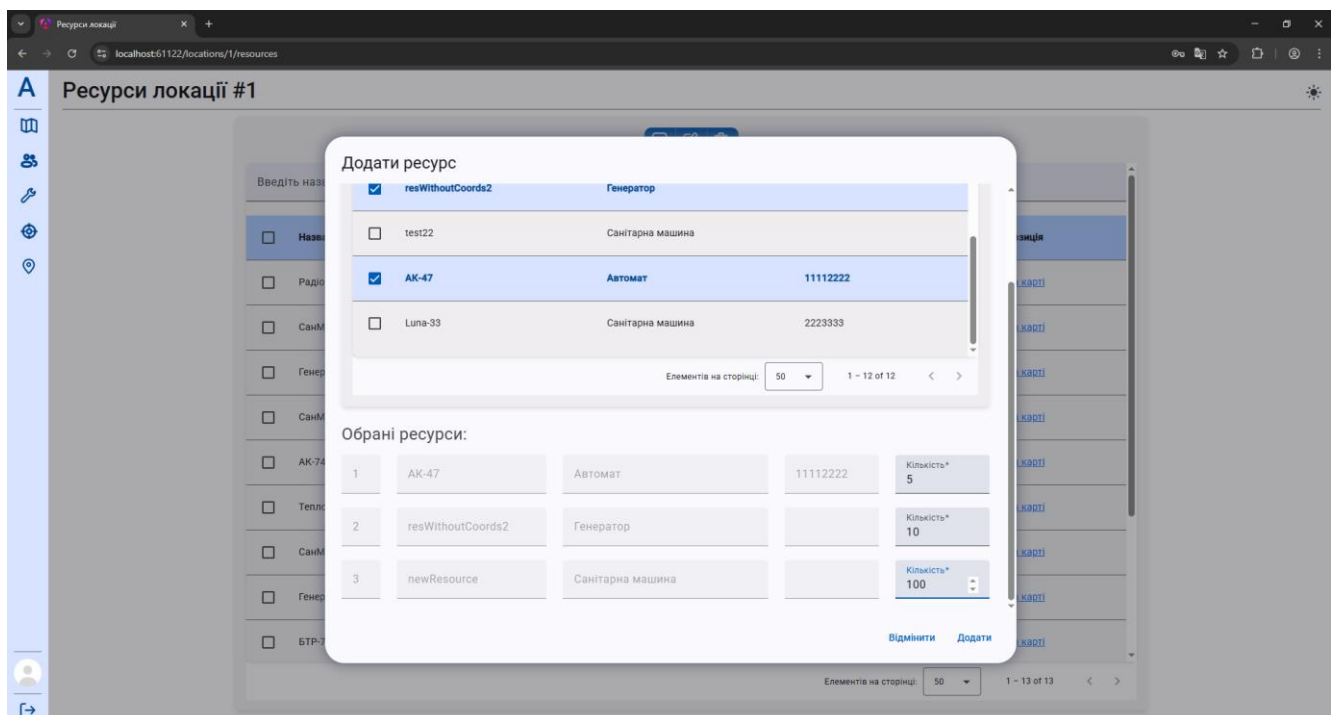


Рисунок Д-32 – Форма додавання нових ресурсів до локації з обраними ресурсами для додавання

Джерело: розроблено автором

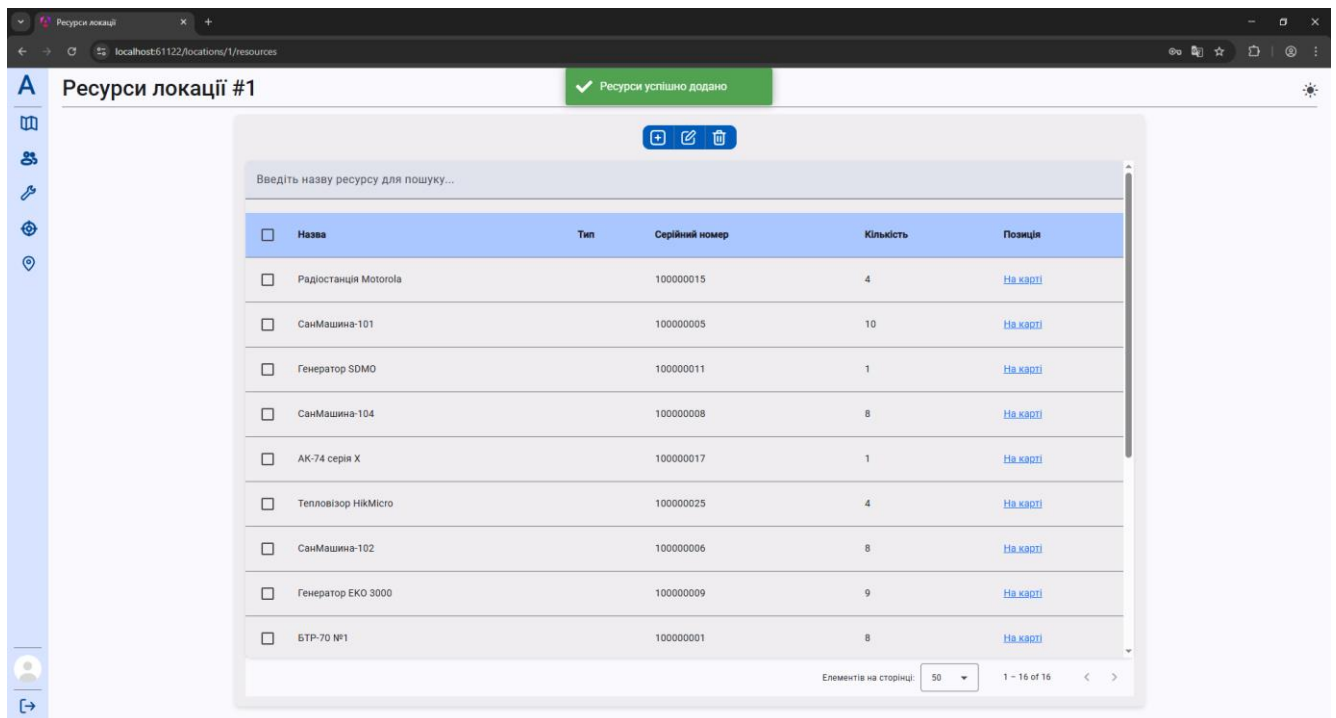


Рисунок Д-33 – Повідомлення про успішне додавання ресурсів до локації
Джерело: розроблено автором

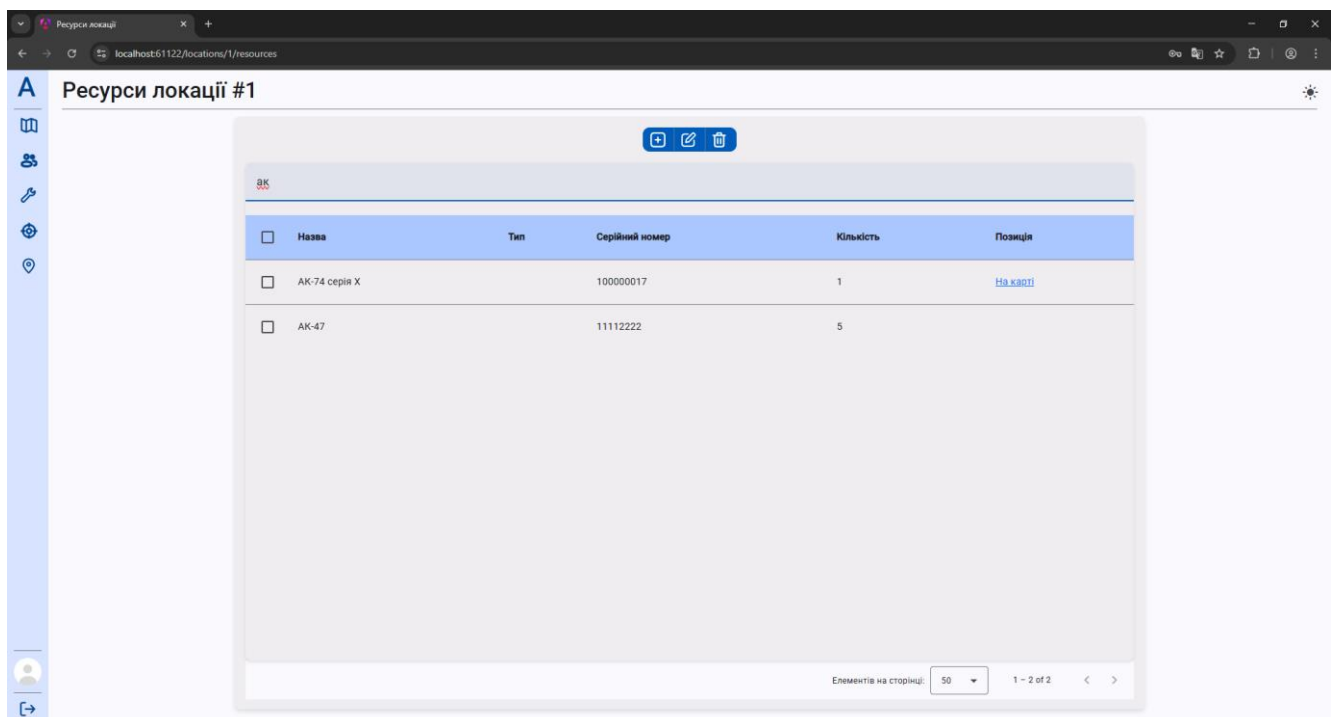


Рисунок Д-34 – Відфільтровані ресурси локації по назві
Джерело: розроблено автором

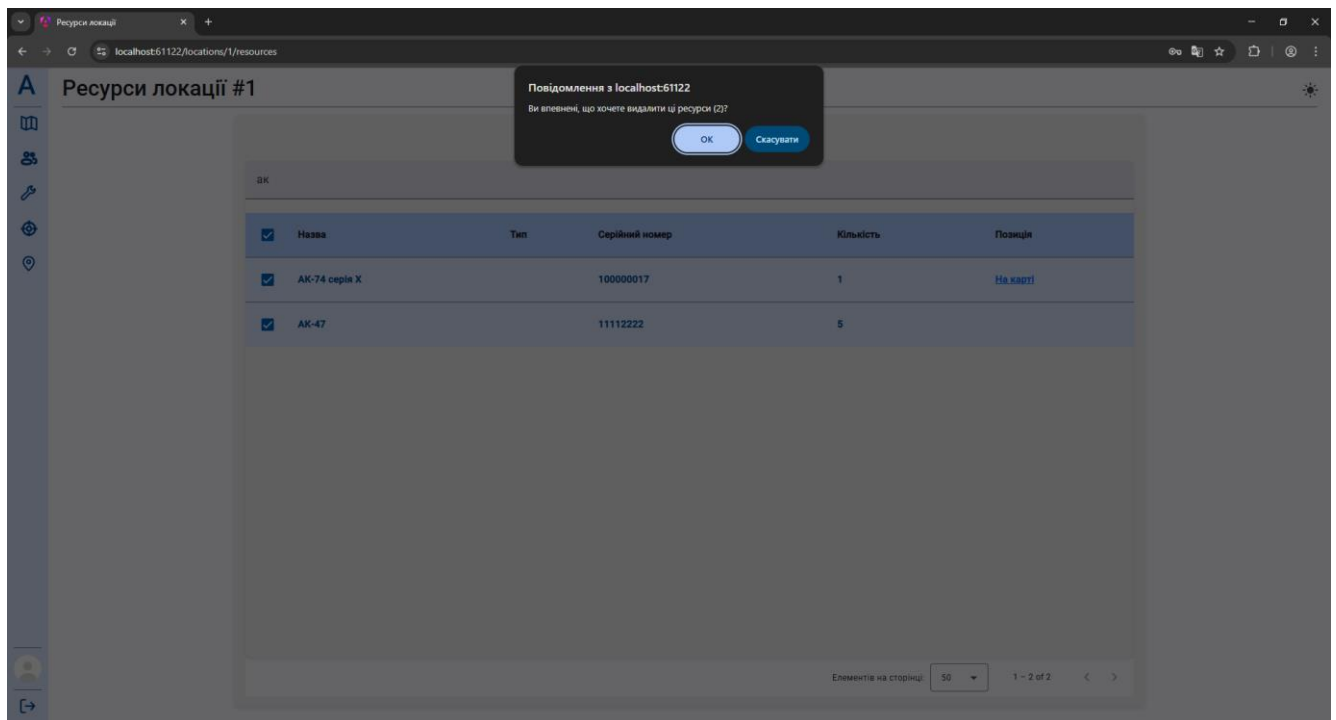


Рисунок Д-35 – Видалення ресурсів локації

Джерело: розроблено автором

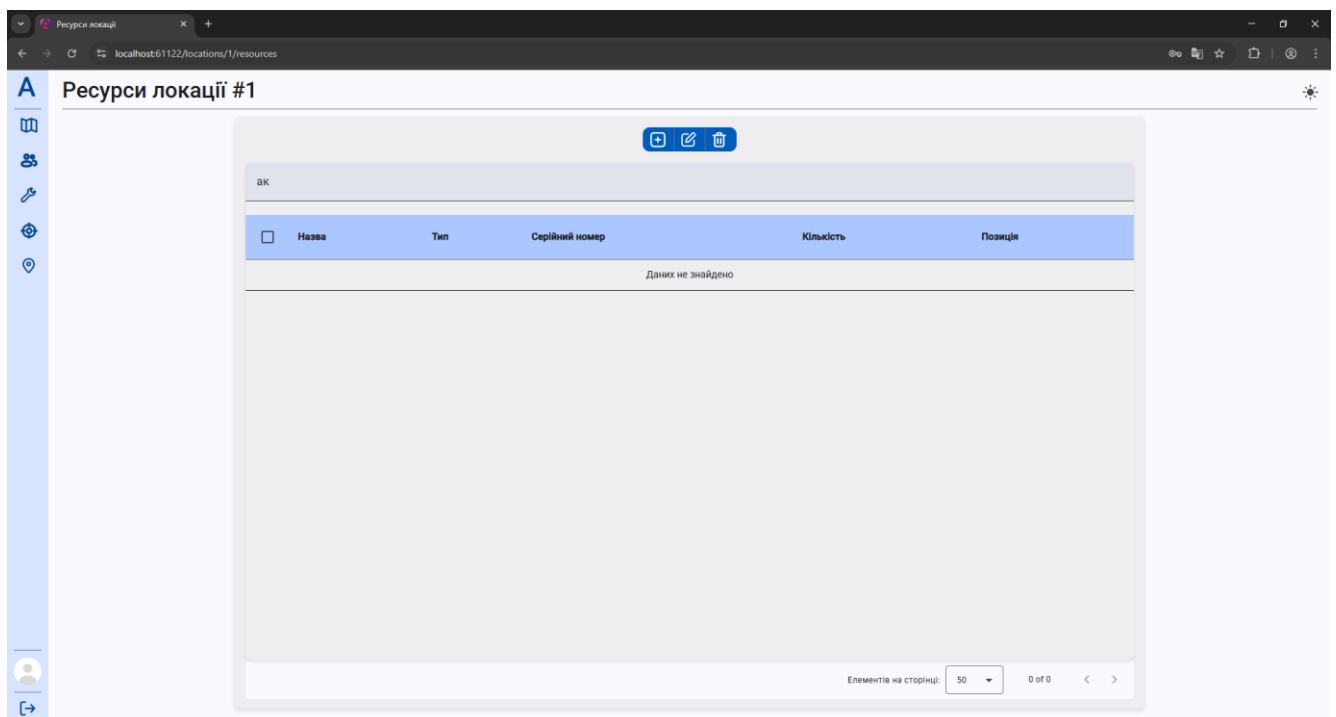


Рисунок Д-36 – Таблиця без даних для відображення

Джерело: розроблено автором

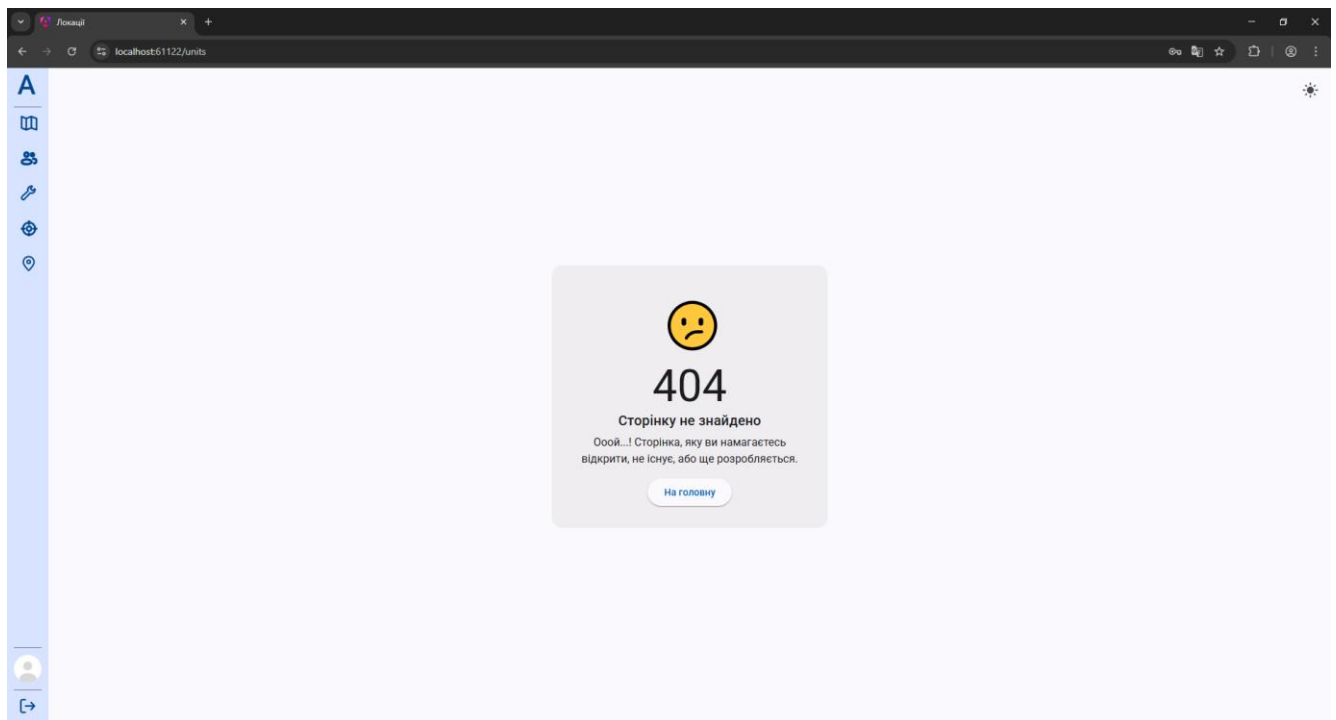


Рисунок Д-37 – Сторінка з 404 помилкою (сторінку не знайдено)

Джерело: розроблено автором

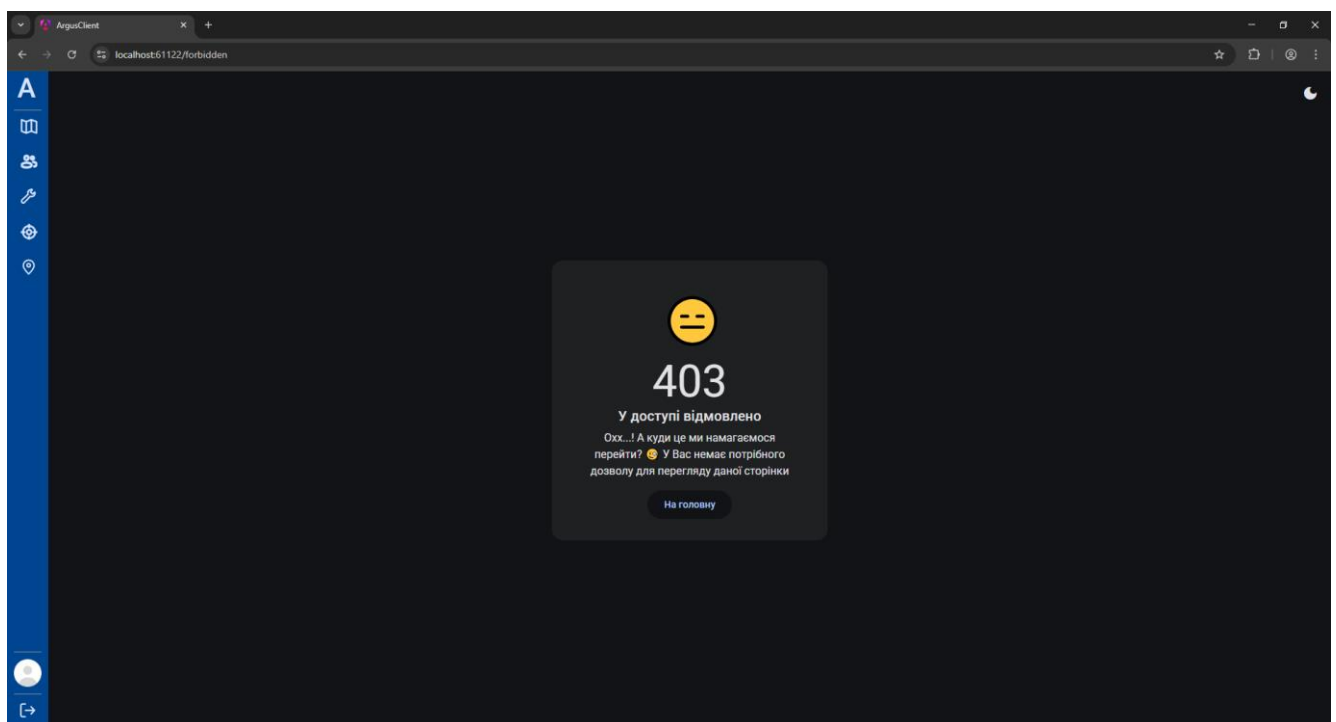


Рисунок Д-38 – Сторінка з 403 помилкою (доступ заблоковано)

Джерело: розроблено автором

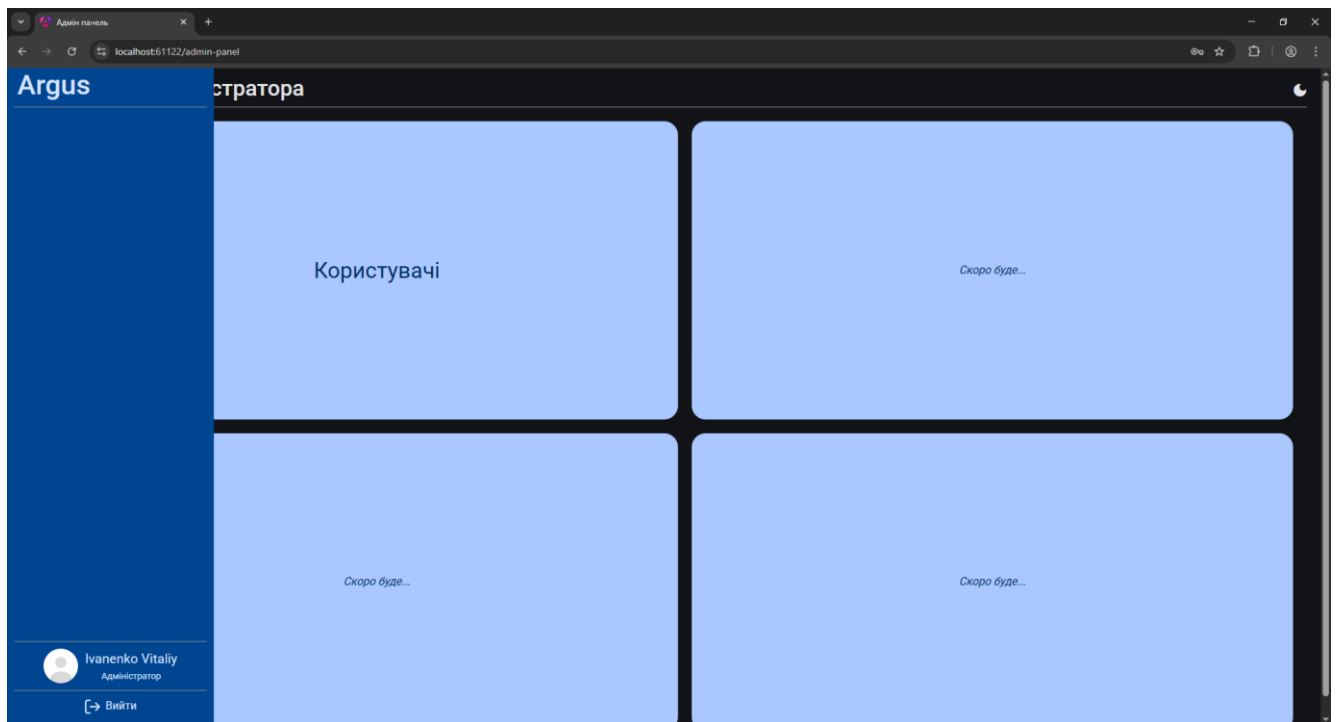


Рисунок Д-39 – Інтерфейс адміністратора

Джерело: розроблено автором

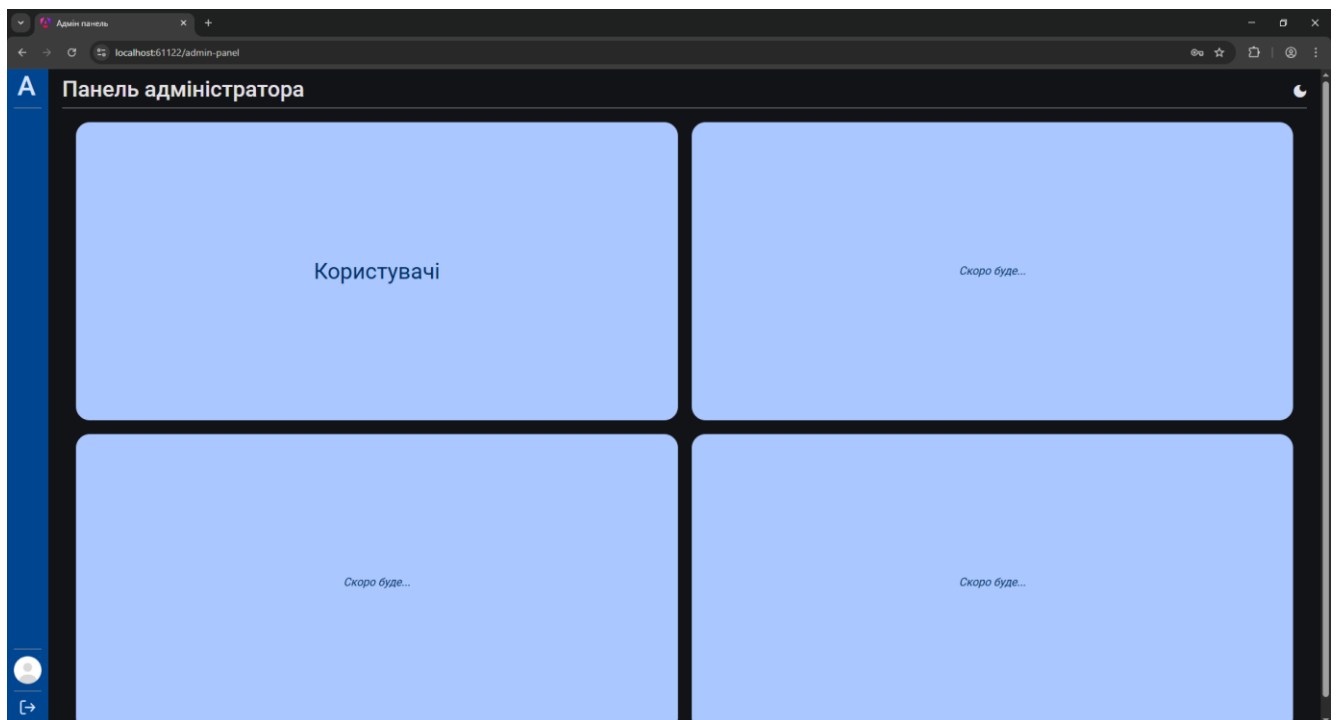


Рисунок Д-40 – Адмін-панель

Джерело: розроблено автором

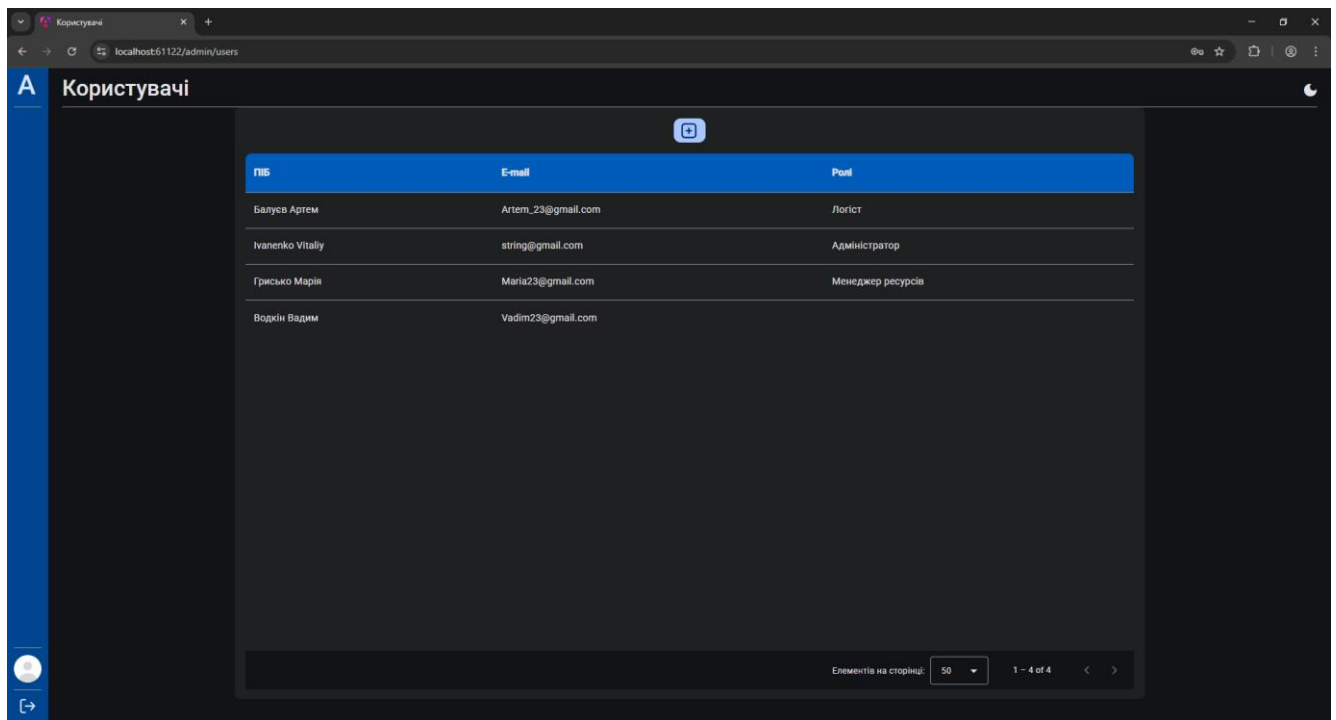


Рисунок Д-41 – Перегляд усіх користувачів

Джерело: розроблено автором

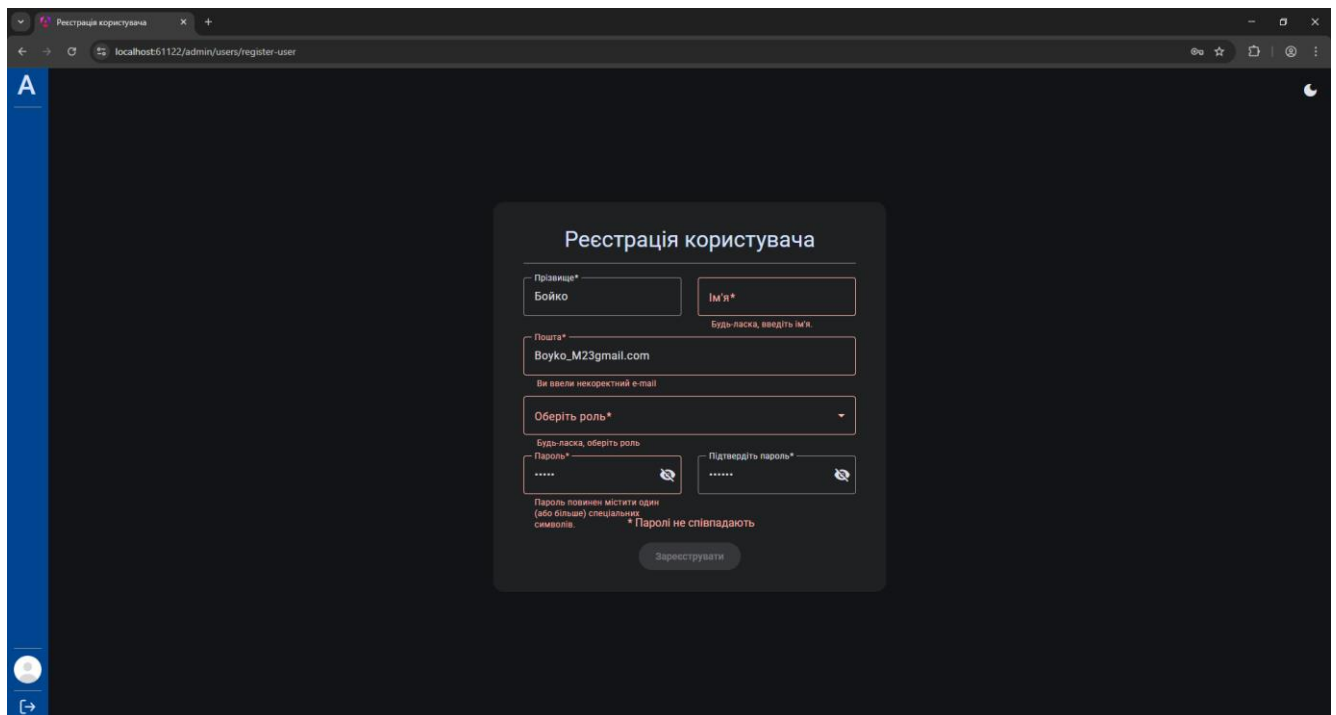


Рисунок Д-42 – Реєстрація нового користувача з помилковими даними

Джерело: розроблено автором

The screenshot shows a web browser window with the address bar displaying 'localhost:61122/admin/users/register-user'. The page has a dark theme with a blue sidebar on the left. The main content area features a registration form titled 'Реєстрація користувача'. The form includes fields for 'Прізвище*' (Last name) with the value 'Бойко', 'Ім'я*' (First name) with the value 'Марія', 'Пошта*' (Email) with the value 'Boyko_M23@gmail.com', 'Оберіть роль*' (Select role) with a dropdown menu showing 'Логіст', and two password fields 'Пароль*' and 'Підтвердіть пароль*' both containing '.....'. A 'Зареєструвати' (Register) button is at the bottom of the form.

Рисунок Д-43 – Реєстрація нового користувача

Джерело: розроблено автором

The screenshot shows the 'Користувачі' (Users) page in the web application. A green notification banner at the top states '✓ Реєстрація успішна! Нового користувача створено!' (Registration successful! New user created!). Below the banner is a table listing users. The table has three columns: 'ПІБ' (Full Name), 'E-mail', and 'Роль' (Role). The data rows are as follows:

ПІБ	E-mail	Роль
Балуєв Артем	Artem_23@gmail.com	Логіст
Ivanenko Vitaliy	string@gmail.com	Адміністратор
Грицько Марія	Maria23@gmail.com	Менеджер ресурсів
Водкін Вадим	Vadim23@gmail.com	
Бойко Марія	Boyko_M23@gmail.com	Логіст

At the bottom of the table, there is a pagination control showing 'Елементів на сторінці: 50' and '1 - 5 of 5'.

Рисунок Д-44 – Повідомлення про успішно доданого користувача.

Джерело: розроблено автором

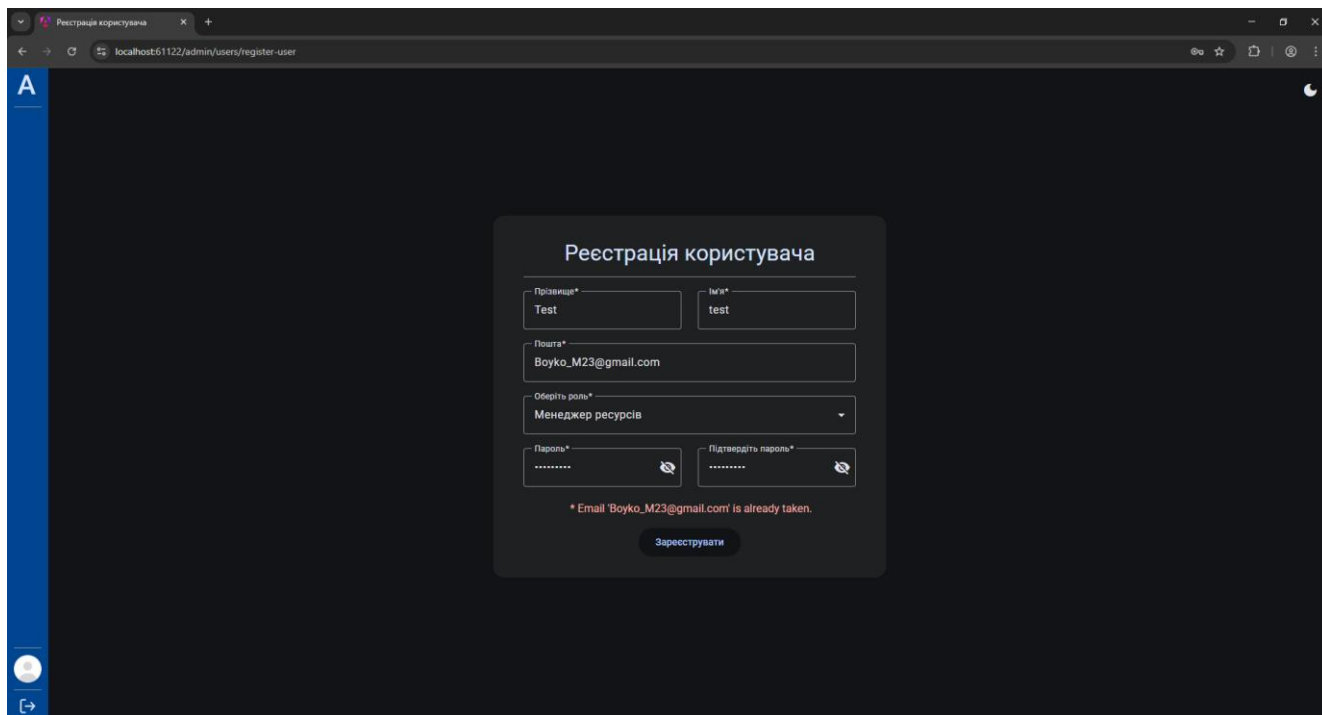


Рисунок Д-45 – Помилка при спробі створити іншого користувача з тою ж
ПОШТОЮ

Джерело: розроблено автором

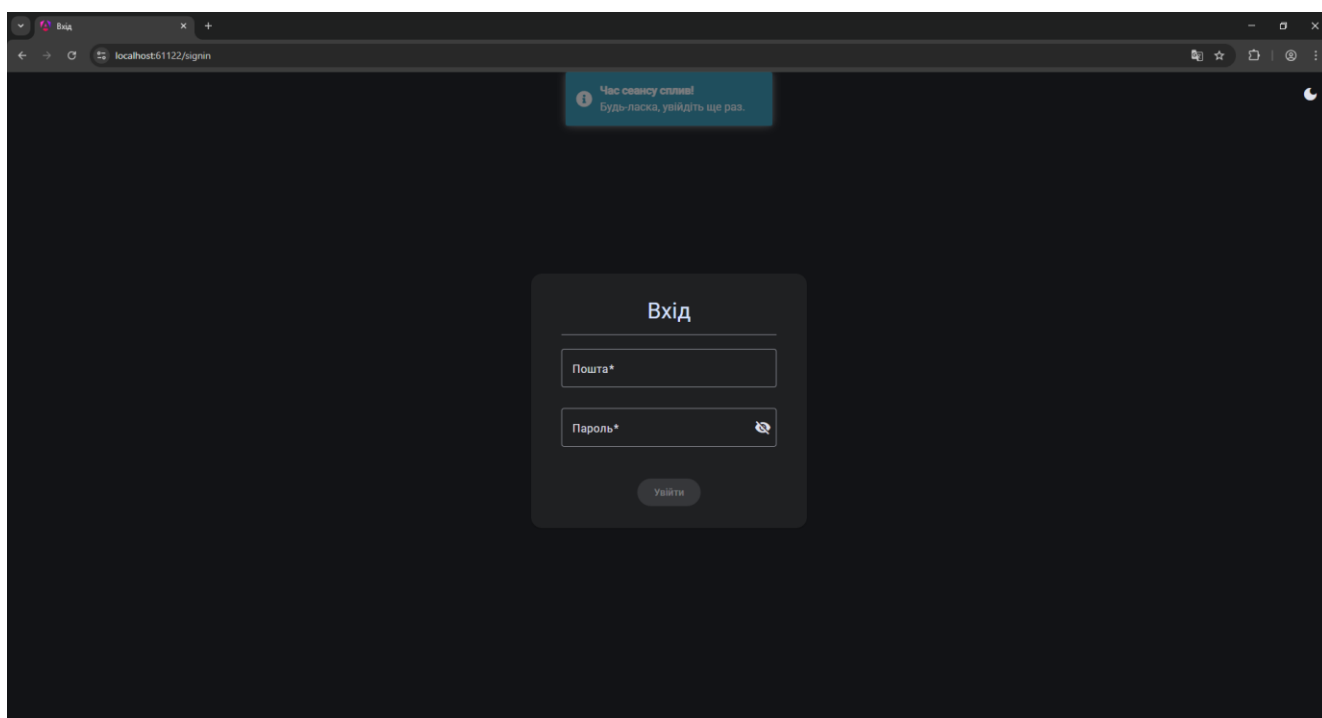


Рисунок Д-46 – Повідомлення про те, що час дії токена сплив і
користувача розлогінило

Джерело: розроблено автором



Звіт подібності

метадані

Назва організації

Kyiv National Economic University named after Vadym Hetman KNEU

Заголовок

Розроблення інформаційної системи для обліку та управління оборонними ресурсами

Автор

Науковий керівник / Експерт

Желноватенко Євгеній Анатолійович Устенко С.В.

підрозділ

кафедра інформаційних систем в економіці

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.



КП 1

25

Довжина фрази для коефіцієнта подібності 2



КП 2

9647

Кількість слів



КЦ

78727

Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв		0
Інтервали		0
Мікропробіли		4
Білі знаки		0
Парафрази (SmartMarks)		4

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз

Копіє тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://ir.kneu.edu.ua/bitstreams/725b64da-359e-479b-b2e4-50ed2d83d86c/download	17 0.18 %
2	https://ir.kneu.edu.ua/bitstreams/725b64da-359e-479b-b2e4-50ed2d83d86c/download	16 0.17 %
3	https://kneu.edu.ua/userfiles/Department_of_Administration_and_Marketing_Personn/Titulina_storD196nk_a_KMD0A0.doc	16 0.17 %

4	https://ru.essays.club/%D0%A2%D0%BE%D1%87%D0%BD%D1%8B%D0%B5-%D0%BD%D0%B0%D1%83%D0%BA%D0%B8/%D0%9A%D0%BE%D0%BC%D0%BC%D1%83%D0%BD%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D0%B8-%D0%B8-%D1%81%D0%B2%D1%8F%D0%B7%D1%8C/%D0%97%D0%B0%D1%85%D0%B8%D1%81%D1%82-%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D1%97-%D0%B7-%D0%BE%D0%B1%D0%BC%D0%B5%D0%B6%D0%B5%D0%BD%D0%B8%D0%BC-470343.html	16 0.17 %
5	https://ir.kneu.edu.ua/bitstreams/725b64da-359e-479b-b2e4-50ed2d83d86c/download	15 0.16 %
6	https://botanichik.at.ua/publ/kursovye_raboty/medzhment/sut_ta_kharakteristika_zmistovnikh_teorij_molivacii_taanaliz_jikh_zastosuvannja_na_pidpriemstvi_pat_jagotinskij_maslozavod_2013/156-1-0-793	12 0.12 %
7	https://ir.kneu.edu.ua/server/api/core/bitstreams/756bd424-c351-46ce-af3a-4a8cd094e536/content	10 0.10 %
8	https://ir.kneu.edu.ua/bitstreams/725b64da-359e-479b-b2e4-50ed2d83d86c/download	9 0.09 %
9	https://ir.kneu.edu.ua/bitstreams/725b64da-359e-479b-b2e4-50ed2d83d86c/download	8 0.08 %
10	https://ir.kneu.edu.ua/bitstreams/725b64da-359e-479b-b2e4-50ed2d83d86c/download	8 0.08 %
з бази даних RefBooks (0.00 %)		
ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
з домашньої бази даних (0.00 %)		
ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
з програми обміну базами даних (0.00 %)		
ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
з Інтернету (1.43 %)		
ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://ir.kneu.edu.ua/bitstreams/725b64da-359e-479b-b2e4-50ed2d83d86c/download	78 (7) 0.81 %
2	https://ru.essays.club/%D0%A2%D0%BE%D1%87%D0%BD%D1%8B%D0%B5-%D0%BD%D0%B0%D1%83%D0%BA%D0%B8/%D0%9A%D0%BE%D0%BC%D0%BC%D1%83%D0%BD%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D0%B8-%D0%B8-%D1%81%D0%B2%D1%8F%D0%B7%D1%8C/%D0%97%D0%B0%D1%85%D0%B8%D1%81%D1%82-%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D1%97-%D0%B7-%D0%BE%D0%B1%D0%BC%D0%B5%D0%B6%D0%B5%D0%BD%D0%B8%D0%BC-470343.html	22 (2) 0.23 %
3	https://kneu.edu.ua/userfiles/Department_of_Administration_and_Marketing_Personn/Titulna_storD196nk_a_KMD0A0.doc	16 (1) 0.17 %
4	https://botanichik.at.ua/publ/kursovye_raboty/medzhment/sut_ta_kharakteristika_zmistovnikh_teorij_molivacii_taanaliz_jikh_zastosuvannja_na_pidpriemstvi_pat_jagotinskij_maslozavod_2013/156-1-0-793	12 (1) 0.12 %
5	https://ir.kneu.edu.ua/server/api/core/bitstreams/756bd424-c351-46ce-af3a-4a8cd094e536/content	10 (1) 0.10 %

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)
------------------	-------	---------------------------------------

МАТЕРІАЛИ V МІЖНАРОДНОЇ
СТУДЕНТСЬКОЇ НАУКОВОЇ

.....

**АКТУАЛЬНІ ПИТАННЯ ТА
ПЕРСПЕКТИВИ ПРОВЕДЕННЯ
НАУКОВИХ ДОСЛІДЖЕНЬ**

.....

м. Тернопіль, Україна
6 жовтня 2023 рік



Голова оргкомітету: Коренюк І.О.

Верстка: Зрада С.І.

Дизайн: Бондаренко І.В.



Конференцію зареєстровано Державною науковою установою «УкрІНТЕІ» в базі даних науково-технічних заходів України та бюлетені «План проведення наукових, науково-технічних заходів в Україні» (Посвідчення №321 від 16.06.2023).

Матеріали конференції знаходяться у відкритому доступі на умовах ліцензії Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

А 43

.....
Актуальні питання та перспективи проведення наукових досліджень:
матеріали V Міжнародної студентської наукової конференції, м. Тернопіль, 6 жовтня, 2023 рік / ГО «Молодіжна наукова ліга». — Вінниця: ТОВ «УКРЛОГОС Груп», 2023. — 180 с.

ISBN 978-617-8126-67-4

DOI 10.36074/liga-inter-06.10.2023

Викладено матеріали учасників V Міжнародної мультидисциплінарної студентської наукової конференції «Актуальні питання та перспективи проведення наукових досліджень», яка відбулася 6 жовтня 2023 року у місті Тернопіль, Україна.

УДК 082:001

© Колектив учасників конференції, 2023

© ГО «Молодіжна наукова ліга», 2023

© ТОВ «УКРЛОГОС Груп», 2023

ISBN 978-617-8126-67-4

СЕКЦІЯ 2.
**ФІНАНСИ ТА БАНКІВСЬКА СПРАВА;
ОПОДАТКУВАННЯ, ОБЛІК І АУДИТ**

THEORETICAL AND METHODOLOGICAL FOUNDATION OF INVESTMENT
PRACTICES IN NON-BANK FINANCIAL INSTITUTIONS IN UKRAINE

Nevmerzhytska V., Scientific supervisor: Vavdiychik I. 19

БАНКІВСЬКА ДІЯЛЬНІСТЬ: СУЧАСНІ ТЕНДЕНЦІЇ

Полянська О.Л., Науковий керівник: Нечипоренко Т.Д. 22

БЮДЖЕТНИЙ ПРОЦЕС В УКРАЇНІ: ОСОБЛИВОСТІ ОРГАНІЗАЦІЇ ТА ШЛЯХИ
УДОСКОНАЛЕННЯ

Казімір К.В., Науковий керівник: Нечипоренко Т.Д. 25

ВПЛИВ ПРЯМИХ ПОДАТКІВ НА ФІНАНСОВИЙ СТАН ПІДПРИЄМСТВА

Венгер К.С., Науковий керівник: Нечипоренко Т.Д. 29

ОСНОВНІ ЗМІНИ ОБЛІКОВОЇ ПОЛІТИКИ ПІДПРИЄМСТВА В УМОВАХ
ВОЄННОГО СТАНУ

Огребчук В.А., Науковий керівник: Шот А.П. 32

ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ УПРАВЛІННЯ
ОСОБИСТИМИ ФІНАНСАМИ

Желноватенко Є.А., Науковий керівник: Устенко С.В. 35

ЦИФРОВІЗАЦІЯ БАНКІВСЬКИХ ОПЕРАЦІЙ

Барабаш Є.Є., Науковий керівник: Нечипоренко Т.Д. 38

Желноватенко Євгеній Анатолійович, здобувач вищої освіти
інституту інформаційних технологій в економіці
Київський національний економічний університет ім. Вадима Гетьмана, Україна

Науковий керівник: Устенко Станіслав Веніамінович, д-р. екон. наук,
професор кафедри інформаційних систем в економіці
Київський національний економічний університет ім. Вадима Гетьмана, Україна

ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСАМИ

В сучасному інформаційному суспільстві, яке постійно розвивається, питання ефективного управління особистими фінансами стає ключовим завданням для індивідуальних користувачів. Відправною точкою цього дослідження є розуміння того, що краще керувати, коли ти знаєш. Важливість цього завдання виражається у зростаючій складності фінансового середовища та необхідності ефективного аналізу та прийняття обґрунтованих фінансових рішень.

Хоча назва «особистий фінансовий менеджмент» часто використовується для позначення способів управління вашими особистими фінансами, це також фактичний термін, часто відомий під аббревіатурою, PFM (personal financial management) [1], і відноситься до типу програмного забезпечення, що використовується для застосунків про особисті фінанси. PFM існує з 1983 року, коли співзасновник Intuit Скотт Кук втомився спостерігати, як його дружина бореться з усіма документами, пов'язаними з оплатою рахунків вручну.

Він почав вивчати, як спростити або автоматизувати процес. Під час розвішування повідомлення про пошук програміста в Стенфордському університеті він випадково зіткнувся з Томом Праулксом, який вже писав програми до цього, і той погодився допомогти Куку і написати програму для балансування чеків, яка стала ранньою версією програми особистих фінансів Quicken.

Проаналізуємо список популярних систем для управління особистими фінансами та порівняємо деякі з них (див. табл. 1) [2].

Таблиця 1

Порівняння систем для управління особистими фінансами

	Monefy	Money Manager Expense & Budget
Український інтерфейс	+	+
Синхронізація на різних пристроях	+	+
Підключення до банківських рахунків	-	-
Підтримка декількох рахунків	+	+
Створення рахунків з різними валютами	+	+
Наявність платної версії	+	+
Зрозумілий інтерфейс	+	+
Перегляд статистики у вигляді графіку	-	+
Велика кількість реклами	+	-

Можна визначити головні функції, які повинні бути наявними у системі, що проектується. Важливим є реалізація тих функцій, які не наявні у окремих програмах. Це має надати системі унікальності, яка виведе її вище за конкурентів.

У таблиці порівняння можна побачити, що жодна з систем не має можливості підключення до банківських рахунків. Така ситуація склалася через те, що, так званий Open Banking (з англ. «Відкритий банкінг»), не працює в Україні[3].

Ідея відкритого банкінгу з'явилася менше ніж 10 років тому, але вже сьогодні мільйони людей в Європі використовують продукти, що базуються на концепції Open Banking. Наприклад, у Великій Британії більше половини споживачів хоча б раз скористалися такими послугами.

В Україні перші важливі кроки в напрямку впровадження відкритого банкінгу були зроблені Національним банком ще влітку 2019 року, і через два роки парламент прийняв необхідне законодавство в цьому напрямку.

Відкритий банкінг (Open Banking) – це концепція, яка включає в себе вільний доступ до банківських даних третім сторонам, зазвичай за допомогою API. Іншими словами, ви ділитеся своєю особистою інформацією, такою як баланс і історія транзакцій, з іншими постачальниками фінансових послуг, а відповідно отримуєте можливість швидко та безпечно оплачувати різні товари та послуги, користуватися інноваційними фінансовими продуктами тощо. Наприклад, ви можете використовувати один мобільний додаток для керування своїми фінансами, який об'єднує дані з усіх ваших банківських рахунків.

Система відкритого банкінгу передбачає високі стандарти безпеки для обміну інформацією між фінансовими компаніями та третіми сторонами. Тому немає необхідності переживати, що ваші особисті дані будуть використані без дозволу.

В багатьох країнах відкритий банкінг стає все більш популярним, і регулятори часто сприяють його розвитку, оскільки це сприяє інноваціям у банківській та фінансовій галузях. Це відкриває більше можливостей для фінтех-компаній розробляти нові продукти та послуги, які використовують дані з різних банківських джерел.

Фахівці вважають, що відкритий банкінг – це позитивний крок для галузі фінансових послуг. Це корисно як для бізнесу, так і для кінцевих споживачів, і обіцяє зробити фінансові послуги більш доступними, прозорими та зручними.

Приступимо до проектування та розробки системи. Для початку створимо UML-діаграму компонентів (див. рис. 1), яка буде відображати основні компоненти системи та їх взаємозв'язок.

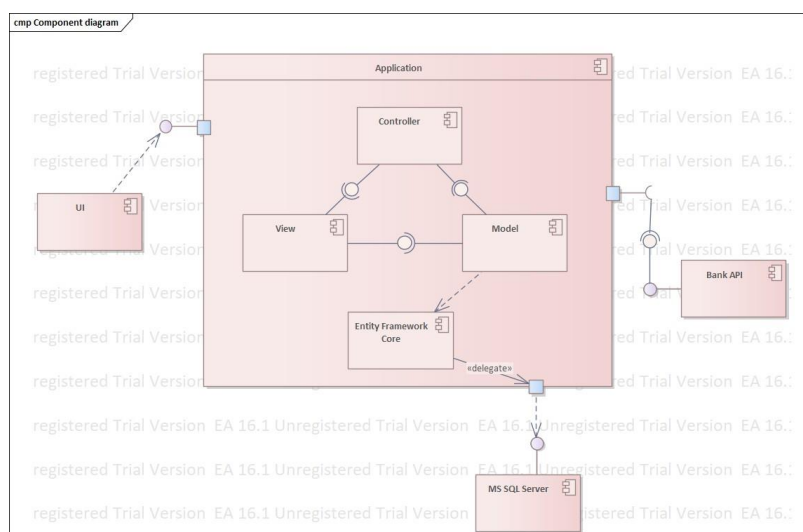


Рис. 1. Діаграма компонентів системи. Джерело [5]

Для розробки даної системи було обрано клієнт-серверну архітектуру. Вона зможе забезпечити гарну масштабованість, надасть можливість створення додатків для різних платформ (таких як веб, мобільні пристрої та настільні комп'ютери), а також завдяки ній ми зможемо легко тестувати та супроводжувати окремі компоненти системи, такі як сервер, окремо від клієнтських додатків.

Таким чином, клієнт-серверна архітектура є ефективним та гнучким варіантом для системи управління особистими фінансами, оскільки вона дозволяє поєднувати в собі ефективність роботи, безпеку, масштабованість та зручний користувацький досвід для користувачів різних платформ.

Висновки. Провівши дослідження існуючих систем управління особистими фінансами, можна сказати, що на ринку вже існують непогані рішення. Але кожне з них має свої переваги та недоліки. При розробці власної системи варто ставити за ціль переймання та вдосконалення усіх плюсів, недопускання мінусів та додавання нового. Прихід відкритого банкінгу (Open Banking) в Україну відкриє багато можливостей, і основною задачею буде представити на ринок продукт для управління особистими фінансами, який зможе їх використовувати.

Список використаних джерел:

1. Tymkiw C. What Is Personal Finance Management (PFM), How Is It Used? [Електронний ресурс] / Catherine Tymkiw. – 2022. – Режим доступу до ресурсу: <https://www.investopedia.com/personal-financial-management-pfm-5181311>. (дата звернення: 20.09.2023).
2. Ареф'єва І. Домашня бухгалтерія – 7 найкращих програм [Електронний ресурс] / Ірина Ареф'єва. – 2023. – Режим доступу до ресурсу: <https://financer.com/ua/blog/best-apps/>. (дата звернення: 23.09.2023).
3. Open Banking: що це і як вплине на фінтех-ринок України [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: <https://fondy.ua/uk/knowledge/open-banking/>. (дата звернення: 23.09.2023).
4. Money Manager [Електронний ресурс] – Режим доступу до ресурсу: <https://realbyteapps.com/>. (дата звернення: 23.09.2023).
5. What is Component Diagram? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/>. (дата звернення: 24.09.2023).

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Київський національний економічний університет імені Вадима Гетьмана
Ярославська Державна Вища Технічно-Економічна Школа
ім. о. Броніслава Маркевича (Польща) Академія
фінансів та бізнесу Vistula (Польща)
Європейський університет у Варшаві (Польща)
Компанія Pro Insight sp. z o.o. (Польща)
Вроцлавський університет економіки та бізнесу (Польща)
Інститут кібернетики імені В.М. Глушкова НАН України
Київський національний університет імені Тараса Шевченка
Національний технічний університет України «КПІ ім. Ігоря Сікорського»
Харківський національний економічний університет ім.Семена Кузнеця
Національний технічний університет «ХПІ»
Харківський національний університет імені В. Н. Каразіна
Харківський національний університет радіоелектроніки
Харківський національний університет міського господарства ім. О.М. Бекетова
Херсонський національний технічний університет
Запорізький національний університет
Хмельницький національний університет
Криворізький державний педагогічний університет
Східноукраїнський національний університет ім. Володимира Даля
Львівський національний університет імені Івана Франка
Львівський національний технічний університет «Львівська політехніка»
Кременчуцький національний університет ім. М. Остроградського

СУЧАСНІ ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ТА СИСТЕМИ В УПРАВЛІННІ

Збірник матеріалів

**V Міжнародної науково-практичної конференції
молодих вчених, аспірантів і студентів
(Інтернет-конференція)**

18–19 квітня 2024 р.

Сіденко Д.С. КОНЦЕПЦІЯ РОЗВИТКУ ШІ У ДЕРЖАВНОМУ СЕКТОРІ	231
Скіцько В.І., Шевченко Д. С., Індик О.С. ПЕРСОНАЛІЗАЦІЯ ОСВІТНЬОГО ПРОЦЕСУ ЗА ДОПОМОГОЮ ШТУЧНОГО ІНТЕЛЕКТУ	233
Скіцько В.І., Корж Є.С. ТЕОРЕТИЧНІ АСПЕКТИ ФУНКЦІОНУВАННЯ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ	235
Скіцько В.І., Мохналь А.В. GPT ЯК АКТУАЛЬНА МОДЕЛЬ ШТУЧНОЇ НЕЙРОННОЇ МЕРЕЖІ	237
Спаська П.Р. КЛАСТЕРНИЙ АНАЛІЗ РЕГІОНІВ УКРАЇНИ ЗА СКЛАДОВИМИ РІВНЯ ЕКОНОМІЧНОЇ БЕЗПЕКИ	239
Степаненко Ю.О., Скіцько В.І. АСПЕКТИ РОЗВИТКУ ШІ-ТЕХНОЛОГІЙ У СФЕРІ ЕМОЦІЙНОЇ БЕЗПЕКИ	241
Степаненко Ю.О. ОПТИМІЗАЦІЯ РОБОЧИХ ПРОЦЕСІВ ЗА ДОПОМОГОЮ ТЕХНОЛОГІЙ РОЗПІЗНАВАННЯ ЕМОЦІЙ	244
Титарчук О. М. ВИКОРИСТАННЯ ШТУЧНОГО ІНТЕЛЕКТУ В ІНФОРМАЦІЙНИХ УПРАВЛІНСЬКИХ СИСТЕМАХ	246
Устенко С.В., Желноватенко Є.А, Павлюк Д.А. ВИКОРИСТАННЯ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ОПТИМІЗАЦІЇ БІЗНЕС-ПРОЦЕСІВ: МОЖЛИВОСТІ ТА ПЕРСПЕКТИВИ	248
Устенко С.В., Терентьєв Є.Г. ІННОВАЦІЙНІ ПІДХОДИ ДЛЯ ТАРГЕТОВАНОГО ПРОСУВАННЯ НАВЧАЛЬНОГО КОНТЕНТУ НА ОСНОВІ ШІ	250
Філоненко І. О. РОЗВИТОК РОБОТЕХНІКИ В УКРАЇНІ	252
Хорканін М. Ю. РОЗПОДІЛЕНІ ІНТЕЛЕКТУАЛЬНІ СИСТЕМИ: ЩО РОБИТЬ ЇХ "РОЗУМНИМИ"	254
Чабаненко О. С. ВПЛИВ ШТУЧНОГО ІНТЕЛЕКТУ НА УПРАВЛІННЯ ЛЮДСЬКИМ КАПІТАЛОМ	255
Шведов Д. В. РОЗРОБКА МОДУЛІВ ОБРОБКИ ДАНИХ ТА ПОБУДОВИ МОДЕЛІ ШТУЧНОЇ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ ОЦІНКИ ТА ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ ФІНАНСОВИХ РИЗИКІВ	257

Устенко С.В.
д.е.н., професор
Желноватенко Є.А.
Павлюк Д.А.
магістри

*Київський національний економічний
університет імені Вадима Гетьмана*
stasustenko@ukr.net
380970319989d42@gmail.com
mr.zhelnovatenko@gmail.com

ВИКОРИСТАННЯ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ОПТИМІЗАЦІЇ БІЗНЕС-ПРОЦЕСІВ: МОЖЛИВОСТІ ТА ПЕРСПЕКТИВИ

ШІ трансформує традиційні підходи до управління, оптимізації та автоматизації бізнес-процесів, пропонуючи значні переваги з точки зору ефективності, зниження витрат та покращення сервісу. В цьому дослідженні проаналізуємо, як інтеграція ШІ може сприяти створенню інноваційних бізнес-моделей та покращити процеси в різних галузях, від фінансів до виробництва, а також які виклики можуть виникнути на цьому шляху.

ШІ відкриває нові можливості для реалізації бізнес-моделей, які оптимізують використання ресурсів, мінімізують витрати та максимізують доходи через автоматизацію та здатність до аналізу великих обсягів даних. Дослідження [1] підкреслює, як ШІ може допомогти в розробці нових, більш адаптивних бізнес-структур, здатних швидко реагувати на змінні ринкові умови.

Оптимізація цифрових ланцюжків поставок за допомогою ШІ передбачає використання комплексних алгоритмів для планування та логістики, що дозволяє компаніям зменшувати простой, забезпечувати своєчасне надходження товарів та ефективніше управляти запасами. В роботі [2] наголошують на значенні математичного програмування в структуризації даних і управлінні операціями, що критично важливо для комплексних ланцюжків поставок.

Стратегічна адаптація до впровадження ШІ вимагає від компаній не тільки технічної готовності, але й здатності інтегрувати нові технології. В роботі модель [3] демонструє, як компанії можуть використовувати ШІ для інновацій у бізнес-моделях та організаційних процесах, підкреслюючи необхідність гнучкого підходу до управлінських змін.

Ці дослідження ілюструють, як ШІ може бути використаний для реалізації інноваційних бізнес-моделей та оптимізації бізнес-процесів, водночас вказуючи на складність інтеграції та потенційні виклики.

ШІ може значно підвищити точність фінансових прогнозів та рішень за допомогою автоматизації та обробки великих обсягів даних, що включає управління активами, кредитним ризиком та ринковими операціями, покращуючи тим самим операційну ефективність.

Зниження оперативних витрат можливе через автоматизацію процесів, таких як обробка запитів клієнтів та транзакцій, що сприяє підвищенню клієнтської сатисфакції через персоналізацію фінансових продуктів і поліпшенню управління ризиками завдяки передовим аналітичним здібностям ШІ.

Використання ШІ також має свої виклики, включаючи проблеми з безпекою даних та приватністю в контексті залежності від цифрових технологій, високі витрати на імплементацію та інтеграцію ШІ в існуючі системи, а також потенційні ризики, пов'язані з технологічними збоями або кібератаками.

Банки можуть розробляти платформи ШІ, які будуть надаватися як послуги іншим меншим фінансовим установам, інтегрувати традиційні банківські послуги з інноваційними ШІ-рішеннями для створення більш адаптивних та ресурсоощадних сервісів та запроваджувати моделі підписки для корпоративних клієнтів, зацікавлених у використанні спеціалізованих ШІ-інструментів.

Очікувані результати. Дослідження передбачає, що впровадження штучного інтелекту (ШІ) в банківській сфері принесе значне підвищення ефективності операцій, зокрема через автоматизацію стандартних процедур, як-от кредитування та обслуговування клієнтів. Очікується, що використання ШІ дозволить банкам надавати більш персоналізовані фінансові продукти та послуги, що покращить клієнтський досвід і збільшить лояльність користувачів. Також передбачається, що ШІ значно покращить управління фінансовими ризиками, дозволяючи банкам ефективніше ідентифікувати та управляти потенційними втратами. Однак дослідження також має показати можливі технічні та безпекові виклики, пов'язані з інтеграцією ШІ, включаючи високі витрати на впровадження, необхідність адаптації існуючих систем і потенційні ризики, що стосуються приватності даних.

Висновки. На основі дослідження можна очікувати, що впровадження ШІ дозволить банкам значно покращити їхні бізнес-процеси, знизити операційні витрати та підвищити рівень задоволеності клієнтів. Однак, банки повинні також враховувати потенційні ризики та виклики, асоційовані з технологічною залежністю та приватністю даних. Ефективна інтеграція ШІ вимагатиме не тільки значних фінансових інвестицій, але й стратегічного планування, освіти персоналу та розвитку внутрішньої корпоративної культури, що сприймає інновації та технологічні зміни.

Перспективи для майбутнього дослідження. Перспективи для майбутнього дослідження включають глибокий аналіз та осмислення способів крос-функціональної інтеграції ШІ, зокрема його взаємодії з іншими новітніми технологіями, як-от Інтернет речей (IoT) та великі дані, що може відкрити нові шляхи для розуміння і вдосконалення бізнес-процесів. Цей підхід дозволить не лише оптимізувати існуючі процеси, але й розробляти нові моделі взаємодії та управління в рамках банківської сфери.

Список використаних джерел

1. Emerging Technology and Business Model Innovation: The Case of Artificial Intelligence [Електронний ресурс] / J.Lee, T. Suh, D. Roy, M. Baucus. – 2019. – Режим доступу до ресурсу: <https://www.sciencedirect.com/science/article/pii/S2199853122009817>.
2. Optimization of extended business processes in digital supply chains using mathematical programming [Електронний ресурс] / [H. Perez, S. Amaran, E. Erisen та ін.]. – 2021. – Режим доступу до ресурсу: <https://www.sciencedirect.com/science/article/abs/pii/S0098135421001010>.
3. AI-enabled business-model innovation and transformation in industrial ecosystems: A framework, model and outline for further research [Електронний ресурс] / T.Burström, V. Parida, T. Lahti, J. Wincent. – 2021. – Режим доступу до ресурсу: <https://www.sciencedirect.com/science/article/abs/pii/S0148296321000163>.
How AI capabilities enable business model innovation: Scaling AI through co-evolutionary processes and feedback loops [Електронний ресурс] / D.Sjödin, V. Parida, M. Palmié, J. Wincent. – 2021. – Режим доступу до ресурсу: <https://www.sciencedirect.com/science/article/pii/S0148296321003386>