МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Навчально-науковий інститут інформаційних технологій та бізнесу
Кафедра інформаційних технологій та аналітики даних

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавра

на тему: «**Розробка модуля групового підбору житла з вбудованим чат-сервісом для платформи "Пошук житла в Острозі"**»

Виконав: студент 4 курсу, групи КН-42
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»
Пухальський Владислав Вадимович

Керівник: доктор філософії з прикладної математики,
старший викладач кафедри інформаційних технологій та аналітики даних, фахівець-практик
Красюк Богдан Віталійович

Рецензент: *кандидат технічних наук, доцент,*
доцент кафедри прикладної математики
Донецького національного університету
імені Василя Стуса
Загоруйко Любов Василівна

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ
Завідувач кафедри інформаційних технологій та аналітики даних _____ (проф., д.е.н. Кривицька О.Р.)
Протокол № 11 від 20 травня 2026 р.

Острог, 2026

АНОТАЦІЯ
кваліфікаційної роботи
на здобуття освітнього ступеня бакалавра

Тема: Розробка модуля групового підбору житла з вбудованим чат-сервісом для платформи "Пошук житла в Острозі"

Автор: Пухальський Владислав Вадимович

Науковий керівник: *доктор філософії з прикладної математики, викладач, фахівець-практик, Красюк Богдан Віталійович*

Захищена «……»…… 20__ року.

Пояснювальна записка до кваліфікаційної роботи: 70 с., 14 рис., 2 табл., 1 додаток, 22 джерела.
Ключові слова: *вебзастосунок, платформа, додаток, бекенд, C#, .NET, FastAPI, робота команди, система чату, система пошуку житла для студентів Острога.*

Короткий зміст праці:

Кваліфікаційна робота присвячена проектуванню та розробці спеціалізованої платформи «Пошук житла в Острозі» (UniStay) з фокусом на інноваційний модуль спільної оренди. Робота включає побудову масштабованого бекенду на базі платформи .NET 8 з використанням принципів Clean Architecture та бази даних PostgreSQL. Було реалізовано надійну систему авторизації на основі JWT-токенів, валідацію вхідних даних через FluentValidation, структуровану маршрутизацію REST API та ефективну систему управління оголошеннями. Крім того, інфраструктура бази даних оптимізована за допомогою Entity Framework Core для швидкої обробки складних реляційних зв'язків.

Особлива увага була приділена розробці модуля групового підбору житла та вбудованого чат-сервісу, що функціонує в режимі реального часу завдяки технології SignalR. Алгоритм роботи модуля дозволяє користувачам не лише індивідуально шукати нерухомість, а й об'єднуватися в групи майбутніх співмешканців. Ініціатор пошуку може створити спільний чат з орендодавцем і додати туди інших учасників, що дозволяє всій групі одночасно вести комунікацію з орендодавцем, обговорювати умови та приймати спільні рішення. Ця логіка базується на чіткому розподілі ролей у чатах (власник, адміністратор, учасник) та інкапсульованих правилах доступу, розроблених за підходом Domain-Driven Design. Система є гнучкою, легко масштабується та повністю готова для реального впровадження в студентське середовище міста Острог.

(підпис автора)

ANNOTATION
of qualification paper
for bachelor's degree

Theme: *Development of a group housing selection module with a built-in chat service for the "Housing Search in Ostroh" platform*

Author: *Puhalskyi Vladyslav*

Scientific supervisor: *Doctor of Philosophy in Applied Mathematics, Lecturer, Practitioner Specialist, Bohdan Vitaliyovych Krasiuk*

Defended on «……»…… 20__.

Explanatory note to the qualification work: 70 pages, 14 figures, 2 tables, 1 appendix, 22 sources.

Keywords: *web application, platform, application, backend, C#, .NET, FastAPI, teamwork, chat system, housing search system for students in Ostroh.*

Brief summary of the work:

The qualification work is dedicated to the design and development of a specialized platform "Housing Search in Ostroh" (UniStay) with a focus on an innovative shared rental module. The work includes building a scalable backend based on the .NET 8 platform using the principles of Clean Architecture and the PostgreSQL database. A reliable authorization system based on JWT tokens, input data validation via FluentValidation, structured REST API routing, and an efficient ad management system were implemented. In addition, the database infrastructure is optimized using Entity Framework Core for fast processing of complex relational relationships.

Particular attention was paid to the development of a group housing selection module and a built-in chat service that operates in real time thanks to SignalR technology. The module's algorithm allows users not only to individually search for real estate, but also to unite into groups of future roommates. The search initiator can create a joint chat with the landlord and add other participants to it, which allows the entire group to simultaneously communicate with the landlord, discuss conditions and make joint decisions. This logic is based on a clear distribution of roles in the chats (owner, administrator, participant) and encapsulated access rules developed using the Domain-Driven Design approach. The system is flexible, easily scalable and fully ready for real implementation in the student environment of the city of Ostroh.

(author's signature)

ЗМІСТ

ВСТУП

У сучасному цифровому суспільстві процеси урбанізації та освітньої міграції зумовлюють стабільно високий попит на ринку оренди нерухомості. Для студентів та молодих спеціалістів пошук комфортного та доступного житла є одним із першочергових і водночас найскладніших завдань. Із розвитком інтернет-технологій платформи для пошуку нерухомості стали основним інструментом взаємодії між орендодавцями та потенційними орендарями.

Проте традиційні вебсервіси з пошуку житла здебільшого орієнтовані на індивідуального загального користувача та функціонують за принципом звичайної дошки оголошень по всій країні, хоча й можна виставити певні фільтри, для знаходження житла в окремому місті, але на ділі ми стикаємось з іншими проблеми, а саме не можливість вести повноцінний груповий діалог, тому що на практиці ж студенти часто стикаються з необхідністю спільної оренди для оптимізації фінансових витрат. Розрізненість комунікації, коли пошук об'єкта відбувається на одній платформі, обговорення з майбутніми сусідами — у сторонніх месенджерах, а перемовини з орендодавцем — по телефону, створює значні незручності та уповільнює процес ухвалення рішень. Друге проблема це те що ми можемо стикатись на велику кількість оголошень які нас не цікавлять, для прикладу продаж житла. Наша платформа в першу чергу розрахована на студентів, які будуть приїжджати в місто на період навчання, ці студенти будуть вчитись та випускатись, і на їх місце будуть приходити нові, тому актуальність платформи є великою, а також стабільною, щоб юнаки могли бачити усі актуальні варіанти на одній платформі, не розриваючись між мессенджерами, додатками, та соціальними мережами.

Ефективне розв'язання цієї проблеми вимагає розробки комплексного програмного рішення, яке б об'єднувало каталог нерухомості та інтерактивний простір для групової комунікації. У цьому контексті актуальним є створення спеціалізованої платформи, де користувачі можуть створювати групові чати з орендодавцем, вести з ними пряму

переписку в єдиному інформаційному середовищі. Орендодавець в свою чергу також буде одразу бачити усіх претендентів на житло, що значно полегшує комунакцію, бо не треба переключатись між різними чатами в мессенджері, і здогадуватись чи це ті самі орендарі, чи інші. Все одразу з самого початку в одному чаті, ніхто не передає один одному якусь інформацію, не потрібно на час знаходження житла створювати окремі групи

Метою даної кваліфікаційної роботи є проєктування та розробка модуля групового підбору житла з вбудованим чат-сервісом для платформи «Пошук житла в Острозі» (UniStay). Рішення базується на використанні підходу Clean Architecture та платформи .NET 8, що забезпечує створення масштабованої, безпечної та високонавантаженої системи взаємодії користувачів у реальному часі.

Для досягнення поставленої мети необхідно виконати такі **основні завдання**:

- провести аналіз предметної області ринку оренди нерухомості та існуючих програмних рішень;
- спроєктувати гнучку архітектуру програмної системи з використанням підходу Domain-Driven Design (DDD) та розділенням на незалежні шари (Clean Architecture);
- спроєктувати та оптимізувати реляційну базу даних PostgreSQL для збереження оголошень, користувацьких даних та історії повідомлень;
- розробити логіку модуля групового підбору житла з підтримкою системи ролей у чатах (власник групи, адміністратор, учасник);
- реалізувати REST API для управління контентом та інтегрувати технологію SignalR для забезпечення обміну повідомленнями в режимі реального часу;
- забезпечити тестування та перевірку працездатності архітектурних рішень системи на практиці.

Об’єкт дослідження: процеси та інформаційні системи автоматизації пошуку нерухомості, а також організація комунікації між користувачами вебплатформ.

Предмет дослідження: методи проєктування, архітектурні рішення (Clean Architecture) та програмні технології (.NET, SignalR, REST API) для розробки модуля групового підбору житла з інтегрованим чат-сервісом.

Таким чином, реалізація власного проєкту «UniStay» не лише дозволить застосувати набуті інженерні знання на практиці, а й розв'яже реальну соціальну проблему студентів міста Острог, надаючи їм інноваційний, зручний та централізований інструмент для спільного пошуку житла та безпечної комунікації з орендодавцями.

РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ ПЛАТФОРМИ ДЛЯ СПІЛЬНОЇ ОРЕНДИ ЖИТЛА

1.1. Постановка проблеми пошуку та спільної оренди житла в студентському середовищі

З розвитком освітньої міграції та процесами урбанізації проблема пошуку житла стає одним із найважливіших викликів для студентів. У таких студентських містах, як Острог, попит на оренду нерухомості має яскраво виражений сезонний характер, а пропозиція часто є обмеженою. З огляду на фінансові можливості здобувачів освіти, індивідуальна оренда квартири чи будинку зазвичай є занадто дорогою, тому найпопулярнішим рішенням стає спільна оренда житла групою з кількох осіб.

Попри наявність великої кількості цифрових сервісів для пошуку нерухомості, процес організації спільного проживання залишається неоптимізованим та стресовим. Головна проблема полягає у фрагментації комунікації. Зазвичай процес виглядає так: один студент із компанії бере на себе роль "пошуковця". Він переглядає варіанти на різних сайтах, зв'язується з орендодавцями, після чого робить знімки екрана (скріншоти) або копіює посилання та надсилає їх у спільну групу в сторонньому месенджері (Telegram, Viber).

Коли у друзів виникають додаткові запитання (наприклад, щодо наявності побутової техніки чи розміру комунальних платежів), "пошуковець" змушений знову писати орендодавцю, чекати на відповідь і переказувати її своїй компанії. Виникає ефект "зіпсованого телефону", що призводить до:

- втрати часу на пересилання інформації туди-й-назад;
- непорозумінь між майбутніми співмешканцями та орендодавцем;
- швидкої втрати вигідних варіантів через затримки в комунікації;
- психологічного виснаження ініціатора пошуку.

Вирішенням цієї проблеми є розробка спеціалізованої платформи «UniStay», орієнтованої виключно на студентів та викладачів. Головною інновацією системи є

вбудований модуль групових чатів. На відміну від стандартних платформ, користувач UniStay може в кілька кліків створити кімнату для спілкування, додати туди своїх друзів (співмешканців) та безпосередньо орендодавця. Таким чином, усі учасники процесу бачать повідомлення одночасно, можуть ставити запитання напряму та оперативно приймати спільні рішення без використання сторонніх додатків.

1.2. Аналіз існуючих платформ нерухомості та їх недоліки у контексті групової оренди

Створення нішевого програмного продукту вимагає детального аналізу існуючих лідерів ринку. Для ринку України найпопулярнішими платформами, де розміщуються оголошення про нерухомість, є OLX, LUN та Flatfy. Хоча ці сервіси є технічно досконалими та мають величезну базу даних, вони орієнтовані на масового, загального користувача.

1.2.1 Платформа OLX

OLX — це найбільший універсальний маркетплейс оголошень в Україні. Тут можна знайти як товари широкого вжитку, так і нерухомість (продаж та оренда).

Переваги: Величезна база оголошень, зручна система фільтрації за ціною та районом.

Недоліки для нашої цільової аудиторії: OLX розрахований на взаємодію «один на один» (B2C або C2C). Вбудований месенджер платформи дозволяє вести діалог лише між автором оголошення та одним потенційним клієнтом. Тут відсутня технічна можливість запросити в існуючий чат третю особу. Це змушує користувачів переходити в інші месенджери, про що йшлося в попередньому підрозділі.

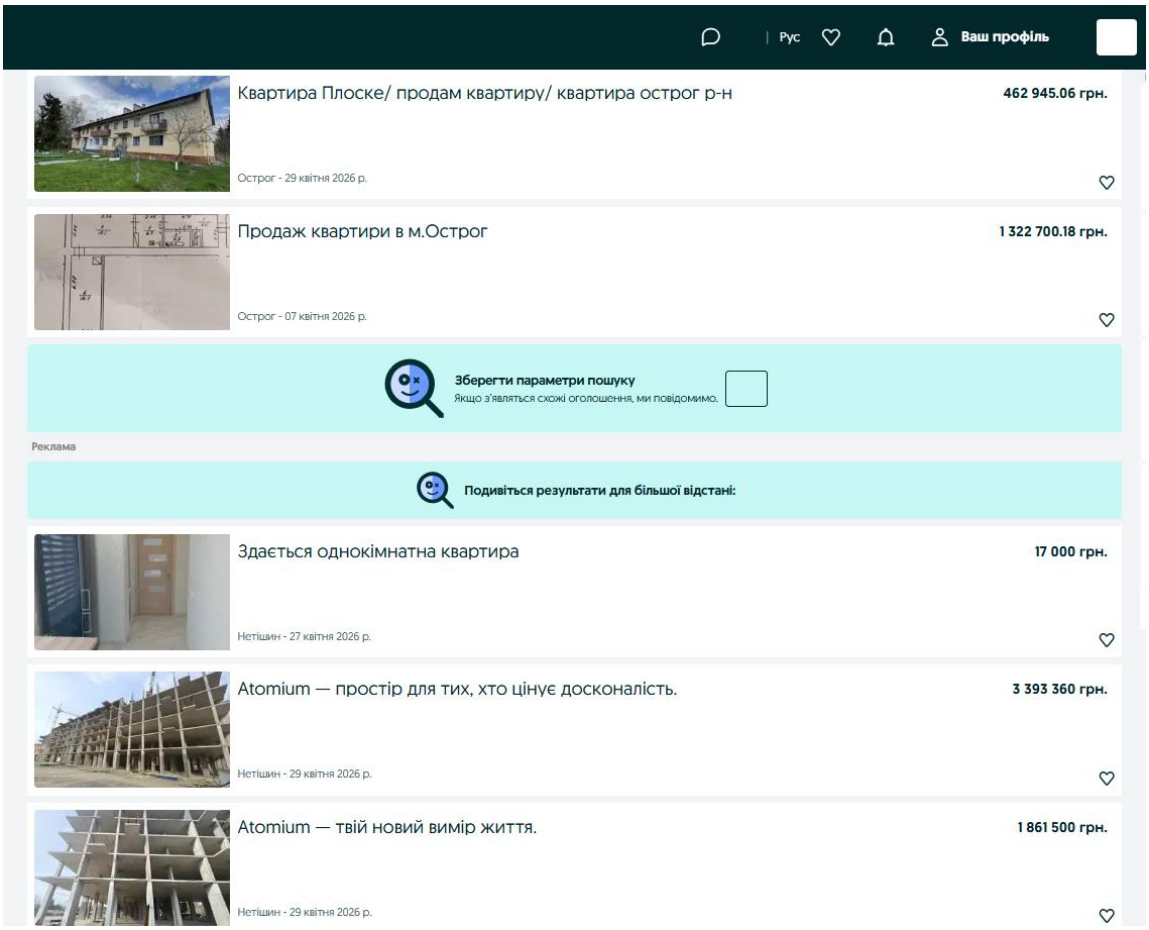


Рис 1.1.1 Вигляд сторінки OLX з пошуком житла в Острозі

Джерело: http:/olx.ua

Що цікаво, під час створення скріншоту, я зайшов на платформу OLX, обрав всі фільтра для пошуку житла в Острозі, і що ми бачимо, всього лиш два оголошення в нашому місті, і всі два це продаж житла, а не оренда, тобто для студента цей сайт приносить нуль користі, так як на ньому немає жодного варіанту, від якого можна відштовхнутись, це вже ми не говоримо про неможливість групового чату.

1.2.2 Платформи LUN та Flatfy

LUN спеціалізується переважно на новобудовах та преміум-сегменті, тоді як Flatfy працює як агрегатор оголошень з різних сайтів.

Переваги: Сучасний інтерфейс, відмінна інтеграція з картою, перевірені фотографії.

Недоліки: Ці платформи здебільшого працюють як вітрини. Вони взагалі не мають функціоналу внутрішніх чатів. Замість цього користувачу пропонується кнопка «Показати телефон», після натискання якої подальша комунікація відбувається у телефонному режимі або через Viber/Telegram. Для групового пошуку житла студентами це абсолютно не вирішує проблему синхронізації інформації.

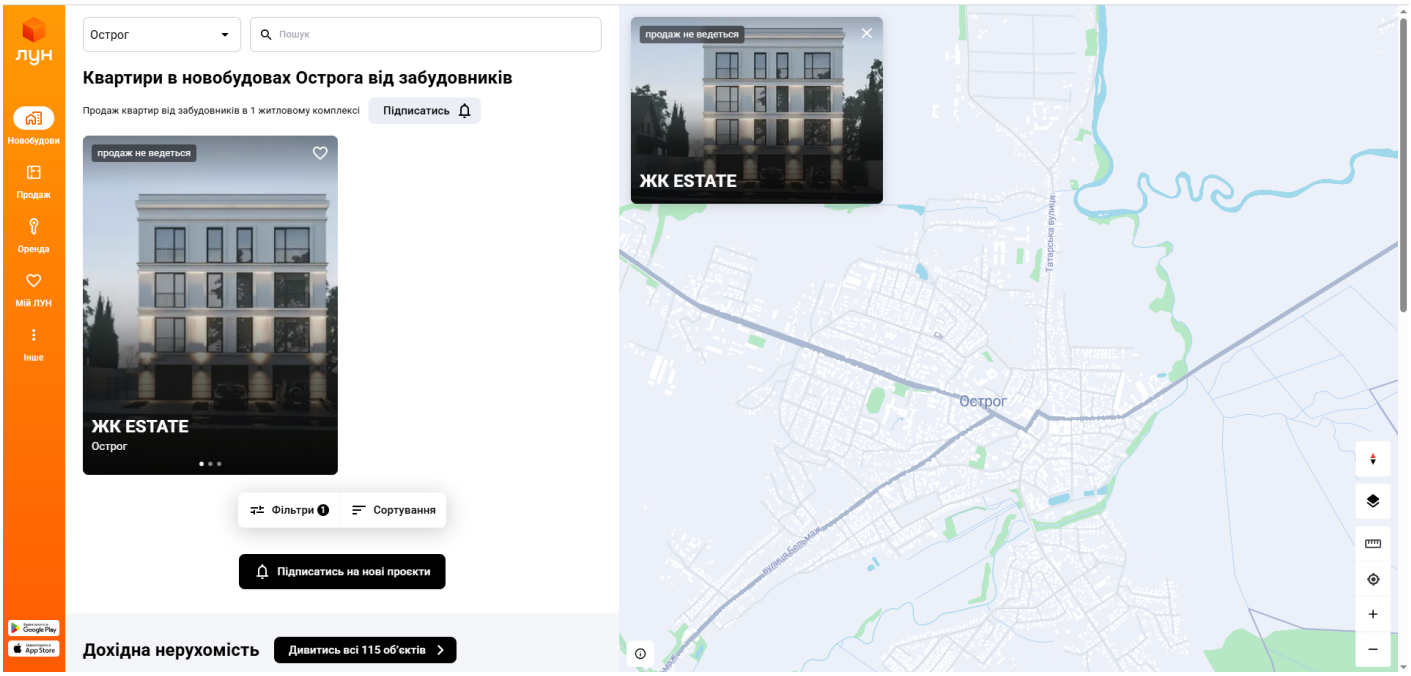


Рис 1.2.2 Вигляд сторінки Lun з пошуком житла в Острозі

Джерело: https://lun.ua

Аналогічну ситуацію можемо спостерігати на платформі Lun, лише одне оголошення, яке не підходить під параметри студента. Тобто на двох великих платформах з пошуком житла, студент бачить нуль пропозицій які б йому підходили, і все зводиться до пошуку різних варіантів в соціальних мережах, чатах, переказами від інших людей та знайомих, що сильно затягує та ускладнює весь процес, і при такому пошуку, не можна бути впевненим в достовірність всієї інформації.

Порівняльна оцінка рішень

На основі аналізу можна сформувати порівняльну таблицю (Таблиця 1.1), яка демонструє, чому створення UniStay є актуальним.

Таблиця 1.1 — Порівняння платформ для пошуку житла

Платформа	**Цільова аудиторія**	**Наявність внутрішнього чату**	**Можливість групових чатів**	**Орієнтація на студентів**
OLX	Масова	Є (лише 1-на-1)	Немає	Немає
LUN/Flatfy	Середній/Преміум	Немає (лише телефон)	Немає	Немає
UniStay (Проєкт)	Студенти/Викладачі	Є	**Є (повноцінні групи)**	**Так**

Отже, існуючі універсальні платформи не задовольняють специфічні потреби студентів щодо колективного підбору житла. Проєкт UniStay заповнює цю нішу, надаючи інструментарій, якого немає у конкурентів.

1.3. Теоретичні аспекти побудови високонавантаженого бекенду та архітектури реального часу

Реалізація заявленого функціоналу, зокрема системи групових чатів, вимагає глибокого розуміння архітектурних патернів. Звичайні вебзастосунки часто будуються за монолітною або спрощеною тришаровою архітектурою, яка добре працює для стандартних CRUD-операцій (створення, читання, оновлення, видалення оголошень). Однак наявність чату в реальному часі значно підвищує вимоги до бекенду.

1.3.1 Clean Architecture (Чиста Архітектура)

Для того, щоб система була стійкою до змін, легко тестувалася та масштабувалася, бекенд доцільно будувати за принципами Clean Architecture, запропонованими Робертом Мартіном. Суть цього підходу полягає в розділенні програми на незалежні шари (кільця).

1. **Domain (Доменний шар):** Серце системи. Тут знаходяться базові сутності (Користувач, Оголошення, Чат, Повідомлення) та сувора бізнес-логіка, яка не залежить від жодних зовнішніх фреймворків чи баз даних.
2. **Application (Шар застосунку):** Містить сценарії використання (Use Cases), інтерфейси та DTO-моделі (Data Transfer Objects). Він визначає, *що* система повинна робити.
3. **Infrastructure (Інфраструктурний шар):** Реалізує роботу з базою даних, файловими сховищами, зовнішніми API.
4. **Presentation / API (Шар представлення):** Точка входу в систему (контролери), яка приймає HTTP-запити від фронтенду та повертає відповіді.

Такий підхід гарантує, що логіка групового чату не переплітається з технічними деталями збереження даних, що дозволяє у майбутньому легко замінити, наприклад, базу даних, не переписуючи ядро системи.

1.3.2 Технології реального часу (WebSockets)

Класична клієнт-серверна взаємодія базується на протоколі HTTP: клієнт робить запит, сервер дає відповідь і закриває з'єднання. Для чату це означало б, що пристрій користувача мав би кожну секунду запитувати сервер: "Чи є нові повідомлення?". Це створює колосальне і невиправдане навантаження на систему (Long Polling).

Для вирішення цієї проблеми використовується технологія **WebSockets**. Вона встановлює постійне двостороннє з'єднання між клієнтом і сервером. Завдяки цьому сервер може самостійно, миттєво "штовхати" (push) нові повідомлення всім учасникам групового чату без необхідності постійних запитів з їхнього боку.

1.4. Обґрунтування вибору технологічного стеку для платформи "UniStay"

Спираючись на архітектурні вимоги, для реалізації серверної частини (бекенду) було обрано наступний технологічний стек:

1. **Мова програмування C# та платформа .NET 8.** Це один із найпотужніших та найшвидших фреймворків для розробки серверних застосунків. Строга типізація мови C# мінімізує кількість помилок на етапі написання коду, а висока продуктивність .NET 8 дозволяє обробляти тисячі одночасних підключень до чату без затримок.
2. **СУБД PostgreSQL.** Надійна, об'єктно-реляційна система управління базами даних з відкритим вихідним кодом. Вона ідеально підходить для зберігання складних зв'язків (наприклад, коли один користувач перебуває у багатьох чатах, а один чат містить багато повідомлень та багато користувачів).
3. **Entity Framework Core (EF Core).** Офіційний ORM-фреймворк (Object-Relational Mapper) від Microsoft. Він дозволяє розробнику працювати з базою даних через об'єкти C#, генерувати безпечні SQL-запити та керувати транзакціями бази даних, що пришвидшує процес розробки та підвищує рівень безпеки.
4. **Технологія SignalR.** Це спеціальна бібліотека для ASP.NET Core, яка інкапсулює роботу з WebSockets (та іншими протоколами реального часу). Вона дозволяє легко створювати "Хаби" (Hubs) — серверні класи, через які відбувається миттєва трансляція повідомлень між учасниками однієї групи у чаті UniStay.

Висновки до розділу 1

У першому розділі кваліфікаційної роботи було проведено комплексне теоретичне дослідження предметної області, проаналізовано існуючі рішення на ринку нерухомості та сформовано архітектурні й технологічні вимоги до розробки серверної частини платформи «UniStay». За результатами дослідження можна зробити наступні висновки:

1. Досліджено проблематику пошуку житла у студентському середовищі. Встановлено, що ключовим бар'єром для комфортної спільної оренди є розрізненість комунікації. Традиційний підхід вимагає використання сторонніх

месенджерів для координації між майбутніми співмешканцями та орендодавцем, що призводить до втрати часу, інформації та ускладнює процес прийняття рішень. Доведено необхідність створення спеціалізованої платформи з інтегрованим простором для групового спілкування.

2. Проведено порівняльний аналіз провідних платформ нерухомості (OLX, LUN, Flatfy). Виявлено, що попри їхню технічну досконаліть, вони орієнтовані переважно на індивідуального споживача. Вбудовані механізми зв'язку на цих платформах або обмежуються діалогом «один на один» (як у випадку з OLX), або взагалі відсутні, пропонуючи лише контактний номер телефону. Жоден із лідерів ринку не забезпечує можливості створення багатокористувацьких чатів для спілкування з орендодавцем, що підтверджує високу актуальність, конкурентоспроможність та унікальність проєкту «UniStay».
3. Обґрунтовано вибір архітектурного підходу. Зважаючи на вимоги до функціоналу чатів у реальному часі та перспективи масштабування проєкту, встановлено недоцільність використання стандартної монолітної архітектури. Обґрунтовано перехід до принципів Clean Architecture (Чистої Архітектури), що дозволить ізолювати бізнес-логіку системи від інфраструктурних залежностей, забезпечить високу гнучкість коду та надійність платформи при високих навантаженнях.
4. Визначено та обґрунтовано технологічний стек розробки бекенду. Для реалізації поставлених завдань обрано платформу .NET 8 завдяки її високій продуктивності та безпеці. Зберігання реляційних даних (користувачі, оголошення, зв'язки між учасниками чатів) буде здійснюватися за допомогою СУБД PostgreSQL та ORM-фреймворку Entity Framework Core. Для вирішення проблеми надмірного навантаження на сервер під час обміну повідомленнями обрано технологію WebSockets та бібліотеку SignalR, які забезпечать стабільне двостороннє з'єднання та миттєву трансляцію даних.

Отже, результати теоретичного дослідження, проведеного в першому розділі, сформували чітку візію продукту та надійну методологічну базу. Визначені проблеми користувачів, архітектурні патерни та технологічні інструменти є основою для переходу до етапу практичного проєктування бази даних, розробки REST API та імплементації модулів системи, що буде детально розглянуто в наступному розділі.

РОЗДІЛ 2. ПРИКЛАДНА ЧАСТИНА РОЗРОБКИ ПЛАТФОРМИ «UNISTAY»

2.1 Серверна частина вебзастосунку

2.1.1 Вибір технологій та загальна структура

Серверна частина платформи «UniStay» реалізована за допомогою платформи .NET 8 та мови програмування C#. Цей вибір зумовлений необхідністю створення високопродуктивного та строго типізованого REST API, здатного підтримувати одночасні підключення в режимі реального часу, масштабуватися під навантаженням та забезпечувати високий рівень безпеки даних. Основні переваги обраного стеку для даного проєкту:

- Висока продуктивність: .NET 8 забезпечує оптимальне використання ресурсів сервера та пам'яті, що критично важливо для додатків із функціоналом чатів та обробки медіафайлів.
- Асинхронність: Повна підтримка парадигми async/await на всіх рівнях (від контролерів до доступу до БД) дозволяє обробляти тисячі запитів без блокування основних потоків виконання.
- Інтегрованість: Вбудована підтримка Dependency Injection (впровадження залежностей) робить код модульним, готовим до юніт-тестування та легким для підтримки.

Окрім ядра ASP.NET Core, у проєкті використовуються наступні ключові технології:

- PostgreSQL — надійна об'єктно-реляційна система керування базами даних (СУБД), що відмінно справляється зі складними зв'язками між сутностями.
- Entity Framework Core (EF Core) — сучасний ORM-фреймворк для опису моделей бази даних засобами C#, управління зв'язками та застосування міграцій без написання сирих SQL-запитів.
- SignalR — бібліотека для забезпечення двосторонньої комунікації між сервером і клієнтом у реальному часі на базі протоколу WebSockets.

- MediatR — інструмент для реалізації патерну CQRS (Command Query Responsibility Segregation) та ізоляції бізнес-логіки.

Проєкт побудований за принципами Чистої архітектури (Clean Architecture). Логіка суворо розділена на ізольовані шари, де залежності спрямовані виключно всередину — до доменного ядра.

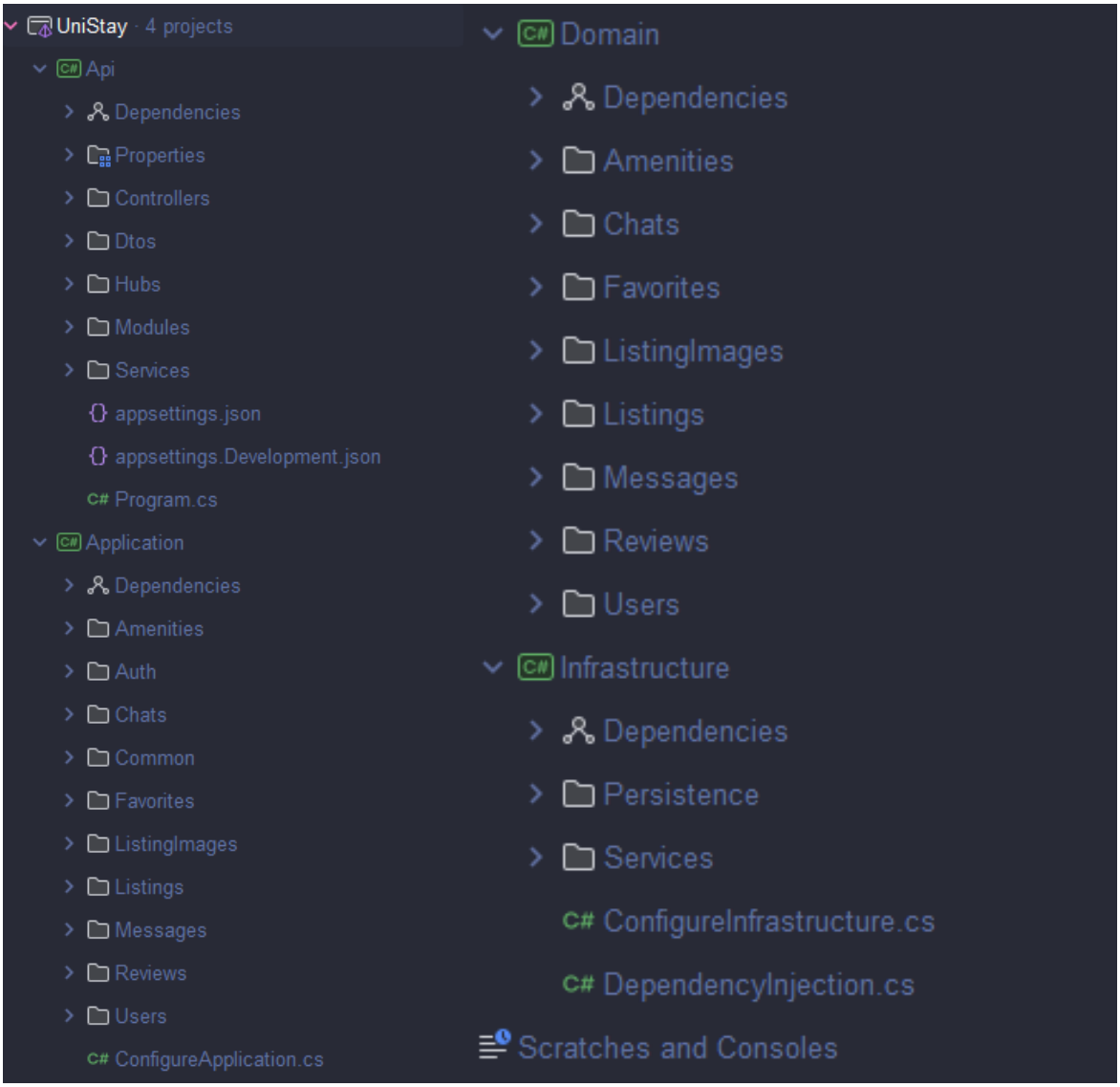


Рис 2.1 — Архітектура Backend-частини платформи.

У процесі реалізації було виділено наступні модулі:

- Domain: Ядро системи. Містить виключно бізнес-сутності та правила. Цей шар не залежить від жодних баз даних чи веб-фреймворків.
- Application: Містить бізнес-логіку програми (Use Cases). Тут знаходяться обробники команд та запитів, DTO (Data Transfer Objects), пайплайни валідації та інтерфейси для взаємодії із зовнішнім світом.
- Infrastructure: Реалізує роботу з базою даних через патерн Repository, а також містить імплементацію сервісів генерації токенів та хешування паролів.
- Api (Presentation Layer): Точка входу в застосунок. Містить Controllers для обробки REST-запитів, Middleware для обробки помилок та Hubs для підключень SignalR.

Завдяки такому підходу вдалося досягти високої масштабованості коду. Бізнес-логіка повністю захищена від змін в інфраструктурі, що дозволяє, наприклад, змінити СУБД без переписування ядра програми.

2.1.2 Моделі бази даних

Для зберігання інформації використовується PostgreSQL. Усі сутності системи описані в шарі Domain і відображаються на таблиці за допомогою EF Core.

Інноваційним рішенням у моделюванні стало використання патерну Strongly Typed IDs (сильно типізованих ідентифікаторів). Замість використання стандартних типів Guid або int для всіх таблиць, були створені типи-обгортки (наприклад, ChatId, UserId, ListingId). Це вирішує проблему «одержимості примітивами» (Primitive Obsession) та унеможливлює помилки на етапі компіляції, коли ідентифікатор однієї сутності помилково передається в метод для іншої.

Основні доменні моделі системи включають:

- User: Зберігає дані профілю, криптографічний хеш пароля та роль користувача.
- Listing: Оголошення про нерухомість. Інкапсулює інформацію про тип житла, координати, ціну, умови проживання та пов'язані зображення.
- Amenity: Довідник зручностей (Wi-Fi, пральна машина тощо), реалізований через зв'язок «багато-до-багатьох» з оголошеннями.

- Chat / ChatMessage: Сутності групового або приватного чату, що акумулюють у собі список учасників та історію повідомлень.
- OstrohLandmark: Допоміжна сутність для збереження координат ключових локацій міста, необхідна для алгоритмів просторового пошуку.

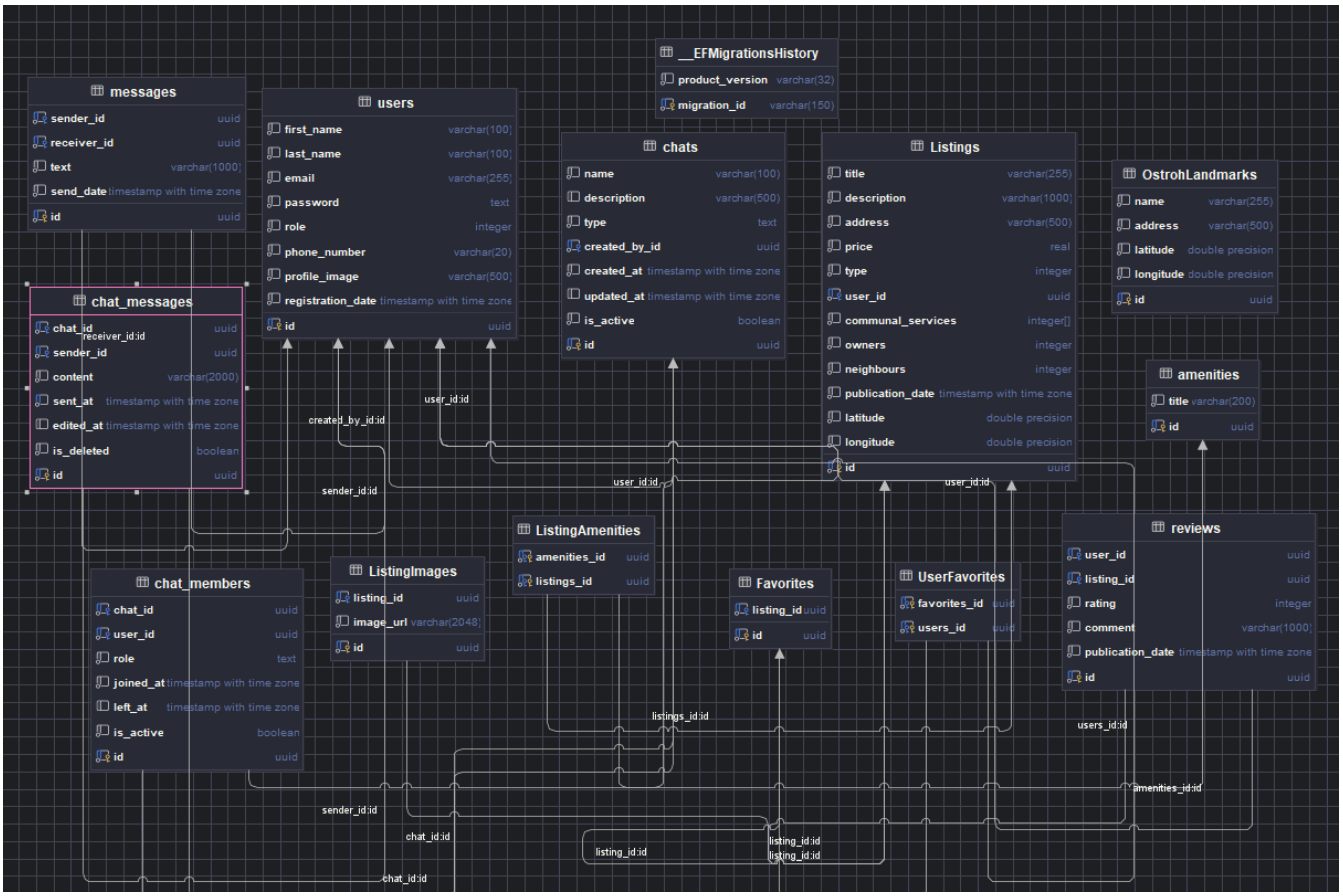


Рис 2.2 — Схема реляційної бази даних проєкту.

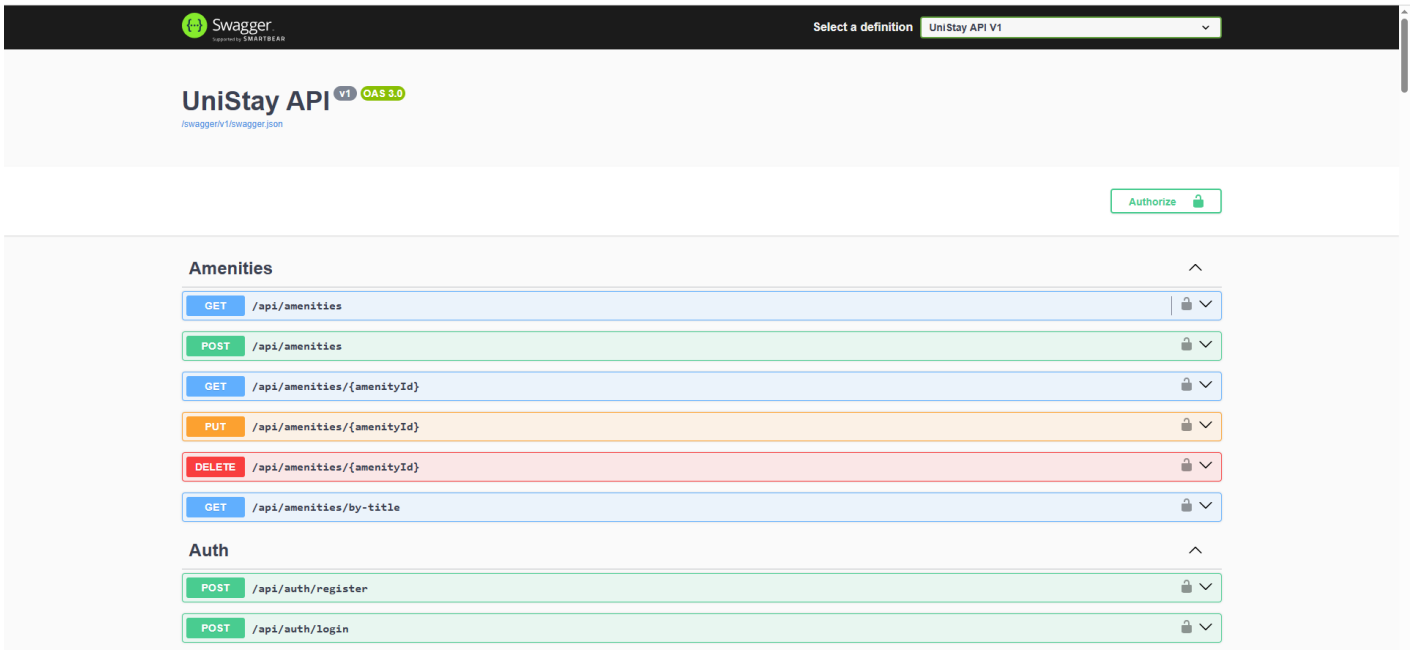
2.1.3 API та маршрутизація

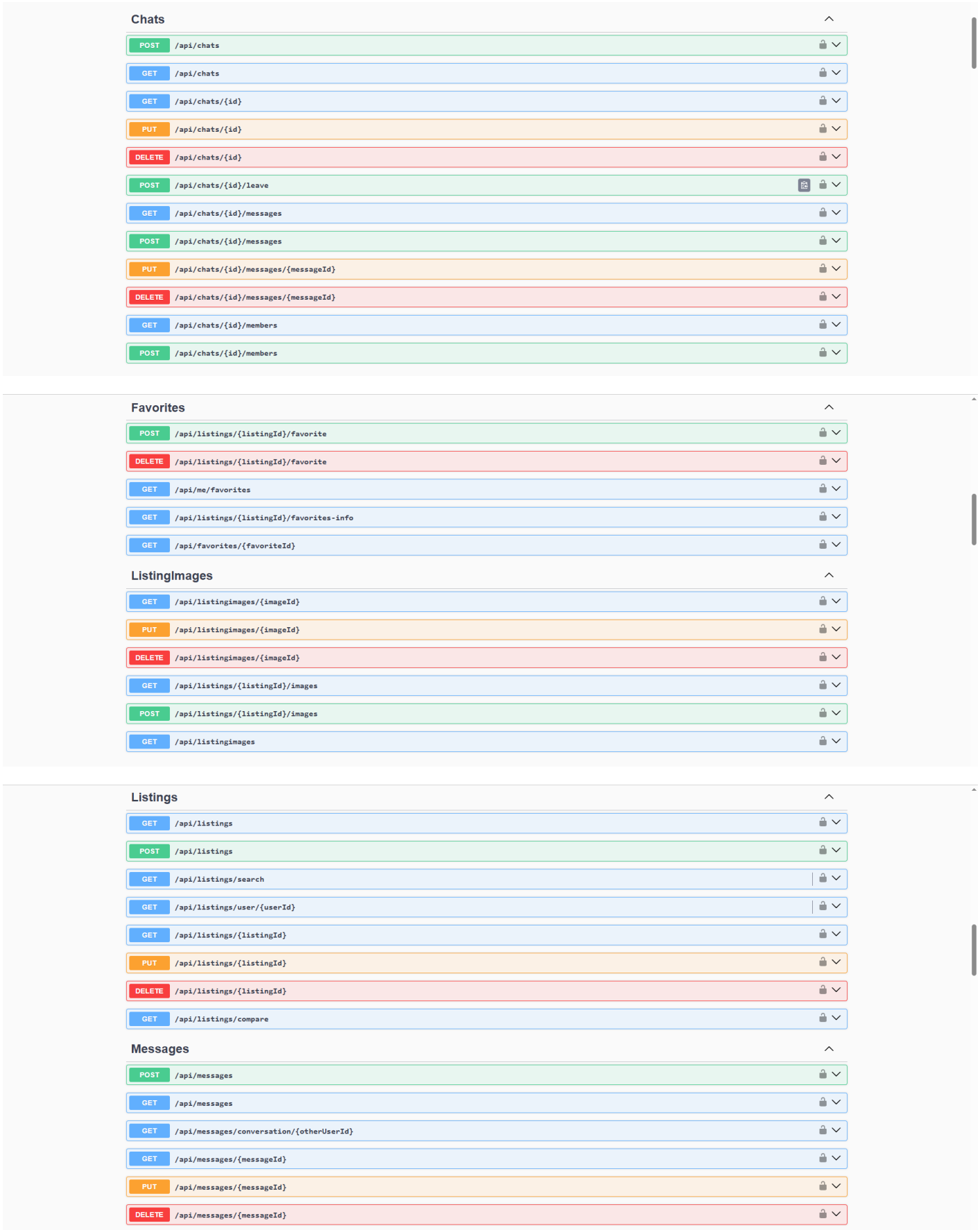
Маршрутизація системи реалізована через контролери, які згруповані за функціональним призначенням. REST API побудовано з суворим дотриманням стандартів HTTP (використання правильних дієслів GET, POST, PUT, DELETE та відповідних статус-кодів).

Основні групи маршрутів охоплюють усі аспекти взаємодії з платформою:

- /api/auth/ — управління доступом: автентифікація, генерація токенів, реєстрація нових користувачів.
- /api/listings/ — отримання каталогу житла з підтримкою складної фільтрації, створення, редагування та видалення оголошень орендодавцями.
- /api/chats/ — керування чатами та отримання історії переписок.
- /api/users/ — управління профілем.

Усі маршрути автоматично документуються за допомогою стандарту OpenAPI (Swagger). Це створює зручний графічний інтерфейс для тестування ендпоінтів розробниками та дозволяє клієнтській частині автоматично генерувати типи даних для взаємодії з бекендом, що мінімізує кількість помилок при інтеграції.




Chats
POST /api/chats
GET /api/chats
GET /api/chats/{id}
PUT /api/chats/{id}
DELETE /api/chats/{id}
POST /api/chats/{id}/leave
GET /api/chats/{id}/messages
POST /api/chats/{id}/messages
PUT /api/chats/{id}/messages/{messageId}
DELETE /api/chats/{id}/messages/{messageId}
GET /api/chats/{id}/members
POST /api/chats/{id}/members
Favorites
POST /api/listings/{listingId}/favorite
DELETE /api/listings/{listingId}/favorite
GET /api/me/favorites
GET /api/listings/{listingId}/favorites-info
GET /api/favorites/{favoriteId}
ListingImages
GET /api/listingimages/{imageId}
PUT /api/listingimages/{imageId}
DELETE /api/listingimages/{imageId}
GET /api/listings/{listingId}/images
POST /api/listings/{listingId}/images
GET /api/listingimages
Listings
GET /api/listings
POST /api/listings
GET /api/listings/search
GET /api/listings/user/{userId}
GET /api/listings/{listingId}
PUT /api/listings/{listingId}
DELETE /api/listings/{listingId}
GET /api/listings/compare
Messages
POST /api/messages
GET /api/messages
GET /api/messages/conversation/{otherUserId}
GET /api/messages/{messageId}
PUT /api/messages/{messageId}
DELETE /api/messages/{messageId}

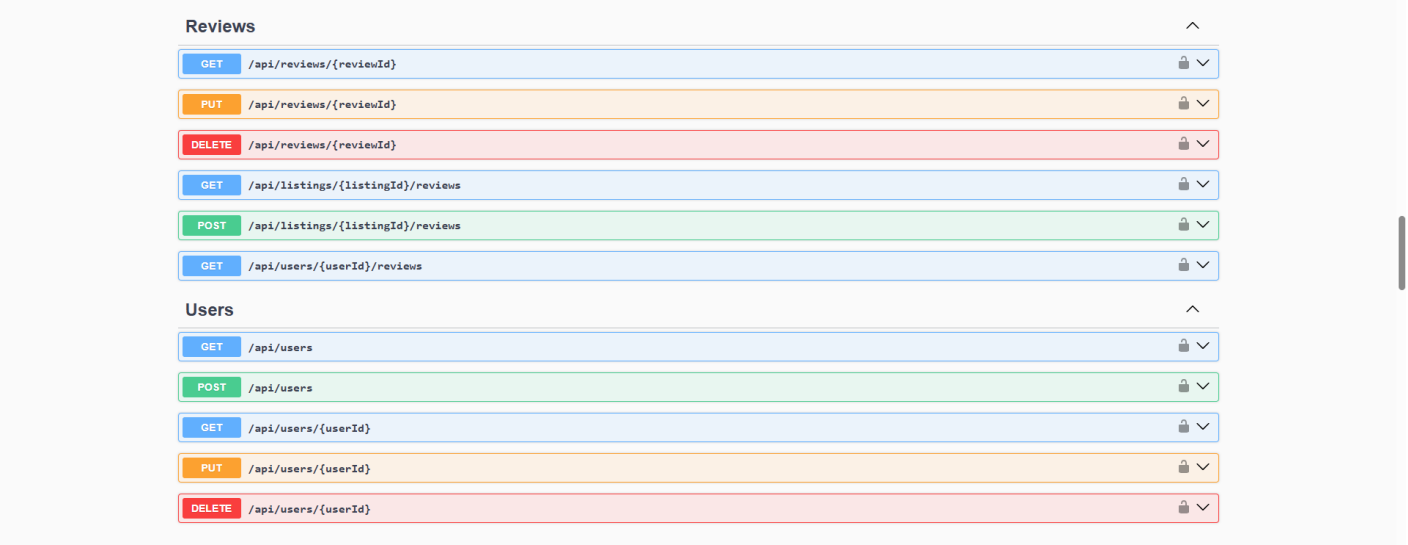


Рис 2.3 — Документація REST API за допомогою Swagger UI.

2.1.4 Модуль групового підбору та чат-сервіс

Функціонал чатів є критично важливим компонентом платформи «UniStay», що вирішує проблему розрізненої комунікації студентів при пошуку спільного житла. Бізнес-логіка чатів повністю інкапсульована в шарі Domain з використанням принципів Domain-Driven Design (DDD). Сутність Chat не є «анемічною» моделлю (простим набором властивостей з публічними сетерами), а самостійно контролює процеси зміни свого стану. Наприклад, логіка перевіряє активність учасника перед додаванням нового повідомлення, гарантуючи консистентність даних. Детальний лістинг доменної моделі чату наведено в **Додатку А (Лістинг А.1)**

Для миттєвого обміну повідомленнями без необхідності ручного оновлення сторінки клієнтом інтегровано технологію SignalR. Оскільки REST API працює за принципом «запит-відповідь», воно не підходить для чатів. SignalR створює постійне з'єднання через протокол WebSockets.

Коли користувач відправляє повідомлення, сервер спочатку зберігає його в PostgreSQL через Application-шар, гарантуючи збереження історії. Після успішного транзакційного запису, сервер через спеціальний клас ChatHub миттєво транслює це

повідомлення (push-повідомлення) всім активним клієнтам, підключеним до відповідної кімнати чату. Фрагмент коду реалізації хабу представлено в **Додатку А (Лістинг А.2)**. Архітектурну схему роботи WebSocket-з'єднання зображено на рисунку 2.4.

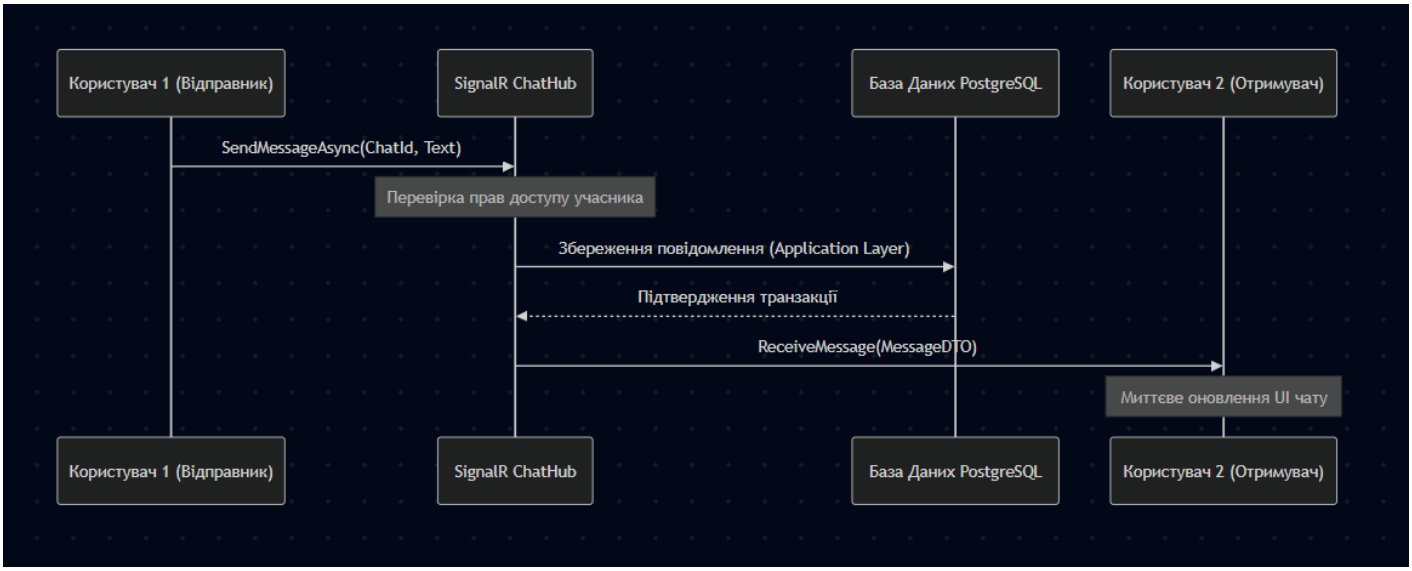


Рис 2.4 — Алгоритм обміну повідомленнями в реальному часі через SignalR

2.1.5 Безпека та автентифікація

Безпека платформи та авторизація запитів базується на використанні стандарту JSON Web Token (JWT). Система автентифікації побудована за архітектурним патерном CQRS, що робить код контролерів максимально тонким, переносячи всю складну логіку обробки в спеціалізовані класи — Command Handlers (див. **Додаток А, Лістинг А.3**).

Процес входу користувача працює за наступним алгоритмом:

1. Виконується пошук користувача в базі даних за електронною поштою.
2. Сервіс IPasswordHash верифікує криптографічний хеш пароля. Система не зберігає паролі у відкритому вигляді, що захищає дані користувачів у випадку компрометації бази даних.

3. У разі успіху сервіс IJwtGenerator створює підписаний токен доступу, що містить ідентифікатор користувача (UserId) та його системну роль (Admin/User).

Схему взаємодії компонентів при використанні патерну CQRS з медіатором наведено на рисунку 2.5.

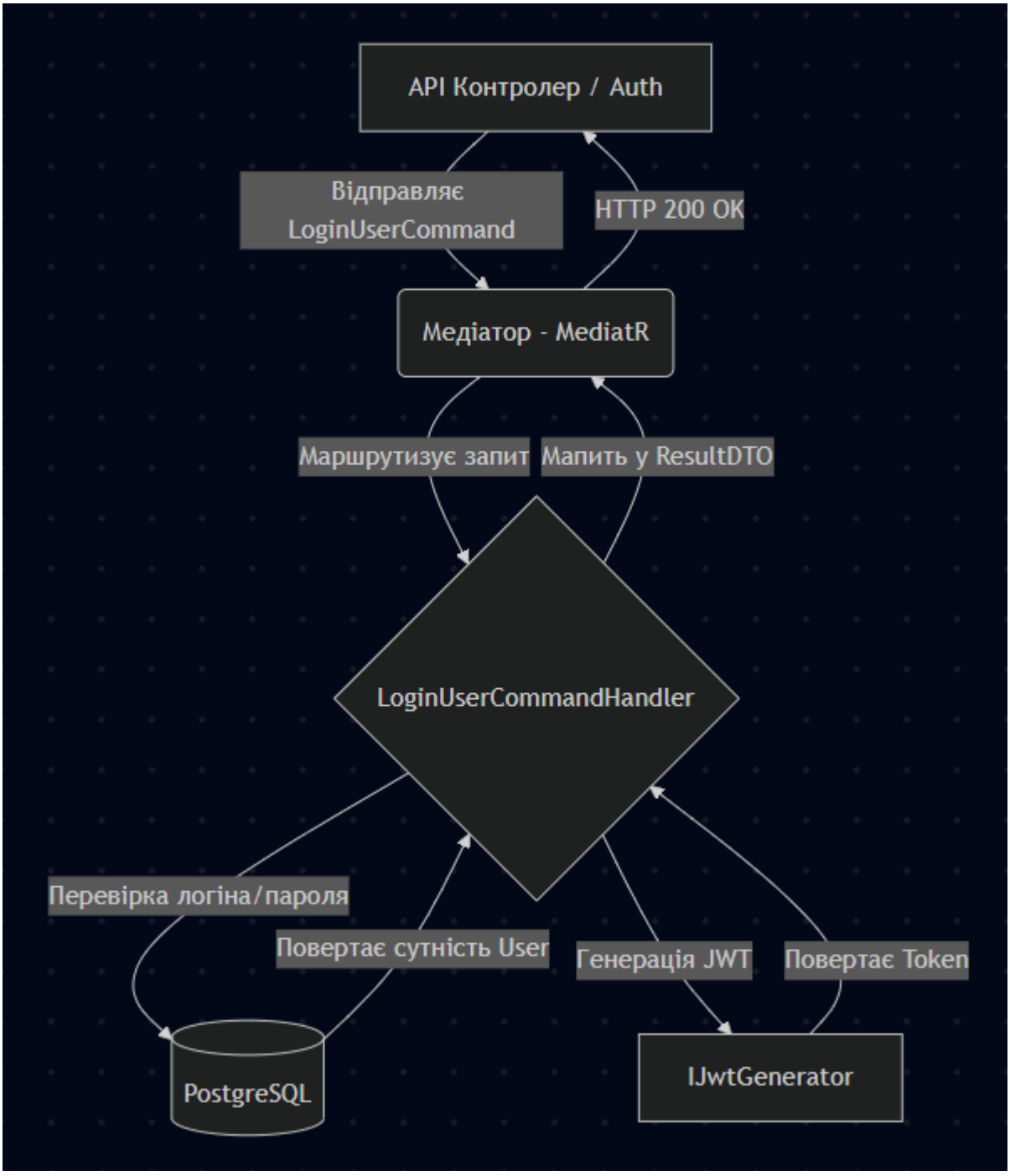


Рис 2.5 — Архітектура обробки запитів автентифікації на базі CQRS

Кожен запит до захищених маршрутів проходить через вбудований Middleware-фільтр перевірки JWT-токена. Важливою архітектурною особливістю є використання патерну Result (повернення об'єкта Result<T, Exception>). Замість того, щоб генерувати дорогі системні винятки (Exceptions) при звичайних помилках авторизації (наприклад, невірний пароль), бекенд повертає об'єкт-результат. Це суттєво оптимізує споживання ресурсів процесора.

2.1.6 Валідація даних і структура схем

Враховуючи специфіку платформи, де користувачі вводять великий обсяг персональних даних та інформації про житло, першочерговим завданням є надійна система перевірки вхідної інформації. У проєкті відмовилися від базової валідації через атрибути даних (Data Annotations), оскільки вона порушує принципи єдиної відповідальності та обмежує можливості для складних перевірок.

Натомість була інтегрована бібліотека FluentValidation. Цей підхід дозволяє виносити всі правила перевірки в окремі класи-валідатори, забезпечуючи чистоту DTO-моделей (приклад реалізації наведено у **Додатку А, Лістинг А.4**).

Щоб зробити процес валідації безшовним, у системі реалізовано патерн Pipeline Behavior за допомогою бібліотеки MediatR. Кожен HTTP-запит автоматично перехоплюється проміжним програмним забезпеченням до потрапляння в бізнес-логіку. Схему роботи глобального фільтру наведено на рисунку 2.6

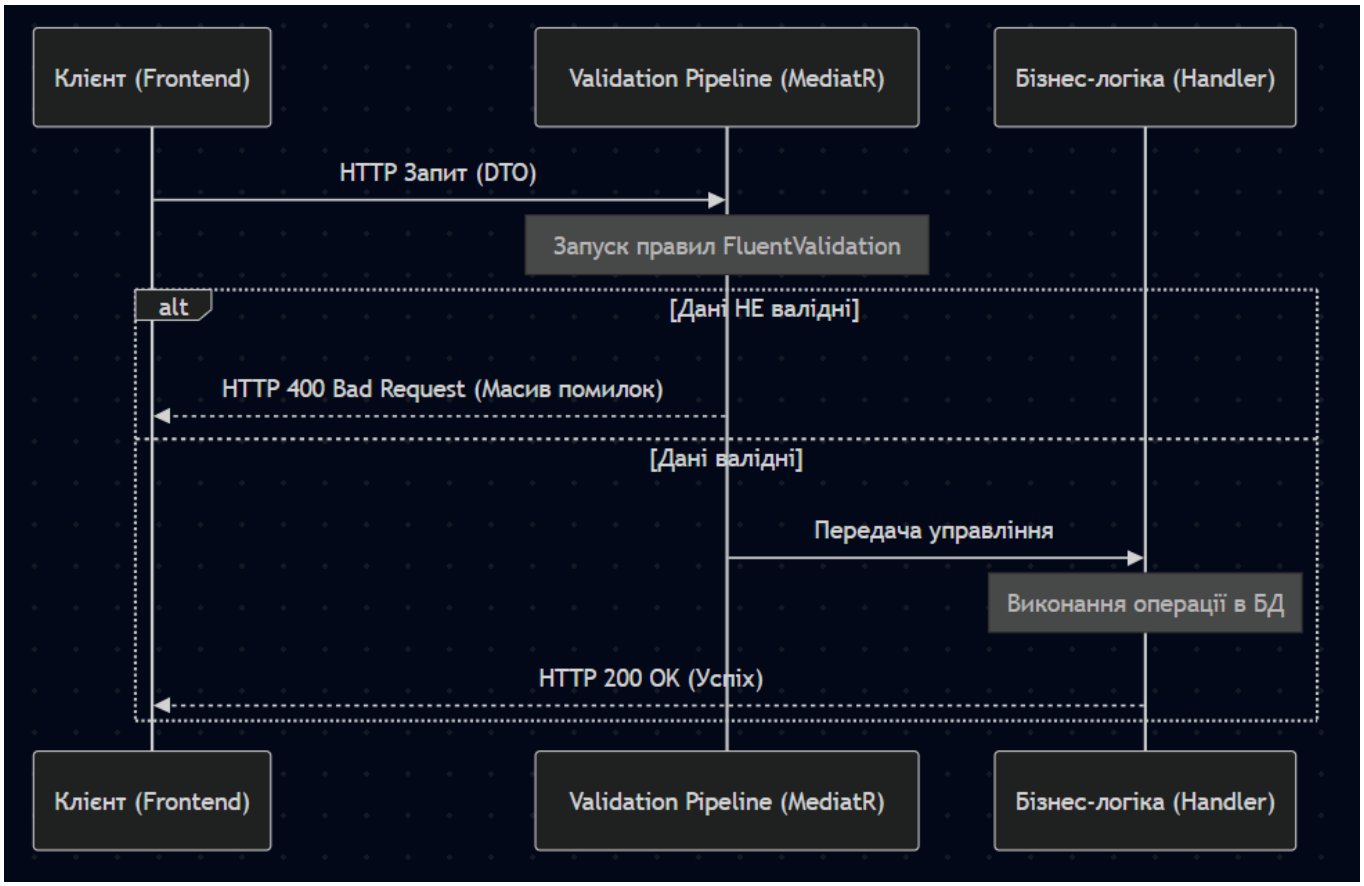


Рис 2.6 — Процес перехоплення та валідації запитів через Pipeline Behavior

2.1.7 Контроль доступу на рівні ресурсів (Resource-based Authorization)

Стандартної автентифікації за допомогою JWT-токенів достатньо лише для перевірки факту входу користувача в систему. Проте для захисту персональних даних платформи необхідно реалізувати контроль доступу на рівні конкретних ресурсів (Resource-based Authorization).

Це означає, що система перевіряє не лише валідність токена, але й те, чи має поточний авторизований користувач права на взаємодію з конкретним об'єктом. У класичній архітектурі цю проблему часто вирішують нагромадженням умов if безпосередньо в бізнес-логіці, що порушує чистоту коду. У платформі «UniStay» ця логіка інкапсульована на рівні приватних методів та фільтрів контролера, не пропускаючи неавторизовані запити глибше до шару Application.

Наприклад, у модулі управління користувачами реалізовано чіткий механізм перевірки прав: системні адміністратори (Administrator) мають повний доступ до всіх профілів на платформі, тоді як звичайні користувачі можуть переглядати, редагувати або видаляти виключно власний обліковий запис. Приклад імплементації цієї логіки наведено в **Додатку А (Лістинг А.5)**.

Коли надходить запит на отримання або зміну даних користувача, система витягує ідентифікатор із JWT-токена (Claim NameIdentifier). Далі метод CanAccessUser перевіряє, чи збігається цей ідентифікатор із тим, який запитується, або чи має користувач роль адміністратора. Якщо перевірка не проходить, сервер миттєво відхиляє запит і повертає клієнту статус-код 403 Forbidden (Доступ заборонено), захищаючи дані від несанкціонованого доступу.

Щоб зробити процес валідації безшовним, у системі реалізовано патерн Pipeline Behavior за допомогою бібліотеки MediatR. Кожен HTTP-запит, який надходить до бекенду, автоматично перехоплюється проміжним програмним забезпеченням (ValidationBehaviour) до потрапляння в бізнес-логіку. Якщо дані не відповідають заданим правилам, пайплайн перериває виконання і миттєво повертає клієнту колекцію помилок. Це гарантує, що до доменного ядра завжди надходять виключно валідні дані.

2.1.8 Геолокація та алгоритми просторового пошуку

Унікальною особливістю платформи «UniStay» є інтелектуальний пошук житла з прив'язкою до конкретних географічних орієнтирів. Оскільки цільовою аудиторією є студенти, одним із ключових критеріїв вибору квартири є її фактична віддаленість від навчальних корпусів.

Для розв'язання цієї задачі створено модуль просторового пошуку на основі сутності **OstrohLandmark**, яка містить точні координати важливих інфраструктурних об'єктів міста. Розрахунок відстані реалізовано безпосередньо в доменній моделі за допомогою формули Гаверсинуса сферичної тригонометрії (див. Додаток А, Лістинг А.6). Вона

враховує кривизну поверхні Землі та дозволяє з високою точністю обчислити найкоротшу відстань між двома точками.

Оскільки базові інструменти Entity Framework Core не можуть напряму транслювати складні математичні C#-методи у SQL-запити до бази даних, процес сортування оптимізовано на рівні Application-шару. Система спочатку отримує попередньо відфільтрований за базовими критеріями (ціна, тип) список житла з бази даних у пам'ять сервера. Після цього до списку застосовується алгоритм Гаверсинуса (через LINQ), який швидко обчислює відстані та ранжує результати для користувача.

2.1.9 Взаємодія з базою даних (Патерн Repository)

Взаємодія бекенду з базою даних побудована на суворому дотриманні патерну Repository, що абстрагує бізнес-логіку від деталей роботи EF Core. Особливістю архітектурного рішення є розділення інтерфейсів доступу до даних відповідно до принципів CQRS (див. Додаток А, Лістинг А.7). Наприклад, клас репозиторію одночасно реалізує інтерфейс для запису даних та окремий інтерфейс виключно для їх читання.

Такий підхід забезпечує вагомі переваги:

1. **Оптимізація продуктивності:** Для всіх запитів на читання використовується розширення .AsNoTracking(). Воно вказує ORM не створювати знімки сутностей у пам'яті для відстеження змін, що критично зменшує навантаження на Garbage Collector та прискорює виконання запитів.
2. **Захист від помилок:** Замість повернення значень null, репозиторії інкапсулюють результати в об'єкт **Option** (бібліотека Optional). Це змушує розробника на рівні сервісу явно обробляти ситуації відсутності даних, повністю виключаючи виникнення критичних помилок NullReferenceException.

Налаштування самої схеми бази даних (зв'язки таблиць, правила каскадного видалення) реалізовані за допомогою інструменту Fluent API. Це дозволяє не «забруднювати» доменні класи інфраструктурними атрибутами.

2.1.10 Автоматична ініціалізація бази даних та Data Seeding

Для забезпечення готовності системи до роботи одразу після розгортання на сервері (Deployment) розроблено сервіс ініціалізації ApplicationDbContextInitialiser.

Цей сервіс виконує дві функції при кожному запуску програми (реалізацію наведено в **Додатку А, Лістинг А.8**):

1. **Автоматичне застосування міграцій схеми:** Метод сервісу перевіряє стан БД та застосовує всі нові міграції. Це гарантує, що структура таблиць завжди синхронізована з поточною версією коду без ручного втручання.
2. **Наповнення даними (Data Seeding):** Для коректної роботи платформи на етапах тестування розроблено алгоритм наповнення порожньої бази. Процес генерує облікові записи адміністраторів, створює словники зручностей, а головне — ініціалізує реальні географічні координати корпусів Національного університету «Острозька академія», місцевих супермаркетів та гуртожитків.

Також створюється пул тестових оголошень різного типу (квартири, кімнати) з прив'язаними фотографіями та відгуками. Завдяки цьому комісія або нові розробники отримують миттєво готову до використання систему з реалістичним контентом, обминаючи етап тривалого ручного заповнення платформи даними.

Висновки до розділу 2

У другому розділі кваліфікаційної роботи було детально розглянуто процес практичної реалізації серверної частини (бекенду) вебплатформи «UniStay» для пошуку житла студентами Національного університету «Острозька академія». Вибір технологічного стека базувався на вимогах до надійності, масштабованості та безпеки, тому основним інструментом розробки було обрано платформу .NET та мову програмування C#.

Під час розробки було вирішено наступні ключові завдання:

1. **Проєктування архітектури системи.** Застосовано підхід «Чиста архітектура» (Clean Architecture), що дозволило чітко розділити систему на шари: Domain (предметна область), Application (бізнес-логіка), Infrastructure (робота з базою даних та зовнішніми сервісами) та API (точки доступу). Це забезпечило високу тестованість коду та його незалежність від конкретних фреймворків чи баз даних.
2. **Впровадження патерну CQRS.** Завдяки використанню бібліотеки MediatR, операції читання (Queries) та запису (Commands) були розділені. Це значно спростило логіку контролерів, зробивши їх тонкими, та дозволило оптимізувати кожен тип запитів окремо.
3. **Реалізація механізмів безпеки.** Систему автентифікації побудовано на основі JWT-токенів, що є сучасним стандартом для створення RESTful API. Крім того, впроваджено контроль доступу на рівні ресурсів (Resource-based Authorization), що гарантує захист конфіденційних даних — користувачі можуть керувати лише власними профілями, тоді як повний доступ надається виключно адміністраторам платформи.
4. **Проєктування та ініціалізація бази даних.** Використано ORM-фреймворк Entity Framework Core з підходом Code-First, що спростило управління схемою бази даних за допомогою міграцій. Для полегшення процесу розробки та тестування було реалізовано механізм первинного наповнення бази даних (Seeding) тестовими

об'єктами, включаючи користувачів, оголошення та ключові орієнтири міста Острог.

У результаті виконання завдань цього розділу було створено повноцінне, надійне та захищене API, яке повністю задовольняє вимоги до серверної частини застосунку. Розроблена архітектура дозволяє легко розширювати функціонал платформи в майбутньому, додавати нові модулі та інтегрувати систему з іншими сервісами без необхідності переписування існуючого коду.

РОЗДІЛ 3. ТЕСТУВАННЯ, РОЗГОРТАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ СИСТЕМИ

3.1 Тестування вебзастосунку

Тестування є невід'ємною частиною життєвого циклу розробки програмного забезпечення, що дозволяє виявити логічні помилки, перевірити стабільність архітектури та забезпечити безперебійну роботу системи. Враховуючи специфіку платформи «UniStay» та її фокус на взаємодії користувачів у реальному часі, у межах цього проєкту було застосовано комплексне ручне тестування API та клієнтської частини.

Мета тестування:

- Перевірити коректність роботи REST API ендпоінтів та їх захищеність (авторизацію).
- Переконатися у правильній роботі бізнес-логіки системи (CQRS-команд та запитів).
- Перевірити інтеграцію клієнтської частини з сервером, включаючи роботу WebSockets (SignalR) для чатів.
- Протестувати валідацію форм і відображення інтерактивних карт (Leaflet).

Інструменти тестування:

- **Swagger UI** — інтегрований інструмент в ASP.NET Core для швидкого тестування та самодокументування API безпосередньо з браузера.
- **Postman** — для складних сценаріїв тестування API з передачею JWT-токенів у заголовках (Bearer Token).
- **Інструменти розробника (Browser DevTools)** — для моніторингу мережевих запитів (Fetch/XHR), перевірки підключень WebSockets та локального сховища.
- **React Developer Tools** — для відстеження стану компонентів і React Context.

Тестування серверної частини (Backend)

За допомогою Swagger та Postman були перевірені ключові контролери системи. Особлива увага приділялася перевірці рольової моделі (Resource-based Authorization). Перевірялося, чи може користувач з роллю User редагувати чужі оголошення (очікувано повертався статус 403 Forbidden). Успішно протестовані всі CRUD-операції для оголошень, обробка зображень та система автентифікації.

Тестування клієнтської частини (Frontend)

Проведено ручне проходження основних клієнтських сценаріїв (User Flows). Зокрема:

- Реєстрація та авторизація з виведенням відповідних повідомлень при помилках вводу.
- Навігація платформою за допомогою React Router v7 без перезавантаження сторінок.
- Коректне відображення маркерів об'єктів нерухомості на карті (через React Leaflet).
- Миттєва відправка та отримання повідомлень у чаті завдяки інтеграції з @microsoft/signalr.

Таблиця 3.1 — Приклади тестових сценаріїв платформи «UniStay»

Сценарій	**Вхідні дані**	**Очікуваний результат**	**Статус**
Реєстрація нового користувача	Валідні email, пароль, ім'я	HTTP 201 Created, отримання JWT-токена	Успішно
Спроба доступу до чужого профілю	JWT-токен користувача A, запит до профілю B	HTTP 403 Forbidden	Успішно
Отримання списку оголошень	GET запит до /api/listings	HTTP 200 OK, JSON-масив об'єктів	Успішно

Сценарій	**Вхідні дані**	**Очікуваний результат**	**Статус**
Перевірка роботи чату (SignalR)	Відправка тексту в компоненті чату	Миттєва поява повідомлення у співрозмовника	Успішно
Відображення карти	Перехід на сторінку з картою	Завантаження тайлів Leaflet, відображення маркерів	Успішно
Створення оголошення без авторизації	POST запит без Bearer токена	HTTP 401 Unauthorized	Успішно

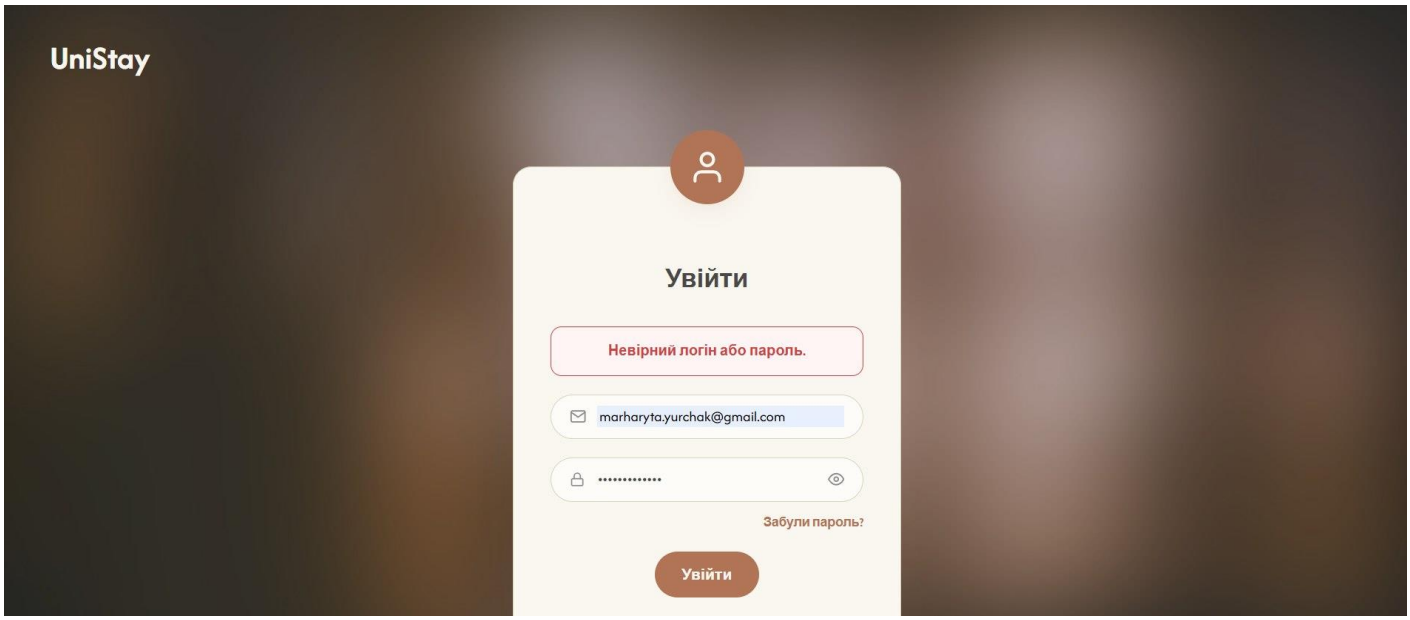


Рис. 3.1. Перевірка валідації даних на клієнтській стороні застосунку

3.2 Розгортання вебзастосунку та організація середовища

Оскільки над проєктом працювала команда з двох розробників (Backend та Frontend), критично важливим було налаштувати зручне, відтворюване та ізольоване середовище для розробки і тестування. Розгортання проєкту здійснювалося локально з використанням сучасних практик контейнеризації та систем контролю версій.

Етапи розгортання та налаштування:

1. **Організація спільної роботи (Git & GitHub):**

Увесь вихідний код проєкту було розміщено в єдиному репозиторії на GitHub. Це дозволило frontend-розробнику стягувати актуальні оновлення серверної частини, запускати її локально та працювати з єдиною кодовою базою, уникаючи конфліктів інтеграції.

2. **Контейнеризація (Docker):**

Для забезпечення ідентичності середовищ розробки було застосовано платформу Docker. За допомогою спеціальних команд було створено та розгорнуто контейнери у

застосунку Docker Desktop. Це дозволило розгорнути локальну базу даних PostgreSQL без необхідності ручного встановлення СУБД на комп'ютери розробників.

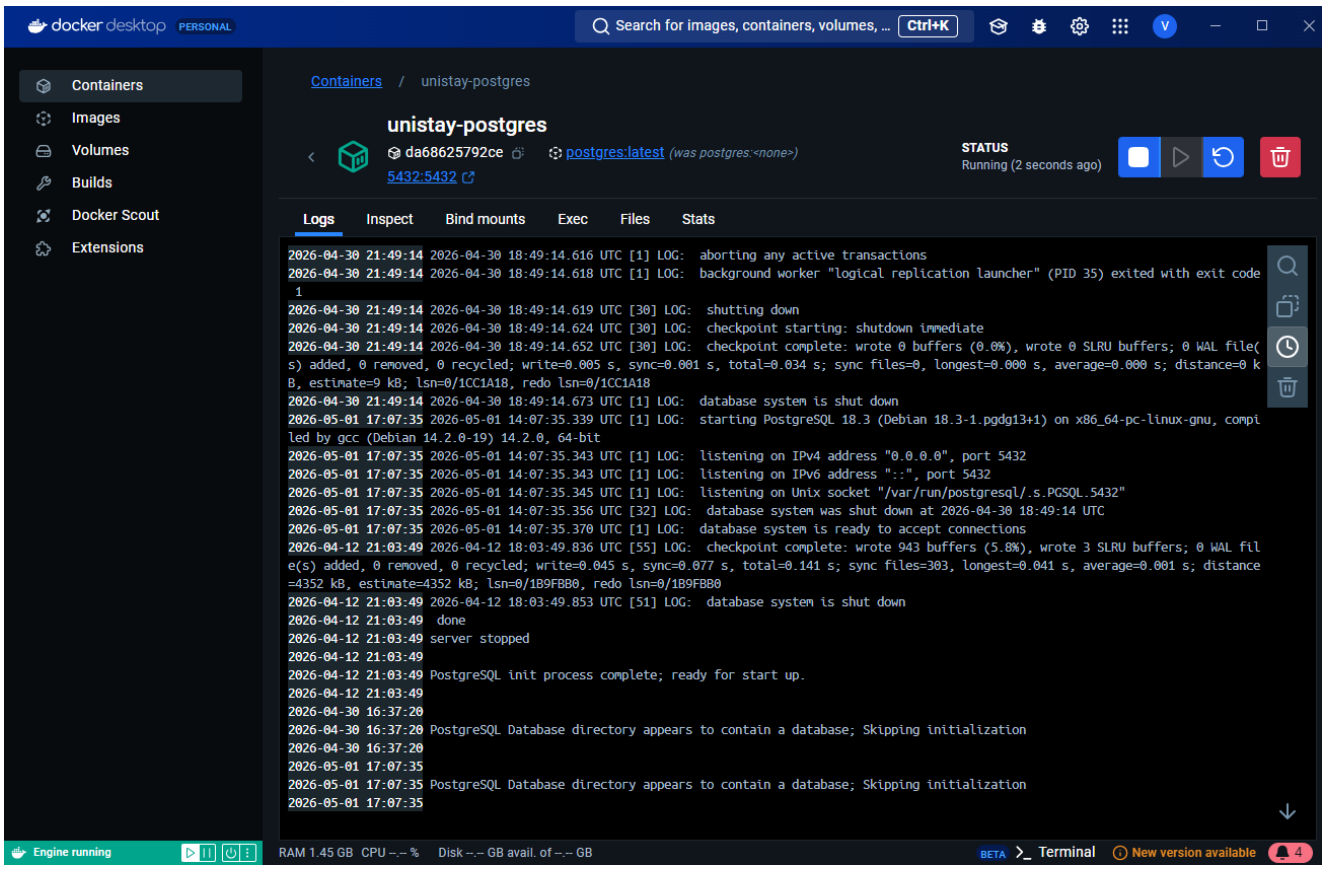


Рис. 3.2. Запущений контейнер бази даних PostgreSQL у середовищі Docker

3. **Запуск серверної частини (.NET):**

Бекенд запускався локально за допомогою вбудованого вебсервера Kestrel (або IIS Express). Для підключення до бази даних використовувався рядок підключення у appsettings.json, який вказував на Docker-контейнер з PostgreSQL. Після запуску Entity Framework Core автоматично застосовував міграції та наповнював локальну базу тестовими даними (Seeding).

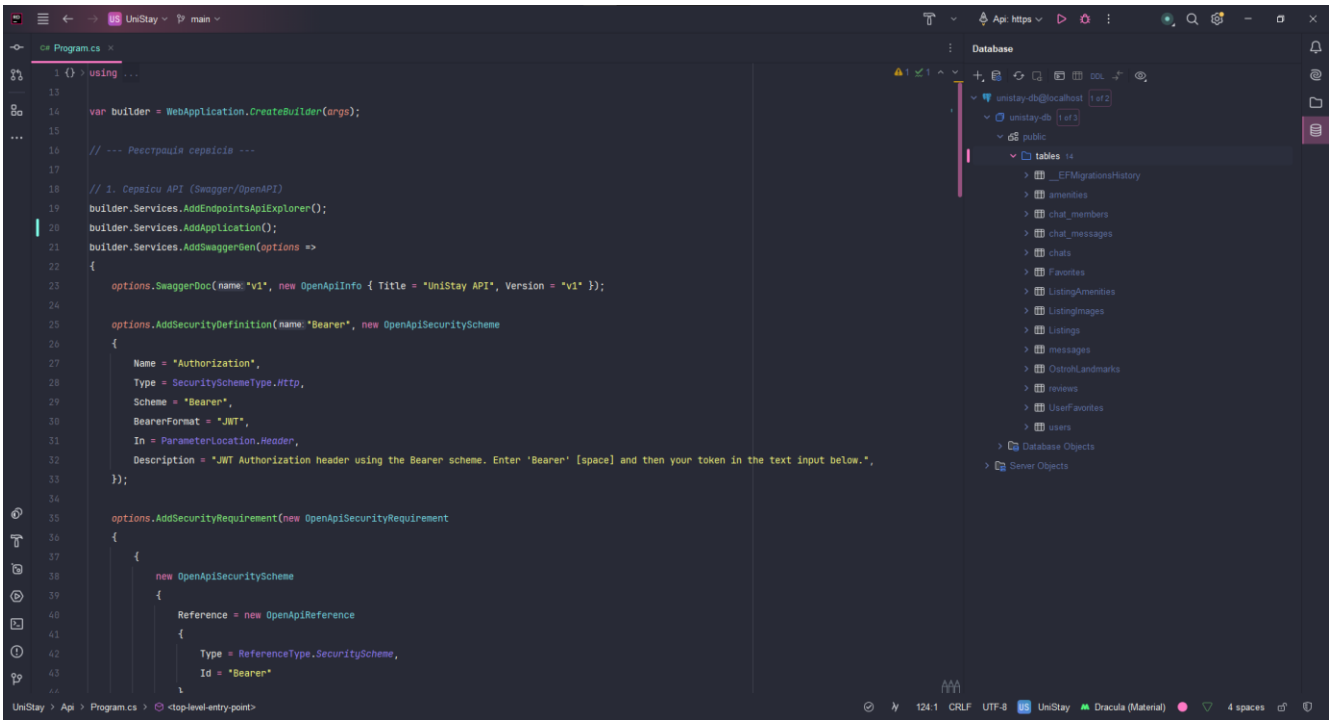


Рис. 3.3. Схема згенерованих таблиць бази даних платформи

4. **Запуск клієнтської частини (React + Vite):**

Frontend-частина компілювалася та запускалася за допомогою надшвидкого збирача Vite (npm run dev). Усі API-запити до локального сервера .NET здійснювалися через бібліотеку Axios, а для уникнення проблем з політикою однакового джерела (CORS), на сервері було налаштовано відповідні дозволи для localhost.

Такий підхід до локального розгортання дозволив створити гнучку екосистему розробки, яку в майбутньому можна легко перенести на хмарні сервери (наприклад, AWS, Azure або Render) завдяки наявності контейнеризації бази даних та чіткому розподілу на клієнтську і серверну частини.

3.3 Оцінка ефективності розробленої системи

Розроблена інформаційна система «UniStay» є сучасним, високопродуктивним рішенням, яке повністю задовольняє початкові вимоги щодо надійності, масштабованості та зручності використання.

Ефективність системи обґрунтовується продуманим вибором технологічного стека та архітектурних патернів:

1. Ефективність серверної архітектури (Backend)

Використання фреймворку **ASP.NET Core** разом із підходом **Clean Architecture** (Чиста архітектура) дозволило створити систему, стійку до змін. Завдяки патерну **CQRS** (за допомогою бібліотеки MediatR) відбулося чітке розділення операцій читання та запису. Це означає, що важкі запити до бази даних (наприклад, фільтрація оголошень) не впливають на продуктивність операцій запису. Інтеграція бази даних **PostgreSQL** через **Entity Framework Core** гарантує цілісність реляційних даних та стійкість до високих навантажень.

2. Ефективність клієнтської частини (Frontend)

Клієнтський застосунок розроблено на найновішій версії **React 19** з використанням збирача **Vite**. Це забезпечило миттєвий час гарячого перезавантаження під час розробки (HMR) та оптимізовану збірку для кінцевого користувача.

Відмова від важких UI-бібліотек (на кшталт Material UI чи Tailwind) на користь **CSS Modules** дала розробникам повний контроль над кожним пікселем інтерфейсу. Інтерфейс є унікальним, легковаговим і не перевантажує браузер клієнта зайвим кодом. Для управління станом замість надлишкового Redux успішно застосовано нативний **React Context**, що є ідеальним та найшвидшим рішенням для поточного масштабу проєкту.

3. Комунікація в реальному часі та система управління чатами

Однією з найскладніших та найефективніших частин системи є модуль обміну повідомленнями, реалізований на базі бібліотеки @microsoft/signalr. На відміну від традиційних REST-запитів, які працюють за принципом «запит-відповідь», SignalR використовує протокол WebSockets, що забезпечує постійне дуплексне (двостороннє) з'єднання.

Основні переваги реалізованої системи комунікації:

- Миттєвість (Real-time): Повідомлення з’являються у отримувача в момент відправки без необхідності оновлення сторінки чи періодичних запитів до сервера (polling).
- Групові чати та управління учасниками: Система дозволяє не лише вести приватні діалоги, а й створювати групові бесіди. Реалізована логіка динамічного додавання учасників до існуючого чату, що є критично важливим для студентів, які планують спільну оренду (co-living).
- Стійкість з’єднання: SignalR автоматично обробляє розриви зв’язку, намагаючись перепідключитися, та підтримує транспортні рівні (Server-Sent Events або Long Polling), якщо браузер не підтримує WebSockets.

На рисунку 3.5 продемонстровано інтерфейс чату із функціоналом управління учасниками бесіди, що підтверджує високу інтерактивність платформи.

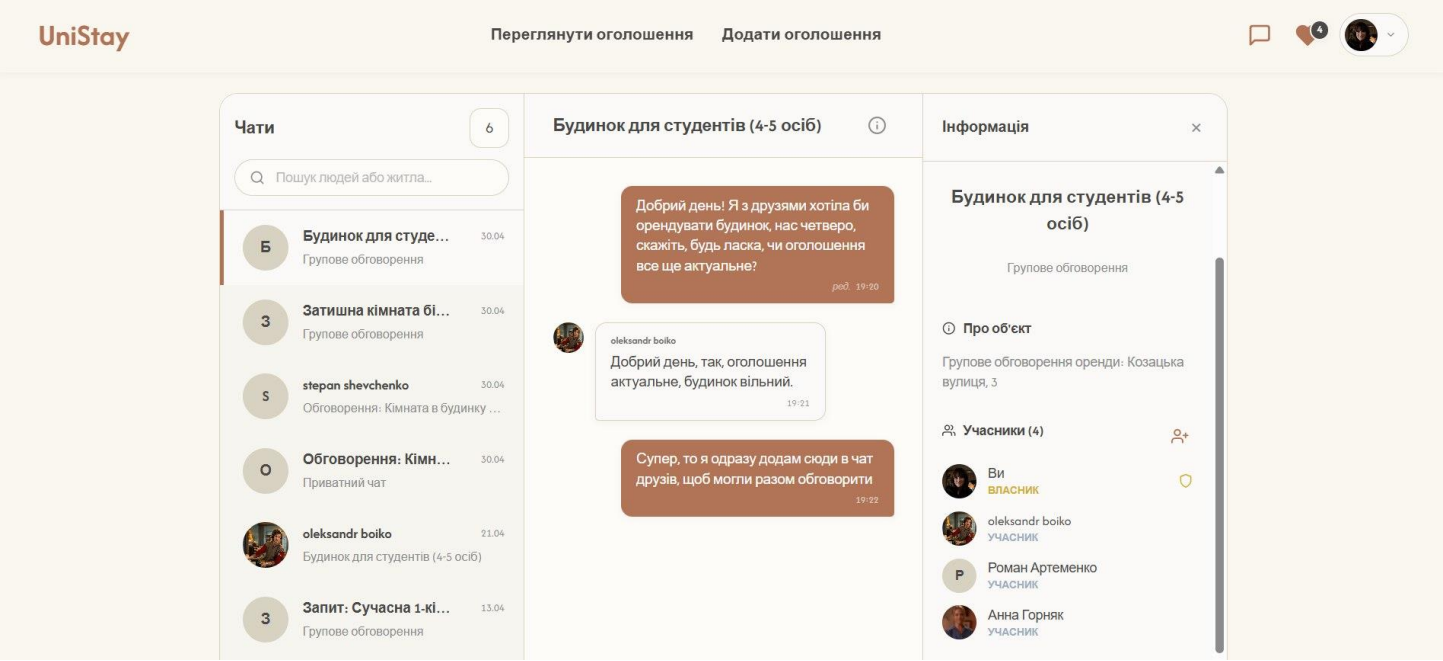


Рис. 3.4. Реалізація групового чату з функцією управління учасниками на основі SignalR

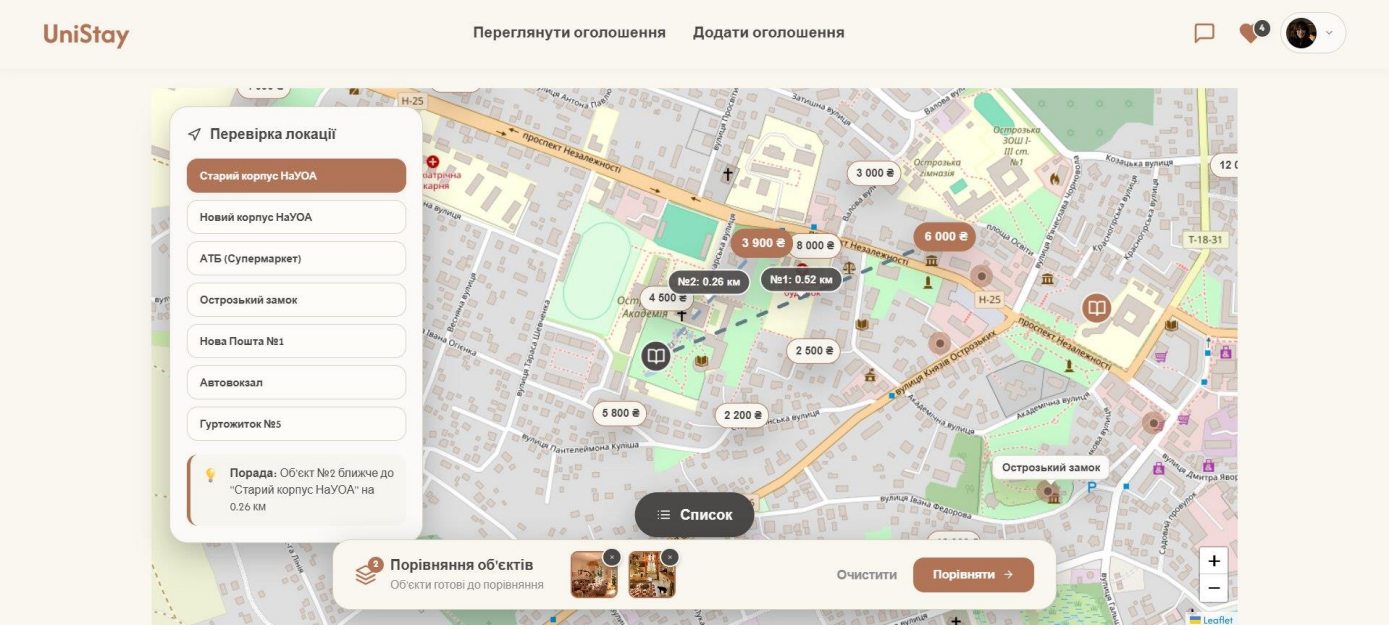


Рис. 3.5. Відображення об'єктів нерухомості на інтерактивній карті Leaflet у клієнтській частині застосунку

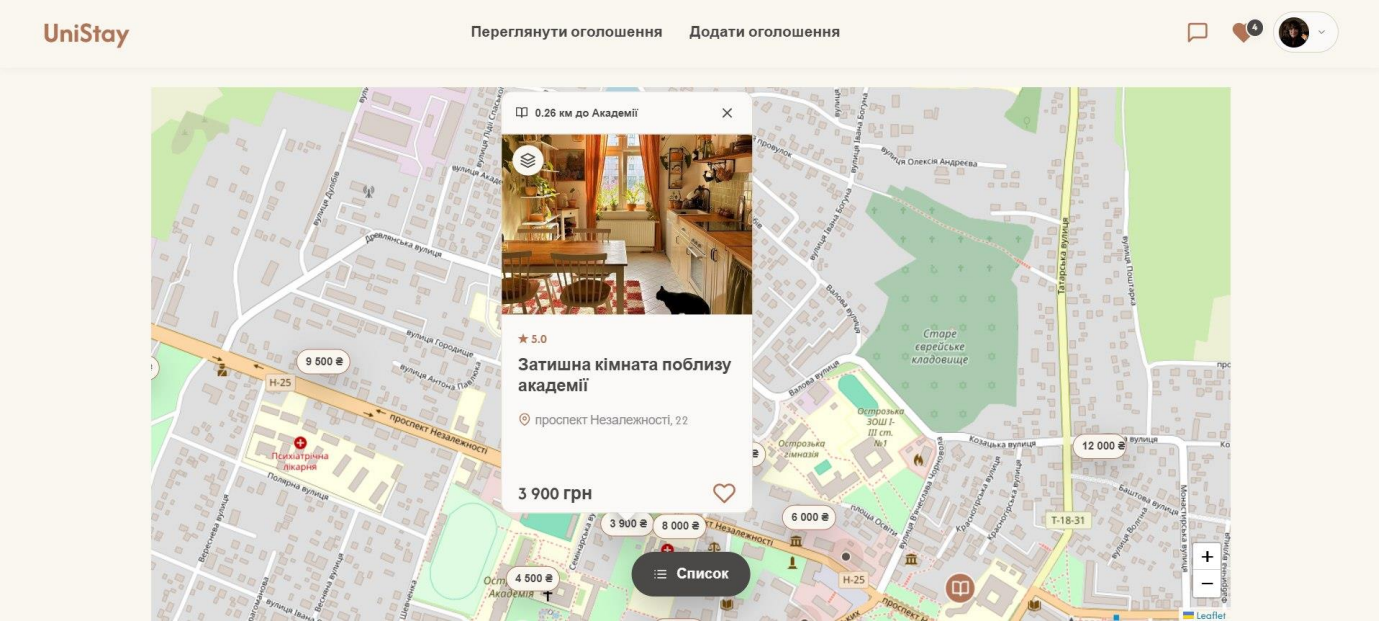


Рис. 3.6. Відображення додаткової відомості про об'єкт нерухомості на інтерактивній карті Leaflet

Висновки до розділу 3

У третьому розділі було виконано завершальний етап життєвого циклу розробки програмного забезпечення (SDLC), що включав комплексне тестування, налаштування ізольованого середовища розгортання та ґрунтовну оцінку загальної технічної ефективності розробленої вебплатформи «UniStay».

Процес перевірки якості (Quality Assurance) складався з детального ручного тестування як клієнтської, так і серверної частин, що підтвердило стабільну та безвідмовну роботу всіх ключових модулів системи.

- **На рівні бекенду:** Перевірка REST API за допомогою інтерактивної документації Swagger та колекцій запитів Postman довела високу надійність реалізованої рольової моделі авторизації (RBAC). Було успішно протестовано механізми захисту маршрутів, процеси валідації JWT-токенів (включно з обробкою випадків їх закінчення) та коректність роботи бізнес-логіки при спробах несанкціонованого доступу.

- **На рівні фронтенду:** Клієнтська частина успішно пройшла перевірку на коректну валідацію вхідних даних на стороні користувача (Client-side validation) та граматичну обробку помилок, що надходять від сервера. Було підтверджено безперебійну взаємодію з інтерактивними картами та миттєвий обмін повідомленнями в реальному часі. Особливу увагу було приділено тестуванню адаптивності інтерфейсу (Responsive Design) на різних роздільних здатностях екранів, що гарантує інтуїтивно зрозумілий користувацький досвід на будь-яких пристроях.

Процедура локального розгортання (Local Deployment) продемонструвала високу гнучкість, портативність та незалежність системи від апаратного забезпечення. Завдяки використанню технології Docker для контейнеризації реляційної бази даних PostgreSQL та чіткому архітектурному розділенню на бекенд і фронтенд, вдалося створити повністю відтворюване середовище. Такий підхід нівелює класичну проблему «на моєму комп'ютері це працює», забезпечує ізоляцію процесів та дозволяє зручно керувати конфігураціями через змінні середовища (Environment Variables). Це закладає міцний інженерний фундамент для швидкого перенесення платформи на повноцінні хмарні сервери (наприклад, AWS, Azure або DigitalOcean) та майбутнього впровадження пайплайнів безперервної інтеграції та розгортання (CI/CD).

Оцінюючи результати роботи з точки зору архітектурної та обчислювальної ефективності, можна з упевненістю стверджувати, що обраний технологічний стек є максимально збалансованим та відповідає сучасним індустріальним стандартам. У проєкті свідомо уникнуто зайвих абстракцій чи «важких» інструментів, які б невиправдано збільшували споживання оперативної пам'яті та уповільнювали роботу сервера. Симбіоз високопродуктивної платформи ASP.NET Core на бекенді та чистої бібліотеки React (з інструментом збірки Vite) на фронтенді забезпечив оптимальний час відповіді сервера (Time to First Byte), миттєвий рендеринг компонентів інтерфейсу та

стабільне, відмовостійке з'єднання через протокол WebSockets за допомогою бібліотеки SignalR.

Наразі інформаційна система «UniStay» є повністю працездатною і готова до виконання своїх бізнес-функцій у форматі повноцінного MVP (Minimum Viable Product). Закладена в основу бекенду концепція «Чистої архітектури» (Clean Architecture) та використання патерну розділення відповідальності CQRS гарантують високу модульність коду. Це дозволяє розробникам легко масштабувати проєкт у майбутньому без ризику порушити існуючий функціонал (регресії).

Завдяки слабкій зв'язності компонентів, архітектура готова до **майбутніх розширень**, серед яких:

- Впровадження фонових процесів (Background Workers) для автоматизованих email-розсилок та push-сповіщень про нові повідомлення.
- Інтеграція зовнішніх хмарних сховищ (наприклад, Amazon S3) для оптимізованого збереження та кешування медіафайлів і фотографій житла.
- Підключення захищених платіжних шлюзів для реалізації функціоналу онлайн-бронювання.

Відповідно, розроблений програмний продукт є не лише вирішенням конкретної прикладної задачі для студентів, але й сучасним, масштабованим і надійним IT-рішенням із високим комерційним потенціалом для реального впровадження.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було успішно спроєктовано, розроблено та підготовлено до впровадження вебплатформу «UniStay» — спеціалізований маркетплейс нерухомості. Створений програмний продукт органічно поєднує в собі функціональність для пошуку житла, оптимізовану серверну архітектуру та інструменти для комунікації користувачів у реальному часі. Розроблена система орієнтована на вирішення гострих соціально-побутових потреб студентів Національного університету «Острозька академія», зокрема на спрощення процесу спільної оренди (co-living), що дозволяє суттєво знизити фінансове навантаження на здобувачів освіти.

На етапі проєктування, на основі детального аналізу існуючих рішень на ринку нерухомості (таких як OLX, ЛУН, DIM.RIA), було виявлено їхні критичні недоліки для цільової студентської аудиторії. До таких недоліків віднесено: перевантаженість інтерфейсу зайвою рекламою, відсутність прив'язки до конкретних навчальних корпусів, неможливість пошуку співмешканців та розрізнену комунікацію, яка змушує переходити у сторонні месенджери. Це дослідження дозволило сформувати чіткі функціональні та нефункціональні вимоги до нової системи, які стали надійним фундаментом для проєктування доменної моделі, реляційної бази даних та розробки загальної програмної логіки.

Серверна частина (Backend) платформи реалізована на базі сучасного та високопродуктивного фреймворку ASP.NET Core 8. Впровадження архітектурного підходу Clean Architecture (Чиста архітектура) забезпечило максимальну ізоляцію доменного ядра від інфраструктурних залежностей. Використання патерну CQRS (Command and Query Responsibility Segregation) за допомогою бібліотеки MediatR дозволило чітко розділити операції читання та запису даних. Це не лише спростило підтримку коду, але й дозволило створити оптимізований REST API, здатний витримувати високі навантаження та легко масштабуватися. Взаємодія з даними

побудована на базі надійної об'єктно-реляційної СУБД PostgreSQL із застосуванням сучасного ORM-фреймворку Entity Framework Core (підхід Code-First).

Окрему увагу в серверній розробці приділено аспектам кібербезпеки. Реалізовано надійну систему безстанної (stateless) автентифікації на базі криптографічно підписаних JWT-токенів. Крім того, впроваджено жорсткий контроль доступу на рівні ресурсів (Resource-based Authorization), що гарантує конфіденційність персональних даних користувачів та унеможливлює несанкціоновану зміну інформації.

Клієнтська частина (Frontend) системи побудована з використанням найновішої версії бібліотеки React 19 та сучасного інструменту швидкої збірки Vite, який забезпечив оптимізацію розміру кінцевого бандлу та миттєве гаряче перезавантаження (HMR) під час розробки. Завдяки свідомій відмові від важких UI-фреймворків на користь CSS Modules, вдалося створити унікальний, швидкий та повністю адаптивний SPA-інтерфейс (Single Page Application). Інтерфейс коректно відображається на будь-яких пристроях (Mobile-First підхід), що є критично важливим, оскільки більшість студентів використовують саме смартфони. Управління глобальним станом застосунку реалізовано через нативний React Context API, що виявилося оптимальним і найменш ресурсомістким рішенням для поточного масштабу проєкту.

Ключовим досягненням технічної реалізації роботи стала розробка інтерактивних модулів, які якісно відрізняють «UniStay» від конкурентів. Було успішно спроєктовано та впроваджено систему обміну повідомленнями в реальному часі на базі протоколу WebSockets за допомогою бібліотеки SignalR. Розроблений функціонал групових чатів із можливістю динамічного управління списком учасників та збереженням історії переписки ідеально задовольняє потребу студентів в оперативному обговоренні умов спільного проживання. Крім того, інтеграція інтерактивних карт (бібліотека Leaflet) та впровадження математичних алгоритмів просторового пошуку (зокрема формули Гаверсинуса) дозволили користувачам підбирати житло з урахуванням точної географічної відстані до конкретних навчальних корпусів або гуртожитків.

Після завершення активної фази розробки було проведено повний цикл тестування ендпоінтів API (за допомогою інтерактивної документації Swagger та колекцій Postman) і верифікацію клієнтського інтерфейсу. Для забезпечення відтворюваності середовища, систему було успішно контейнеризовано з використанням технології Docker. Це забезпечило повну ізольованість процесів бази даних та бекенду, мінімізувало проблему «на моєму комп'ютері це працює» та підготувало проєкт до безшовної інтеграції в хмарні сервіси через автоматизовані пайплайни CI/CD.

Таким чином, розроблений вебзастосунок є повноцінним, стабільним MVP-рішенням (Minimum Viable Product), яке готове до дослідної експлуатації. Проєкт демонструє повний життєвий цикл створення сучасної інформаційної системи: від збору вимог до розгортання, із суворим дотриманням інженерних стандартів.

Перспективи подальшого розвитку проєкту включають:

- Інтеграцію з державним сервісом «Дія» для верифікації орендодавців, що гарантуватиме повну відсутність шахрайських схем на платформі.
- Розробку системи рекомендацій на базі алгоритмів машинного навчання (Machine Learning) для автоматичного підбору ідеальних співмешканців за психотипом та побутовими звичками.
- Впровадження платіжних шлюзів для можливості безпечного бронювання житла та сплати завдатку безпосередньо через платформу.
- Створення нативних мобільних застосунків для операційних систем iOS та Android на базі технологій React Native або .NET MAUI.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2017. 432 p.
2. Price M. J. C# 12 and .NET 8 – Modern Cross-Platform Development Fundamentals. Packt Publishing, 2023. 822 p.
3. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002. 560 p.
4. Esposito D. Clean Architecture with .NET. Microsoft Press, 2024. 350 p.
5. Seemann M., van Deursen S. Dependency Injection Principles, Practices, and Patterns. Manning Publications, 2019. 552 p.
6. ASP.NET Core Documentation [Електронний ресурс] / Microsoft Learn. – Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/ (дата звернення: 02.05.2024).
7. Entity Framework Core Documentation [Електронний ресурс] / Microsoft Learn. – Режим доступу: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 05.05.2024).
8. Introduction to SignalR [Електронний ресурс] / Microsoft Learn. – Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/signalr/introduction (дата звернення: 08.05.2024).
9. PostgreSQL: The World's Most Advanced Open Source Relational Database [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs/ (дата звернення: 10.05.2024).
10. MediatR Wiki. Simple, unambitious mediator implementation in .NET [Електронний ресурс] / GitHub. – Режим доступу: https://github.com/jbogard/MediatR/wiki (дата звернення: 11.05.2024).
11. CQRS Pattern (Command and Query Responsibility Segregation) [Електронний ресурс] / Microsoft Azure Architecture Center. – Режим доступу:

https://learn.microsoft.com/en-us/azure/architecture/patterns/cqrs (дата звернення: 12.05.2024).

12. Docker Documentation. Official Docker Guide [Електронний ресурс]. – Режим доступу: https://docs.docker.com/ (дата звернення: 14.05.2024).
13. Swagger (OpenAPI) Specification [Електронний ресурс]. – Режим доступу: https://swagger.io/specification/ (дата звернення: 15.05.2024).
14. JSON Web Token (JWT) Introduction [Електронний ресурс]. – Режим доступу: https://jwt.io/introduction (дата звернення: 16.05.2024).
15. Web API design best practices [Електронний ресурс] / Microsoft Azure. – Режим доступу: https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design (дата звернення: 17.05.2024).
16. React: The library for web and native user interfaces [Електронний ресурс]. – Режим доступу: https://react.dev/ (дата звернення: 18.05.2024).
17. Vite: Next Generation Frontend Tooling [Електронний ресурс]. – Режим доступу: https://vitejs.dev/ (дата звернення: 19.05.2024).
18. Leaflet: an open-source JavaScript library for mobile-friendly interactive maps [Електронний ресурс]. – Режим доступу: https://leafletjs.com/ (дата звернення: 20.05.2024).
19. Calculate distance, bearing and more between Latitude/Longitude points (Haversine formula) [Електронний ресурс]. – Режим доступу: https://www.movable-type.co.uk/scripts/latlong.html (дата звернення: 22.05.2024).
20. React Router Documentation [Електронний ресурс]. – Режим доступу: https://reactrouter.com/ (дата звернення: 23.05.2024).
21. Axios HTTP Client [Електронний ресурс]. – Режим доступу: https://axios-http.com/ (дата звернення: 24.05.2024).

22. Clean Architecture Solution Template for .NET [Електронний ресурс] / GitHub. – Режим доступу: https://github.com/jasontaylordev/CleanArchitecture (дата звернення: 25.05.2024).

ДОДАТКИ

ДОДАТОК А - Фрагменти вихідного коду серверної частини (ASP.NET Core)

Лістинг А.1. Реалізація доменної бізнес-логіки чату на базі DDD

```
using Domain.Users;
namespace Domain.Chats;
public class Chat
{
    public ChatId Id { get; }
    public string Name { get; private set; }
    public string? Description { get; private set; }
    public ChatEnums.ChatType Type { get; private set; }
    public UserId CreatedById { get; private set; }
    public User? CreatedBy { get; }
    public DateTime CreatedAt { get; }
    public DateTime? UpdatedAt { get; private set; }
    public bool IsActive { get; private set; }

    public List<ChatMember> Members { get; private set; } = new();
    public List<ChatMessage> Messages { get; private set; } = new();

    private Chat(ChatId id, string name, string? description,
ChatEnums.ChatType type, UserId createdById, DateTime createdAt)
    {
        Id = id;
        Name = name;
        Description = description;
        Type = type;
        CreatedById = createdById;
        CreatedAt = createdAt;
        IsActive = true;
```

```
    }

    public static Chat New(ChatId id, string name, string? description,
ChatEnums.ChatType type, UserId createdById)
        => new(id, name, description, type, createdById, DateTime.UtcNow);

    public void UpdateDetails(string name, string? description)
    {
        Name = name;
        Description = description;
        UpdatedAt = DateTime.UtcNow;
    }

    public void AddMember(ChatMember member)
    {
        if (Members.Any(m => m.UserId == member.UserId && m.IsActive))
            throw new InvalidOperationException("User is already a member
of this chat");

        Members.Add(member);
        UpdatedAt = DateTime.UtcNow;
    }

    public void RemoveMember(UserId userId)
    {
        var member = Members.FirstOrDefault(m => m.UserId == userId &&
m.IsActive);
        if (member == null)
            throw new InvalidOperationException("User is not a member of
this chat");

        member.Leave();
```

```
        UpdatedAt = DateTime.UtcNow;
    }

    public void AddMessage(ChatMessage message)
    {
        if (!Members.Any(m => m.UserId == message.SenderId && m.IsActive))
            throw new InvalidOperationException("User is not a member of
this chat");

        Messages.Add(message);
        UpdatedAt = DateTime.UtcNow;
    }

    public void PromoteMember(UserId userId, ChatEnums.ChatMemberRole
newRole)
    {
        var member = Members.FirstOrDefault(m => m.UserId == userId &&
m.IsActive);
        if (member == null)
            throw new InvalidOperationException("User is not a member of
this chat");

        member.ChangeRole(newRole);
        UpdatedAt = DateTime.UtcNow;
    }

    public void Deactivate()
    {
        IsActive = false;
        UpdatedAt = DateTime.UtcNow;
    }
}
```

Лістинг А.2. Реалізація обміну повідомленнями в реальному часі через SignalR Hub

```
using Microsoft.AspNetCore.SignalR;
using Microsoft.AspNetCore.Authorization;
using System.Security.Claims;
using Api.Dtos;
using Application.Common.Interfaces.Queries;
using Domain.Chats;
using Domain.Users;

namespace Api.Hubs
{
    [Authorize]
    public class ChatHub : Hub<IChatClient>
    {
        private readonly IChatsQueries _chatsQueries;
        private readonly ILogger<ChatHub> _logger;

        public ChatHub(IChatsQueries chatsQueries, ILogger<ChatHub>
logger)
        {
            _chatsQueries = chatsQueries ?? throw new
ArgumentNullException(nameof(chatsQueries));
            _logger = logger ?? throw new
ArgumentNullException(nameof(logger));
        }

        /// <summary>
        /// Підключення до чату
        /// </summary>
        public async Task JoinChat(Guid chatId)
```

```
        {
            var userId = GetCurrentUserId();
            if (userId == null)
            {
                _logger.LogWarning("Unauthorized user tried to join chat
{ChatId}", chatId);
                return;
            }

            var chatIdDomain = new ChatId(chatId);
            var userIdDomain = new UserId(userId.Value);

            // Перевіряємо чи користувач є членом чату
            var isMember = await _chatsQueries.IsUserMember(chatIdDomain,
userIdDomain, Context.ConnectionAborted);
            if (!isMember)
            {
                _logger.LogWarning("User {UserId} is not a member of chat
{ChatId}", userId, chatId);
                throw new HubException("You are not a member of this
chat");
            }

            // Додаємо користувача до групи чату
            await Groups.AddToGroupAsync(Context.ConnectionId,
GetChatGroupName(chatId));

            _logger.LogInformation("User {UserId} joined chat {ChatId}
with connection {ConnectionId}",
                userId, chatId, Context.ConnectionId);
        }
```

```
        /// <summary>
        /// Відключення від чату
        /// </summary>
        public async Task LeaveChat(Guid chatId)
        {
            await Groups.RemoveFromGroupAsync(Context.ConnectionId,
GetChatGroupName(chatId));

            var userId = GetCurrentUserId();
            _logger.LogInformation("User {UserId} left chat {ChatId} with
connection {ConnectionId}",
                userId, chatId, Context.ConnectionId);
        }

        /// <summary>
        /// Сповіщення про те, що користувач друкує
        /// </summary>
        public async Task NotifyTyping(Guid chatId)
        {
            var userId = GetCurrentUserId();
            if (userId == null) return;

            var userName = Context.User?.FindFirst(ClaimTypes.Name)?.Value
?? "Unknown";

            await Clients
                .OthersInGroup(GetChatGroupName(chatId))
                .UserTyping(chatId, userId.Value, userName);
        }

        /// <summary>
        /// Сповіщення про те, що користувач перестав друкувати
```

```
        /// </summary>
        public async Task NotifyStoppedTyping(Guid chatId)
        {
            var userId = GetCurrentUserId();
            if (userId == null) return;

            await Clients
                .OthersInGroup(GetChatGroupName(chatId))
                .UserStoppedTyping(chatId, userId.Value);
        }

        public override async Task OnConnectedAsync()
        {
            var userId = GetCurrentUserId();
            _logger.LogInformation("User {UserId} connected with
connection {ConnectionId}",
                userId, Context.ConnectionId);

            await base.OnConnectedAsync();
        }

        public override async Task OnDisconnectedAsync(Exception?
exception)
        {
            var userId = GetCurrentUserId();
            _logger.LogInformation("User {UserId} disconnected with
connection {ConnectionId}. Exception: {Exception}",
                userId, Context.ConnectionId, exception?.Message);

            await base.OnDisconnectedAsync(exception);
        }
```

```
        private Guid? GetCurrentUserId()
        {
            var userIdClaim =
Context.User?.FindFirst(ClaimTypes.NameIdentifier)?.Value;
            return Guid.TryParse(userIdClaim, out var userId) ? userId :
null;
        }

        private static string GetChatGroupName(Guid chatId) =>
$"chat_{chatId}";
    }
}
```

Лістинг А.3. Реалізація обробника авторизації на базі патерну CQRS

```
using MediatR;
     using Application.Common.Interfaces.Auth;
     using Application.Common.Interfaces.Queries;
     using Application.Users.Exceptions;
     using Application.Auth.Dto;
     using Application.Common;
     using Optional.Unsafe;

     namespace Application.Auth.Commands
     {
         public class LoginUserCommandHandler :
     IRequestHandler<LoginUserCommand, Result<AuthResultDto,
     UserException>>
         {
             private readonly IUsersQueries _usersQueries;
             private readonly IPasswordHash _passwordHash;
```

```
        private readonly IJwtGenerator _jwtGenerator;

        public LoginUserCommandHandler(IUsersQueries usersQueries,
IPasswordHash passwordHash, IJwtGenerator jwtGenerator)
        {
            _usersQueries = usersQueries;
            _passwordHash = passwordHash;
            _jwtGenerator = jwtGenerator;
        }

        public async Task<Result<AuthResultDto, UserException>>
Handle(LoginUserCommand request, CancellationToken cancellationToken)
        {
            var userOption = await
_usersQueries.GetByEmail(request.Email, cancellationToken);

            if (!userOption.HasValue)
            {
                return Result<AuthResultDto,
UserException>.Failure(UserException.InvalidCredentials("Invalid
email or password."));
            }

            var user = userOption.ValueOrFailure();

            if (!_passwordHash.VerifyPassword(user.Password,
request.Password))
            {
```

```
                return Result<AuthResultDto,
    UserException>.Failure(UserException.InvalidCredentials("Invalid
    email or password."));
            }

            var fullName = $"{user.FirstName} {user.LastName}";

            var token = _jwtGenerator.GenerateToken(
                user.Id,
                user.Email,
                user.Role.ToString(),
                fullName,
                user.ProfileImage
            );

            return Result<AuthResultDto, UserException>.Success(new
    AuthResultDto { Token = token });
        }
    }
}
```

Лістинг А.4. Правила перевірки вхідних даних за допомогою FluentValidation

```
using FluentValidation;

namespace Application.Users.Commands
{
    public class CreateUserCommandValidator :
AbstractValidator<AdminCreateUserCommand>
    {
        public CreateUserCommandValidator()
```

```
        {
            RuleFor(x => x.FirstName)
                .NotEmpty().WithMessage("Ім'я є обов'язковим.")
                .MaximumLength(100).WithMessage("Ім'я не може перевищувати
100 символів.");

            RuleFor(x => x.LastName)
                .NotEmpty().WithMessage("Прізвище є обов'язковим.")
                .MaximumLength(100).WithMessage("Прізвище не може
перевищувати 100 символів.");

            RuleFor(x => x.Email)
                .NotEmpty().WithMessage("Email є обов'язковим.")
                .MaximumLength(255).WithMessage("Email не може
перевищувати 255 символів.")
                .EmailAddress().WithMessage("Неправильний формат Email.");

            RuleFor(x => x.Password)
                .NotEmpty().WithMessage("Пароль є обов'язковим.")
                .MinimumLength(8).WithMessage("Пароль має містити
щонайменше 8 символів.")
                .MaximumLength(100).WithMessage("Пароль не може
перевищувати 100 символів.");

            RuleFor(x => x.PhoneNumber)
                .MaximumLength(20).When(x =>
!string.IsNullOrEmpty(x.PhoneNumber))
                .WithMessage("Номер телефону не може перевищувати 20
символів.");

            RuleFor(x => x.ProfileImage)
                .MaximumLength(500).When(x =>
```

```
!string.IsNullOrEmpty(x.ProfileImage))
                .WithMessage("URL зображення профілю не може перевищувати
500 символів.");
        }
    }
}
```

Лістинг А.5. Реалізація перевірки доступу на рівні ресурсів (Controllers)

```
private Guid? GetAuthenticatedUserId()
  {
      var userIdString = User.FindFirstValue(ClaimTypes.NameIdentifier);
      return Guid.TryParse(userIdString, out var userId) ? userId : null;
  }

  private bool IsAdminUser() => User.IsInRole("Administrator");

  private bool CanAccessUser(Guid requestedUserId)
  {
      var authUserId = GetAuthenticatedUserId();
      if (!authUserId.HasValue) return false;

      return IsAdminUser() || authUserId.Value == requestedUserId;
  }

  [HttpGet("{userId:guid}")]
  [ProducesResponseType(typeof(UserDto), StatusCodes.Status200OK)]
  [ProducesResponseType(StatusCodes.Status403Forbidden)]
  [ProducesResponseType(StatusCodes.Status404NotFound)]
```

```
  public async Task<ActionResult<UserDto>> GetUserById([FromRoute] Guid
userId, CancellationToken cancellationToken)
  {
      // Перевірка доступу до конкретного ресурсу
      if (!CanAccessUser(userId))
      {
          return Forbid();
      }

      var userOption = await _usersQueries.GetById(new UserId(userId),
cancellationToken);

      return userOption.Match<ActionResult<UserDto>>(
          user => Ok(UserDto.FromDomainModel(user)),
          () => NotFound(new { Message = $"User with id {userId} not
found." })
      );
  }
```

Лістинг А.6. Математичний алгоритм обчислення відстані по координатах

```
  namespace Domain.Listings
{
    public class OstrohLandmark
    {
        public OstrohLandmarkId Id { get; private set; }
        public string Name { get; private set; }
        public string Address { get; private set; }
        public double Latitude { get; private set; }
        public double Longitude { get; private set; }

        private OstrohLandmark(OstrohLandmarkId id, string name, string
```

```
address, double latitude, double longitude)
        {
            Id = id;
            Name = name;
            Address = address;
            Latitude = latitude;
            Longitude = longitude;
        }

        public static OstrohLandmark Create(OstrohLandmarkId id, string
name, string address, double latitude, double longitude)
        {
            return new OstrohLandmark(id, name, address, latitude,
longitude);
        }

        public double CalculateDistanceTo(double latitude, double
longitude)
        {
            const double EarthRadiusKm = 6371.0;

            var lat1Rad = Latitude * Math.PI / 180.0;
            var lat2Rad = latitude * Math.PI / 180.0;
            var deltaLat = (latitude - Latitude) * Math.PI / 180.0;
            var deltaLon = (longitude - Longitude) * Math.PI / 180.0;

            var a = Math.Sin(deltaLat / 2) * Math.Sin(deltaLat / 2) +
                    Math.Cos(lat1Rad) * Math.Cos(lat2Rad) *
                    Math.Sin(deltaLon / 2) * Math.Sin(deltaLon / 2);

            var c = 2 * Math.Atan2(Math.Sqrt(a), Math.Sqrt(1 - a));
```

```
            return EarthRadiusKm * c;
        }
    }
}
```

Лістинг А.7. Розділення логіки читання та запису в репозиторії

```
using Application.Common.Interfaces.Queries;
using Application.Common.Interfaces.Repositories;
using Domain.Users;
using Infrastructure.Persistence;
using Microsoft.EntityFrameworkCore;
using Optional;
using Optional.Async.Extensions;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;

namespace Infrastructure.Persistence.Repositories
{
    public class UsersRepository : IUsersRepository, IUsersQueries
    {
        private readonly ApplicationDbContext _context;

        public UsersRepository(ApplicationDbContext context)
        {
            _context = context ?? throw new
ArgumentNullException(nameof(context));
        }
```

```
        public async Task<User> Add(User user, CancellationToken
cancellationToken)
        {
            await _context.Users.AddAsync(user, cancellationToken);
            await _context.SaveChangesAsync(cancellationToken);
            return user;
        }

        public async Task<User> Update(User user, CancellationToken
cancellationToken)
        {
            _context.Users.Update(user);
            await _context.SaveChangesAsync(cancellationToken);
            return user;
        }

        public async Task<User> Delete(User user, CancellationToken
cancellationToken)
        {
            _context.Users.Remove(user);
            await _context.SaveChangesAsync(cancellationToken);
            return user;
        }

        async Task<Option<User>> IUsersRepository.GetById(UserId id,
CancellationToken cancellationToken)
        {
            var user = await _context.Users.FirstOrDefaultAsync(u => u.Id
== id, cancellationToken);
            return user.SomeNotNull();
        }
```

```
        async Task<Option<User>> IUsersRepository.GetByEmail(string email,
CancellationToken cancellationToken)
        {
            var user = await _context.Users.FirstOrDefaultAsync(u =>
u.Email == email, cancellationToken);
            return user.SomeNotNull();
        }


        public async Task<IReadOnlyList<User>> GetAll(CancellationToken
cancellationToken)
        {
            return await
_context.Users.AsNoTracking().ToListAsync(cancellationToken);
        }

        async Task<Option<User>> IUsersQueries.GetById(UserId id,
CancellationToken cancellationToken)
        {
            var user = await _context.Users
                                .AsNoTracking()
                                .FirstOrDefaultAsync(u => u.Id == id,
cancellationToken);
            return user.SomeNotNull();
        }

        async Task<Option<User>> IUsersQueries.GetByEmail(string email,
CancellationToken cancellationToken)
        {
            var user = await _context.Users
                                .AsNoTracking()
```

```
                                .FirstOrDefaultAsync(u => u.Email ==
email, cancellationToken);
            return user.SomeNotNull();
        }
    }
}
```

Лістинг А.8. Програмне наповнення бази даних початковим контентом

```
  private async Task SeedOstrohLandmarksAsync()
  {
  if (!_context.OstrohLandmarks.Any())
  {
  _logger.LogInformation("Seeding Ostroh landmarks...");
  var landmarks = new List<OstrohLandmark>
  {
  OstrohLandmark.Create(OstrohLandmarkId.New(), "Новий корпус Острозької
академії", "вул. Семінарська, 2", 50.32917, 26.51278),
  OstrohLandmark.Create(OstrohLandmarkId.New(), "АТБ (Супермаркет)", "вул.
Гальшки Острозької, 1в", 50.32729, 26.52463),
  OstrohLandmark.Create(OstrohLandmarkId.New(), "Острозький замок", "вул.
Академічна, 5", 50.32626, 26.52212),
  OstrohLandmark.Create(OstrohLandmarkId.New(), "Автовокзал \"Острог\"",
"просп. Незалежності, 166", 50.33557, 26.49400),
  OstrohLandmark.Create(OstrohLandmarkId.New(), "Богоявленський собор",
"вул. Академічна, 5в", 50.32661, 26.52128)
  };
  _context.OstrohLandmarks.AddRange(landmarks);
  _logger.LogInformation($"Seeded {landmarks.Count} Ostroh landmarks.");
  }
  }
```