

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Вебплатформа оренди житла»

Виконав студент групи ІПЗ-22-2
БАЛЕНКО Дмитро Сергійович

Керівник роботи:
ЧИЖМОТРЯ Олексій Володимирович

Рецензент:
ВАКАЛЮК Тетяна Анатоліївна

Житомир – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна Вакалюк
«27» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **БАЛЕНКО Дмитро Сергійович**

Керівник роботи: **ЧИЖМОТРЯ Олексій Володимирович**

Тема роботи: **«Вебплатформа оренди житла»**,

затверджена Наказом закладу вищої освіти від «27» лютого 2026 р., № 92/с

Вихідні дані для роботи: розробка вебплатформи оренди житла, що оптимізує пошук об'єктів нерухомості, дозволить автоматизувати процеси бронювання без перетину дат і цифровізувати комунікацію між орендарем та орендодавцем.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Чижмотря О.В.	01.03.2026р.	01.03.2026р
2	Чижмотря О.В.	01.03.2026р	01.03.2026р
3	Чижмотря О.В.	01.03.2026р	01.03.2026р

РЕФЕРАТ

Випускна кваліфікаційна робота присвячена розробці вебплатформи для автоматизації процесів оренди житлової нерухомості. Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел із 30 найменувань (3 стор.), 4 додатків (7 стор.), 6 таблиць та 21 ілюстрації по тексту. Повний обсяг роботи становить 66 сторінок, з них 48 сторінок основного тексту.

Метою роботи є розробка програмного комплексу, що забезпечує пошук та багатокритеріальну фільтрацію об'єктів нерухомості, автоматизацію процесу бронювання із запобіганням перетину дат та двосторонню комунікацію між орендарем і орендодавцем у режимі реального часу.

У роботі проведено аналіз предметної області та існуючих систем оренди, визначено функціональні й нефункціональні вимоги, спроектовано архітектуру за принципами Clean Architecture, реляційну базу даних і алгоритми управління станами заявок. Реалізовано клієнтську частину на React і серверну частину на ASP.NET Core Web API з використанням PostgreSQL, Entity Framework Core та SignalR. Проведено ручне функціональне тестування основних бізнес-процесів, серверна частина та база даних працюють у хмарному середовищі Microsoft Azure, а клієнтська частина під час тестування запускала локально.

КЛЮЧОВІ СЛОВА: АВТОМАТИЗАЦІЯ, БРОНЮВАННЯ, ВЕБПЛАТФОРМА, НЕРУХОМІСТЬ, ОRENDA

					ІПЗ.КР.Б – 121 – 26 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Баленко Д.С.			Вебплатформа оренди житла	Літ.	Арк.	Аркушів
Керівник		Чижмоторя О.В.					3	66
Рецензент		Вакалюк Т.А.						
Зав. каф.		Вакалюк Т.А.				Житомирська політехніка, група ІПЗ-22-2		

ABSTRACT

The graduation qualification work is devoted to the development of a web platform for automating residential real estate rental processes. The thesis consists of an introduction, 3 chapters, conclusions, a list of references containing 30 sources (3 pages), 4 appendices (7 pages), 6 tables, and 21 figures within the text. The total volume of the work is 66 pages, including 48 pages of the main text.

The purpose of the work is to develop a software system that provides search and multi-criteria filtration of real estate objects, automation of the booking process with date overlap prevention, and real-time bilateral communication between the tenant and the landlord.

The work analyzes the subject area and existing rental systems, defines functional and non-functional requirements, and designs the architecture using Clean Architecture principles, a relational database, and application state management algorithms. The client side was implemented using React, and the server side using ASP.NET Core Web API with PostgreSQL, Entity Framework Core, and SignalR. Manual functional testing of the main business processes was performed, the server side and database operate in the Microsoft Azure cloud environment, while the client side was run locally during testing.

KEYWORDS: AUTOMATION, BOOKING, REAL ESTATE, RENT, WEB PLATFORM

					ИПЗ.КР.Б – 121 – 26 – ИПЗ	Арк.
						4

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ У СФЕРІ ОРЕНДИ ЖИТЛА	9
1.1. Постановка задачі автоматизації та аналіз процесів взаємодії суб'єктів ринку нерухомості	9
1.2. Порівняльний аналіз існуючих програмних аналогів систем оренди житла	12
1.3. Обґрунтування вибору архітектури та структури розподіленої вебплатформи	15
1.4. Обґрунтування вибору інструментальних засобів розробки та вимоги до апаратного забезпечення.....	19
Висновки до першого розділу	22
РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБПЛАТФОРМИ ОРЕНДИ ЖИТЛА.....	24
2.1. Проєктування логічної та фізичної структури бази даних	24
2.2. Проєктування та архітектурна структура серверної частини (REST API)...	28
2.3. Проєктування алгоритмів роботи вебсистеми.....	31
2.4. Проєктування та розробка клієнтської частини (фронтенду)	36
Висновки до другого розділу	39
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБПЛАТФОРМОЮ ОРЕНДИ ЖИТЛА.....	41
3.1. Порядок встановлення та налаштування параметрів системи	41
3.2. Структура інтерфейсу вебплатформи.....	44
3.3. Тестування роботи програмного комплексу	50
Висновки до третього розділу	54
ВИСНОВКИ	55
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	56
ДОДАТКИ	59

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ACID – Атомарність, узгодженість, ізолюваність, довговічність (Atomicity, Consistency, Isolation, Durability)

API – Прикладний програмний інтерфейс (Application Programming Interface)

CI/CD – Безперервна інтеграція та безперервне розгортання (Continuous Integration / Continuous Deployment)

DI – Впровадження залежностей (Dependency Injection)

DTO – Об'єкт передачі даних (Data Transfer Object)

E2E – Наскрізне тестування (End-to-End Testing)

JSON – Об'єктна нотація JavaScript (JavaScript Object Notation)

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

LINQ – Мова інтегрованих запитів (Language Integrated Query)

MVP – Мінімально життєздатний продукт (Minimum Viable Product)

ORM – Об'єктно-реляційне відображення (Object-Relational Mapping)

REST – Передача репрезентативного стану (Representational State Transfer)

SPA – Односторінковий застосунок (Single Page Application)

UI – Інтерфейс користувача (User Interface)

UML – Уніфікована мова моделювання (Unified Modeling Language)

UX – Досвід користувача (User Experience)

UUID – Універсальний унікальний ідентифікатор (Universally Unique Identifier)

БД – База даних

СКБД – Система керування базами даних

ВСТУП

Сфера оренди житла швидко переходить у цифровий формат. Користувачі очікують, що пошук житла, перегляд доступних варіантів, бронювання та спілкування з власником будуть виконуватися в одному зручному середовищі. Водночас значна частина оголошень і домовленостей досі ведеться через дошки оголошень, соціальні мережі або сторонні месенджери. Через це користувачам складніше перевіряти актуальність інформації, а орендодавцям – контролювати бронювання та комунікацію з клієнтами.

Основними проблемами таких підходів є розрізненість даних, відсутність автоматичної перевірки доступності житла на обрані дати та ризик подвійного бронювання. Додатковою проблемою є перенесення спілкування між орендарем і орендодавцем за межі платформи, оскільки в такому разі історія домовленостей не зберігається в системі. Для українського ринку також важливо враховувати специфічні параметри житла, наприклад наявність генератора або іншого альтернативного джерела живлення. Саме тому актуальною є розробка вебплатформи, яка поєднує пошук, фільтрацію, бронювання та обмін повідомленнями в межах одного програмного продукту.

Метою кваліфікаційної роботи є розробка вебплатформи оренди житлової нерухомості RentPlatform, яка забезпечує пошук і фільтрацію об'єктів, створення заявок на бронювання з перевіркою перетину дат та комунікацію між орендарем і орендодавцем у режимі реального часу.

Для досягнення поставленої мети необхідно вирішити такі завдання: проаналізувати предметну область і наявні програмні аналоги систем оренди житла; визначити функціональні та нефункціональні вимоги до вебплатформи; спроектувати структуру бази даних, серверної частини та основні алгоритми роботи системи; реалізувати клієнтський інтерфейс, модуль бронювання, внутрішній чат, особисті кабінети користувачів та інструменти адміністрування; виконати ручне функціональне тестування основних сценаріїв роботи платформи.

Об'єктом дослідження є процес автоматизації оренди житлової нерухомості, що охоплює пошук об'єктів, подання заявки на бронювання, зміну її статусу та комунікацію між учасниками. Предметом дослідження є програмні засоби, архітектурні рішення та алгоритми, використані для створення клієнт-серверної вебплатформи оренди житла.

Інформаційну базу роботи склали офіційна документація використаних технологій, навчальні й науково-практичні джерела з проєктування програмного забезпечення, баз даних, вебзастосунків та тестування. У процесі розробки застосовано методи системного аналізу, реляційного моделювання, об'єктно-орієнтованого проєктування, компонентного підходу до побудови інтерфейсу та ручного функціонального тестування.

Практичне значення результатів полягає у створенні працездатного прототипу платформи оренди житла. Розроблена система може бути використана як основа для подальшого розвитку сервісу, адаптації під конкретного замовника або демонстрації архітектурних і функціональних рішень у сфері автоматизації оренди нерухомості. Серверна частина та база даних працюють у хмарному середовищі Microsoft Azure, а клієнтська частина під час тестування запускала локально та взаємодіяла з віддаленим API.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ У СФЕРІ ОРЕНДИ ЖИТЛА

1.1. Постановка задачі автоматизації та аналіз процесів взаємодії суб'єктів ринку нерухомості

Оренда житла пов'язана з постійним обміном інформацією між орендарями, орендодавцями та адміністраторами сервісу. У межах цього процесу потрібно зберігати дані про об'єкти нерухомості, їх розташування, вартість, доступність, заявки на бронювання та повідомлення між користувачами. Якщо ці дані обробляються вручну або в різних сервісах, зростає ймовірність помилок, втрати актуальної інформації та дублювання заявок на ті самі дати. Тому для цієї предметної області доцільним є створення єдиної вебсистеми, яка централізовано обробляє основні дії користувачів і спирається на узгоджену структуру даних [1, 2].

Під час проєктування RentPlatform було виділено три основні ролі користувачів. Орендар використовує систему для пошуку житла, перегляду деталей оголошення, додавання об'єктів в обране, створення бронювання та спілкування з орендодавцем. Для зручного пошуку передбачено фільтрацію за містом, типом житла, ціною, кількістю кімнат і наявністю генератора. Після вибору відповідного варіанта орендар може створити заявку на бронювання та відстежувати її стан у власному кабінеті.

Орендодавець відповідає за наповнення системи оголошеннями та обробку заявок. Він може створювати нові об'єкти нерухомості, редагувати їх опис, додавати зображення, переглядати вхідні бронювання та змінювати їхній статус. Також для орендодавця передбачено перегляд балансу, який оновлюється після завершення підтвердженого бронювання. Внутрішній чат дає змогу уточнювати деталі з орендарем без переходу до сторонніх месенджерів.

Адміністратор виконує допоміжну роль контролю якості даних у системі. Він може переглядати користувачів, блокувати облікові записи, а також модерувати оголошення, якщо їхній зміст не відповідає вимогам платформи. Такий розподіл ролей дає змогу відокремити дії звичайних користувачів від

адміністративних операцій. Узагальнену модель взаємодії користувачів із системою подано на (рис. 1.1).



Рис. 1.1. Діаграма прецедентів використання вебплатформи

Алгоритмічна складність автоматизації життєвого циклу оренди житла концентрується навколо розробки надійного модуля бронювання. Математичне та програмне моделювання цього процесу вимагає впровадження концепції кінцевого автомата (state-machine), який унеможливорює перебування замовлення в невизначеному стані або здійснення невалідних переходів між статусами. Кожне нове замовлення ініціалізується зі статусом очікування підтвердження. Програмна логіка прикладного шару в цей момент виконує серверну перевірку доступності об'єкта з використанням запиту до бази даних, аналізуючи перетин часових інтервалів [3]. Якщо обрані дати початку та закінчення оренди накладаються хоча б на одне вже підтверджене бронювання для цього конкретного об'єкта нерухомості, операція створення переривається з

поверненням відповідного повідомлення про помилку. Після успішного проходження валідації орендодавець отримує можливість перевести замовлення у статус підтвердженого, що робить цей календарний діапазон недоступним для повторного бронювання. Подальша еволюція стану передбачає або успішне завершення оренди з розрахунком підсумкової вартості на основі збереженого тарифу та тривалості перебування і додаванням відповідної суми до балансу власника, або скасування замовлення за ініціативою однієї зі сторін, що повертає часовий інтервал у категорію доступних для пошуку.

Окремою частиною системи є модуль повідомлень. У звичайних вебзастосунках для перевірки нових повідомлень інколи використовують періодичні HTTP-запити до сервера, однак такий підхід створює зайве навантаження і не дає повного відчуття роботи в реальному часі. У RentPlatform для цього застосовано SignalR, який підтримує WebSocket-з'єднання між клієнтом і сервером. Завдяки цьому повідомлення надходять без перезавантаження сторінки, а користувач бачить оновлення діалогу одразу після їх появи.

Для автентифікації користувачів у системі використано JWT-токени. Після успішного входу сервер формує токен, у якому зберігаються ідентифікатор користувача та його роль. Клієнтська частина додає цей токен до подальших HTTP-запитів, а сервер перевіряє права доступу до захищених маршрутів. Паролі користувачів зберігаються не у відкритому вигляді, а як хеші, сформовані за допомогою BCrypt. У межах MVP-версії refresh-токени не реалізовано, тому цей механізм можна розглядати як напрям подальшого розвитку підсистеми безпеки.

Отже, система повинна підтримувати роботу з користувачами різних ролей, каталог оголошень, фільтрацію, обране, бронювання, чат і адміністративну модерацію. Серверна частина має відокремлювати бізнес-логіку від доступу до даних, а клієнтська частина повинна забезпечувати зрозумілий інтерфейс для виконання основних сценаріїв [4, 5]. Ці вимоги стали основою для подальшого вибору архітектури, технологій та структури бази даних.

1.2. Порівняльний аналіз існуючих програмних аналогів систем оренди житла

Перед розробкою власної платформи було розглянуто кілька наявних сервісів оренди або пошуку житла. Метою аналізу було не повне копіювання їх функціоналу, а визначення сильних і слабких сторін, які варто врахувати під час створення RentPlatform. Основну увагу приділено способам пошуку житла, роботі з оголошеннями, бронюванню, комунікації між сторонами та можливості адаптації системи під локальні потреби.

Airbnb є одним із найвідоміших міжнародних сервісів для короткострокової оренди житла. Платформа має зручний каталог, фільтри, карту, сторінки об'єктів і власні механізми бронювання. Для користувача це цілісний сервіс, однак його архітектура та внутрішні процеси є закритими. Тому такий продукт не можна розгорнути як окрему локальну систему або адаптувати під конкретного невеликого замовника. Загальний вигляд головної сторінки Airbnb наведено на (рис. 1.2).

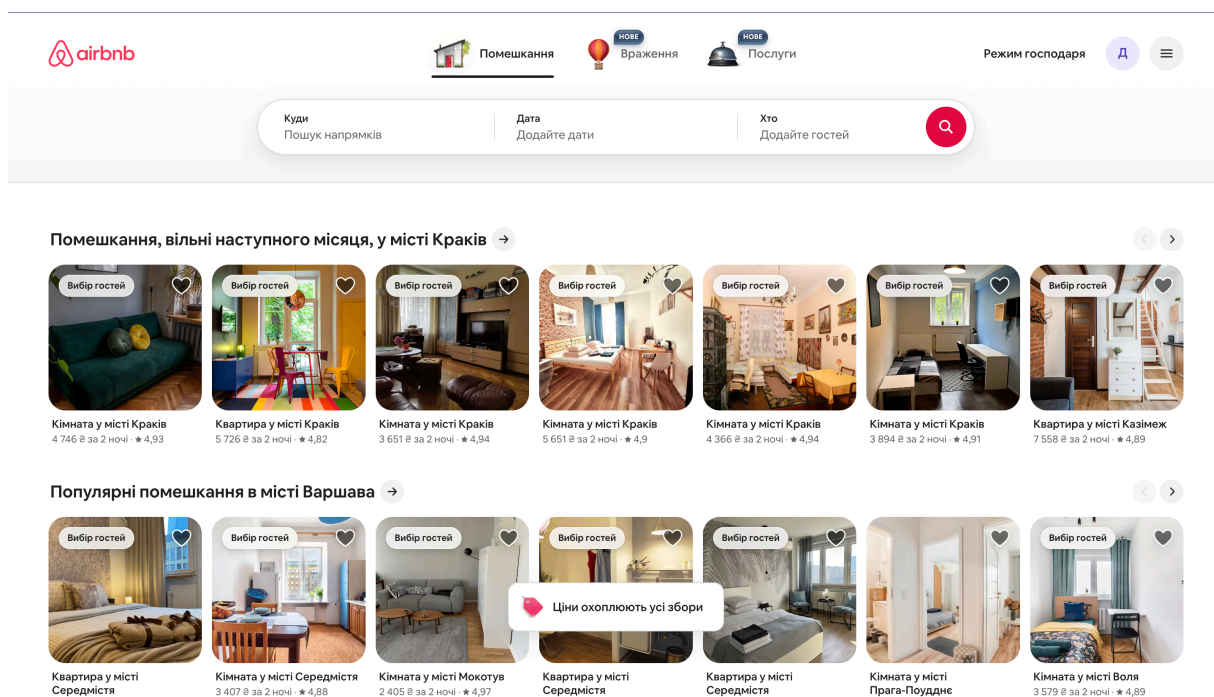


Рис. 1.2. Головна сторінка сайту Airbnb

Для цієї роботи Airbnb розглянуто як приклад зрілого сервісу з добре продуманим користувацьким інтерфейсом і повним циклом бронювання. Водночас його модель не відповідає задачі створення власного керованого

прототипу, оскільки користувачі працюють лише з готовою SaaS-платформою, а розробник не має доступу до її внутрішньої логіки або можливості змінювати правила роботи системи під локальні вимоги [6].

Booking.com також належить до великих міжнародних платформ, проте історично він більше орієнтований на готельний бізнес і короткострокове розміщення. Система підтримує пошук, фільтрацію, бронювання, відгуки та роботу з великою кількістю постачальників житла. Приклад сторінки об'єкта на Booking.com показано на (рис. 1.3).

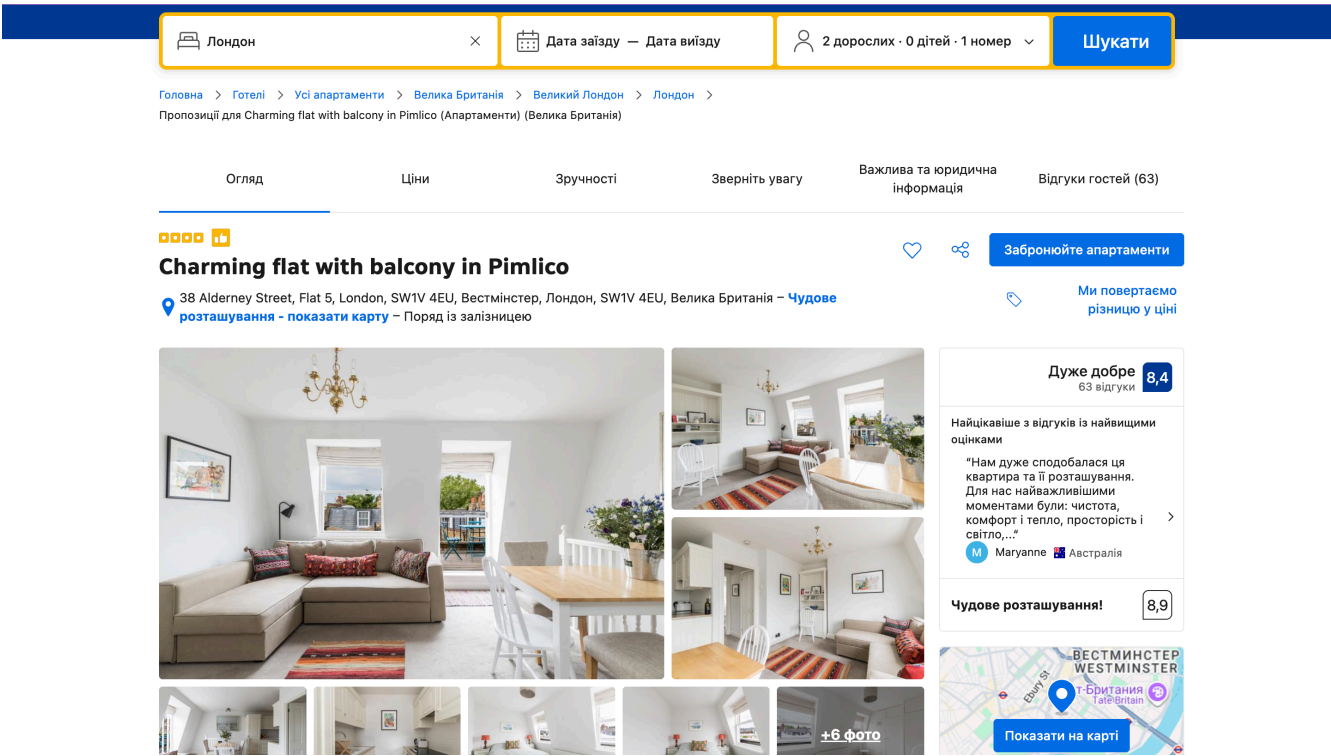


Рис. 1.3. Сторінка картки квартири на Booking.com

Booking.com має розвинену інфраструктуру бронювання, однак для задачі цієї роботи його модель є надто масштабною та закритою. Комунікація з користувачем часто доповнюється електронними листами й системними повідомленнями, а внутрішня логіка платформи не може бути використана як основа для власного розгортання. Тому цей сервіс доцільно розглядати як аналог за функціональним призначенням, а не як прямий технічний зразок для реалізації RentPlatform [7].

Серед українських сервісів важливим прикладом є DIM.RIA. Він працює як агрегатор оголошень про нерухомість і надає користувачам зручний пошук,

фільтри, картки об'єктів і контактну інформацію. Приклад картки квартири на порталі DIM.RIA наведено на (рис 1.4).

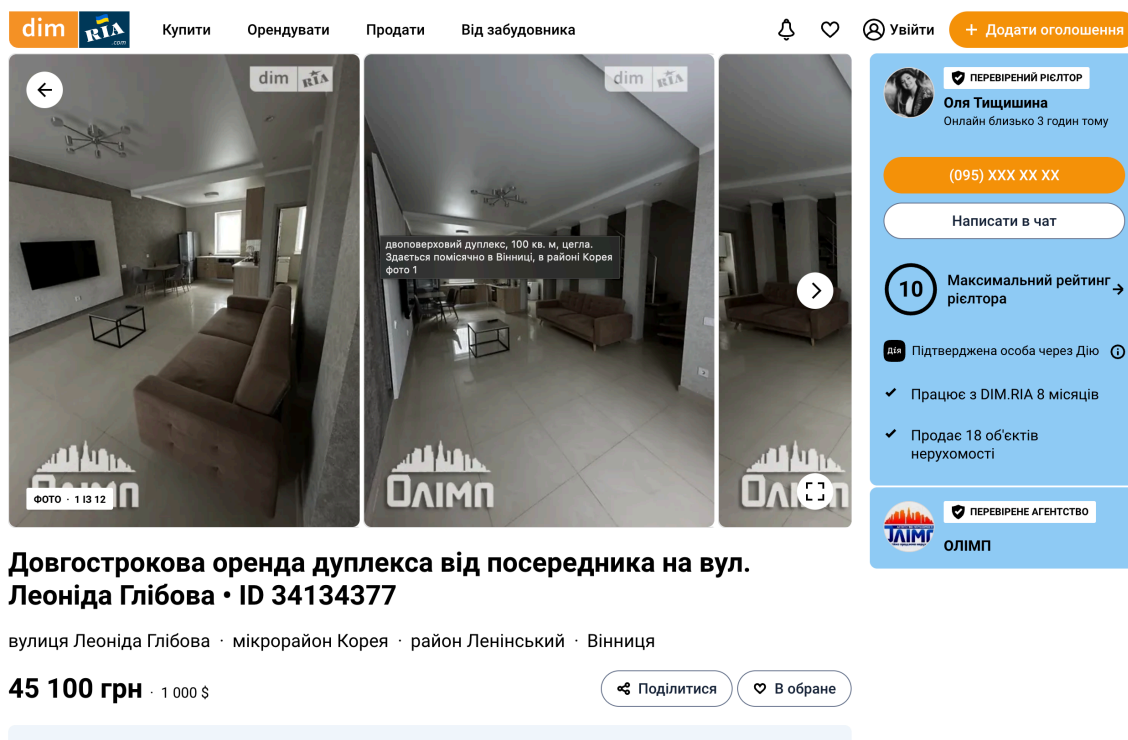


Рис. 1.4. Картка квартири на українському порталі DIM.RIA

DIM.RIA добре вирішує задачу публікації та пошуку оголошень, однак не є повноцінною платформою керування бронюваннями. У межах такого сервісу користувач зазвичай переходить до дзвінка або зовнішнього месенджера, а сама угода не проходить через внутрішній життєвий цикл заявки. Через це в системі не фіксуються всі етапи домовленості між орендарем і орендодавцем, а перевірка доступності об'єкта на конкретні дати залишається поза межами платформи [8].

Для порівняння аналогів було обрано критерії, які безпосередньо пов'язані з темою роботи: архітектурна модель, контроль перетину дат, наявність внутрішнього чату, облік балансу орендодавця та спосіб розгортання. Результати порівняння наведено в таблиці 1.1.

Таблиця 1.1

Порівняльна характеристика програмних аналогів систем оренди житла

Критерії порівняння	Платформа Airbnb	Система Booking.com	Агрегатор DIM.RIA	Проект RentPlatform
Архітектурна концепція	Закрита сервісна платформа	Закрита комерційна платформа	Напівзакрита сервісна	Відкрита Clean Architecture
Контроль перетину дат	Автоматичний хмарний	Автоматизоване керування доступністю	Відсутній (ручний)	Серверна перевірка перетину дат (LINQ HasOverlapAsync)
Технологія внутрішнього чату	Пропрієтарний асинхронний	Вбудований модуль повідомлень	Комунікація переважно поза процесом угоди	Дуплексний SignalR (WebSocket)
Облік внутрішнього балансу	Інтегрований глобальний	Сторонній білінг	Не є основним сценарієм у межах публічного каталогу	Відображення/поповнення балансу орендодавця
Модель розгортання	Виключно SaaS	Виключно SaaS	Корпоративна закрита	Хмарна (Azure App Service)

Порівняння показало, що наявні сервіси мають сильні сторони, але не відповідають задачі створення окремого навчального прототипу з відкритою клієнт-серверною архітектурою. У RentPlatform основний акцент зроблено на власному REST API, внутрішньому чаті, перевірці перетину дат під час бронювання, модерації оголошень та простому механізмі відображення балансу орендодавця. Клієнтську частину розроблено як SPA-застосунок із використанням Vite, що відповідає обраному підходу до побудови сучасного фронтенду [9].

1.3. Обґрунтування вибору архітектури та структури розподіленої вебплатформи

Для RentPlatform обрано клієнт-серверну архітектуру, у якій серверна частина відповідає за бізнес-логіку, доступ до бази даних і перевірку прав користувачів, а клієнтська частина – за відображення інтерфейсу та взаємодію з користувачем. Такий поділ спрощує супровід системи, оскільки зміни в

інтерфейси не потребують зміни структури бази даних, а серверний API може залишатися єдиною точкою доступу до основної логіки платформи [10]. Дані між клієнтом і сервером передаються у форматі JSON, що є зручним для взаємодії React-застосунку з REST API.

Серверну частину побудовано за принципами Clean Architecture. Для цього код розділено на окремі проєкти, кожен з яких має власну відповідальність. Такий підхід зменшує залежність бізнес-правил від конкретної бази даних або вебфреймворку. У центрі архітектури знаходиться доменна модель і прикладні сервіси, а зовнішні шари відповідають за інфраструктуру, контролери та підключення залежностей [11].

Структура серверного рішення складається з чотирьох основних проєктів. RentPlatform.Domain містить доменні сутності та переліки. RentPlatform.Application містить інтерфейси, DTO та сервіси, у яких реалізовано основні сценарії роботи системи. RentPlatform.Infrastructure відповідає за роботу з Entity Framework Core, PostgreSQL і реалізації репозиторіїв. RentPlatform.WebAPI містить контролери, конфігурацію застосунку, middleware та налаштування залежностей. Напрямок залежностей між цими проєктами наведено на (рис. 1.5).

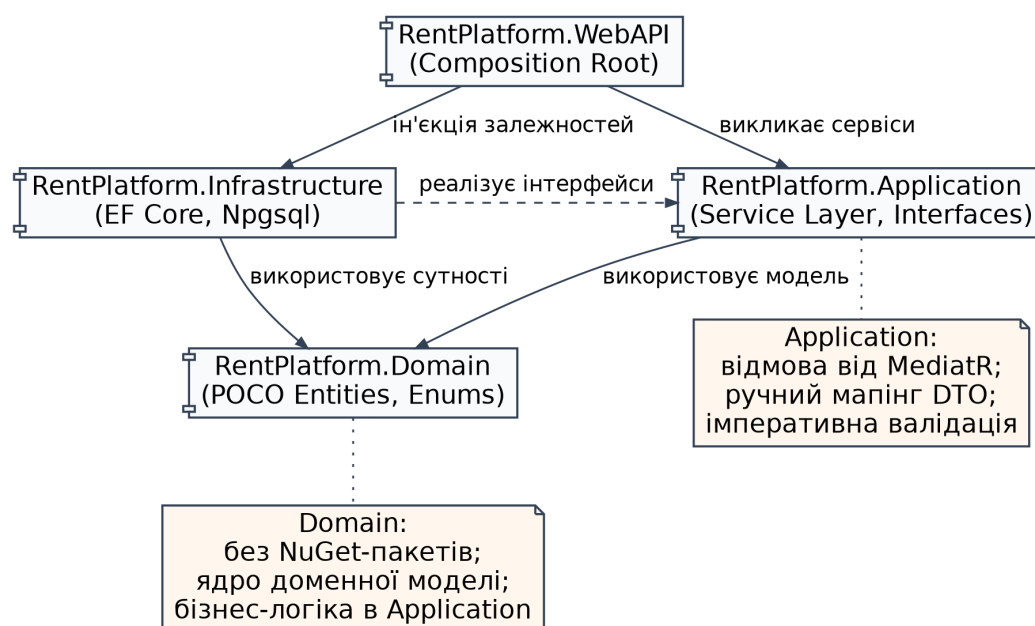


Рис. 1.5. Схема напрямку залежностей шарів Clean Architecture

У прикладному шарі не використовувалися MediatR, AutoMapper або окремий CQRS-підхід. Для масштабу цієї роботи достатньо сервісного шару, у якому методи безпосередньо описують основні дії користувачів. Перетворення сутностей у DTO виконується вручну у відповідних сервісах. Такий підхід робить код простішим для відстеження під час налагодження і не потребує додаткової конфігурації сторонніх бібліотек. Валідація частини вхідних даних виконується за допомогою DataAnnotations, а бізнес-правила перевіряються в прикладних сервісах [12].

Зовнішній шар архітектури представлений проєктами Infrastructure та WebAPI. У Infrastructure розміщено контекст бази даних, конфігурацію сутностей і репозиторії. WebAPI виконує роль точки запуску системи: у ньому налаштовуються контролери, JWT-автентифікація, CORS-політика, Swagger, SignalR і контейнер залежностей. Саме в цьому шарі абстракції прикладного рівня зв'язуються з їхніми конкретними реалізаціями.

Для збереження даних використано PostgreSQL. Реляційна модель підходить для системи оренди житла, оскільки бронювання, користувачі, оголошення, обране та повідомлення мають чіткі зв'язки між собою. Крім того, для операцій зі зміною статусів бронювання та оновленням балансу важливо підтримувати узгодженість даних. PostgreSQL забезпечує роботу із зовнішніми ключами, індексами, транзакціями та типами даних, які використовуються у проєкті.

Клієнтську частину реалізовано як односторінковий застосунок на React 19 з використанням TypeScript і Vite. Для запитів до сервера застосовано Axios, а для чату – SignalR. На стороні клієнта використовується єдине SignalR-з'єднання в контексті UnreadContext, що дає змогу отримувати нові повідомлення та оновлювати лічильник непрочитаних діалогів без створення окремого підключення для кожної сторінки чату [13].

Описана архітектура також спрощує подальший розвиток системи. Наприклад, у майбутньому можна додати окремий механізм refresh-токенів, централізовану обробку помилок або автоматизовані тести прикладного шару,

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						17

не змінюючи загальний принцип взаємодії між рівнями. Так само можна замінити частину інтерфейсу або додати нові сторінки на фронтенді без зміни доменних сутностей. Саме тому для RentPlatform було обрано не найскладнішу, а практично достатню архітектуру, яка відповідає реальному обсягу реалізованого програмного продукту.

Клієнтська частина винесена в окремий застосунок, що взаємодіє із сервером через REST API і SignalR. Це рішення дає змогу запускати фронтенд локально під час розробки, а серверну частину розміщувати окремо в хмарному середовищі. У такій схемі React-застосунок не містить бізнес-логіки бронювання або модерації, а лише передає запити та відображає отримані дані. Валідація, перевірка прав доступу і зміна станів бронювань залишаються на сервері, що зменшує ризик обходу правил через зміну клієнтського коду.

Під час розробки не ставилося завдання створити надмірно складну мікросервісну систему. Для бакалаврської роботи і для поточного функціоналу RentPlatform достатньою є модульна клієнт-серверна структура з чітким поділом відповідальності. Мікросервісний підхід потребував би окремого налаштування взаємодії між сервісами, додаткового моніторингу, централізованого логування та механізмів розподілених транзакцій. У межах цієї реалізації це лише ускладнило б підтримку проєкту і не дало б практичної переваги для демонстраційної версії платформи.

Окрему роль у цій архітектурі відіграють інтерфейси репозиторіїв і сервісів. Вони описують, які дії може виконувати прикладний шар, але не прив'язують його до конкретного способу збереження даних. Завдяки цьому сервіс бронювання працює з абстракцією репозиторію, а не з конкретним класом Entity Framework Core. Такий підхід спрощує читання коду і робить залежності зрозумілішими: контролер приймає HTTP-запит, сервіс виконує бізнес-перевірки, репозиторій звертається до бази даних, а результат повертається клієнту у вигляді DTO.

У межах роботи архітектура розглядалася не лише як спосіб розділення коду на декілька проєктів, а як засіб зменшення залежностей між окремими

частинами системи. Для платформи оренди житла це важливо тому, що різні функції мають різну природу. Наприклад, автентифікація користувачів, публікація оголошень, створення бронювань і надсилання повідомлень належать до різних бізнес-сценаріїв, але всі вони використовують спільні сутності та спільний доступ до бази даних. Обрана структура дозволяє змінювати реалізацію одного модуля без повного переписування інших частин застосунку.

1.4. Обґрунтування вибору інструментальних засобів розробки та вимоги до апаратного забезпечення

Вибір технологій для RentPlatform пов'язаний із тим, що система складається з кількох взаємопов'язаних частин: серверного API, бази даних, клієнтського застосунку та модуля повідомлень у реальному часі. Тому технологічний стек мав забезпечувати зручну розробку, зрозумілу структуру коду, стабільну роботу з даними та можливість подальшого розширення функціоналу.

Серверну частину реалізовано на платформі .NET 10.0 мовою C#. Ця екосистема підходить для створення веб API та бізнес-застосунків, оскільки має розвинені засоби типізації, асинхронного програмування, роботи з колекціями, обробки винятків і впровадження залежностей. Для цього проєкту важливо, що C# дає змогу писати структурований код із чітким поділом на контролери, сервіси, репозиторії та доменні моделі.

Платформа .NET також має засоби, які корисні для серверних застосунків із багатьма одночасними запитами. Асинхронні методи дають можливість не блокувати виконання під час роботи з базою даних або мережею, а вбудоване керування пам'яттю спрощує розробку. У межах RentPlatform це використано для обробки HTTP-запитів, роботи з Entity Framework Core та підтримки SignalR-з'єднань.

Для побудови REST API використано ASP.NET Core. Фреймворк надає маршрутизацію, контролери, серіалізацію JSON, налаштування авторизації, CORS-політик і Swagger-документації. Вбудований контейнер залежностей дає

змогу передавати в контролери потрібні сервіси через конструктори, що відповідає обраній структурі Clean Architecture [14].

Для обміну повідомленнями в режимі реального часу використано SignalR. Ця технологія спрощує роботу з WebSocket-з'єднаннями та підтримує автоматичне перепідключення у разі короткочасної втрати зв'язку. У проєкті SignalR застосовано для внутрішнього чату між орендарем і орендодавцем, щоб нові повідомлення з'являлися без ручного оновлення сторінки.

Для збереження даних обрано PostgreSQL. Ця СКБД підтримує зовнішні ключі, індекси, транзакції та стандарт SQL, що важливо для системи з пов'язаними сутностями. У RentPlatform PostgreSQL використовується для збереження користувачів, оголошень, бронювань, обраного та повідомлень. Робота з конкурентним доступом і транзакціями допомагає підтримувати узгодженість даних під час одночасної роботи кількох користувачів.

У проєкті також використано можливості PostgreSQL для роботи з масивами даних. Зокрема, список URL-адрес зображень об'єкта нерухомості зберігається як масив текстових значень. Такий підхід не потребує створення окремої таблиці для зображень у межах поточної MVP-версії і водночас дозволяє зберігати кілька посилань для одного оголошення [15].

Зв'язок між C#-кодом і базою даних реалізовано за допомогою Entity Framework Core 10. ORM-підхід дозволяє працювати з таблицями через класи, властивості та колекції, а не писати всі SQL-запити вручну. У проєкті використано підхід Code-First: структура бази формується на основі доменних класів і конфігурації Fluent API, а зміни застосовуються через міграції.

Entity Framework Core також автоматично параметризує запити, що зменшує ризик SQL Injection. Крім того, він підтримує асинхронні операції, відстеження змін сутностей, налаштування зв'язків і виконання запитів через LINQ. Ці можливості використовуються в репозиторіях, які відокремлюють прикладні сервіси від деталей роботи з базою даних.

Клієнтську частину створено на React 19. Бібліотека дає змогу будувати інтерфейс із багаторазових компонентів, що зручно для сторінок каталогу,

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						20

карток житла, форм, особистих кабінетів, модальних вікон і чату. Компонентний підхід спрощує підтримку інтерфейсу, оскільки окремі елементи можна змінювати незалежно від решти сторінки.

React оновлює тільки ті частини інтерфейсу, стан яких змінився. Це важливо для сторінок із фільтрами, списками оголошень, картою та повідомленнями, де користувач постійно взаємодіє з елементами сторінки. У поєднанні з Vite це забезпечує зручний процес розробки та достатню швидкість роботи клієнтської частини.

Для клієнтського коду використано TypeScript. Статична типізація допомагає раніше виявляти помилки, пов'язані з неправильними структурами даних або параметрами функцій. Це особливо корисно під час роботи з DTO, які передаються між клієнтом і сервером через REST API [16].

Vite використано як інструмент збирання фронтенду. Він швидко запускає локальний сервер розробки та підтримує гарячу заміну модулів, тому зміни в інтерфейсі одразу відображаються в браузері. Це пришвидшує перевірку сторінок, форм і компонентів під час розробки [9].

Для стилізації інтерфейсу використано Tailwind CSS. Його утилітарні класи дозволяють описувати зовнішній вигляд елементів без великих окремих CSS-файлів. У RentPlatform цей підхід застосовано для побудови адаптивних карток, форм, кнопок, панелей фільтрації та особистих кабінетів.

Серверна частина та база даних розгорнуті в хмарному середовищі Microsoft Azure. Такий підхід дає змогу запускати API незалежно від локального комп'ютера розробника, а параметри підключення до бази даних зберігати через змінні оточення. Клієнтська частина під час тестування запускалася локально через Vite та зверталася до віддаленого API.

Для нормальної роботи системи потрібні базові апаратні та програмні умови для серверної й клієнтської частин. Сервер повинен мати достатні ресурси для обробки HTTP-запитів, роботи з PostgreSQL і підтримки SignalR-з'єднань. Користувачеві потрібен сучасний браузер із підтримкою HTML5, JavaScript та виконанням клієнтських сценаріїв. Узагальнені вимоги наведено в таблиці 1.2.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						21

Вимоги до апаратного та програмного забезпечення системи

Компонент системи	Мінімальні апаратні та програмні вимоги	Рекомендовані апаратні та програмні вимоги
Серверний вузол (Бекенд + СКБД PostgreSQL)	Процесор: 2 ядра (2.0 ГГц) Оперативна пам'ять: 4 ГБ Дисковий простір: 20 ГБ SSD ОС: Linux (Ubuntu 22.04) / Windows Server	Процесор: 4 ядра (3.0 ГГц і вище) Оперативна пам'ять: 8 ГБ Дисковий простір: 50 ГБ NVMe SSD Хмарне середовище: Azure App Service (Tier Standard)
Клієнтська робоча станція (Користувач)	Процесор: 2 ядра (1.6 ГГц) Оперативна пам'ять: 2 ГБ Екран: роздільна здатність 1024x768 Програмне забезпечення: будь-який браузер з підтримкою HTML5 та ES6	Процесор: 4 ядра (2.4 ГГц) Оперативна пам'ять: 4 ГБ Екран: роздільна здатність 1920x1080 Програмне забезпечення: Google Chrome 110+, Safari 15+

Мінімальні вимоги наведені для локального або самостійного розгортання системи. У демонстраційній конфігурації серверна частина працює в Azure App Service, база даних PostgreSQL розміщена як окремий віддалений ресурс, а клієнтський застосунок запускається локально через Vite.

Наведені вимоги забезпечують умови для запуску серверної частини, бази даних і клієнтського інтерфейсу. Обраний технологічний стек дозволяє розвивати систему надалі, додавати нові модулі та змінювати окремі частини застосунку без повного перепроектування всієї платформи.

Висновки до першого розділу

У першому розділі кваліфікаційної роботи проаналізовано предметну область і визначено, які процеси оренди житла доцільно автоматизувати. Було виділено три ролі користувачів: орендар, орендодавець та адміністратор. Для кожної ролі описано основні дії, а їхню взаємодію з платформою подано на діаграмі прецедентів. Окремо визначено, що для системи важливими є перевірка перетину дат під час бронювання та внутрішній чат між користувачами.

Порівняльний аналіз Airbnb, Booking.com і DIM.RIA показав, що ці сервіси мають розвинений функціонал, однак не відповідають задачам цієї роботи. Для RentPlatform важливими є не глобальні можливості великої комерційної платформи, а реалізація власного керованого прототипу з оголошеннями,

бронюваннями, чатом, обраним, модерацією та відображенням балансу орендодавця.

Для реалізації системи обрано клієнт-серверну архітектуру. Серверна частина побудована на ASP.NET Core за принципами Clean Architecture, база даних – на PostgreSQL, а клієнтський інтерфейс – на React, TypeScript, Vite і Tailwind CSS. Для повідомлень у реальному часі використано SignalR. Сформовані вимоги та вибрані технології стали основою для подальшого проєктування і реалізації системи.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБПЛАТФОРМИ ОРЕНДИ ЖИТЛА

2.1. Проєктування логічної та фізичної структури бази даних

Робота вебплатформи RentPlatform залежить від коректно спроектованого шару збереження даних. Для цього використано PostgreSQL, який підключається до серверної частини через провайдер Npgsql. Реляційна база даних обрана тому, що основні сутності системи мають чіткі зв'язки між собою, а операції бронювання й оновлення статусів потребують узгодженості даних.

Під час моделювання бази даних було виділено п'ять основних сутностей: користувачі, об'єкти нерухомості, бронювання, обране та повідомлення. Для первинних ключів використано UUID. Це дозволяє створювати унікальні ідентифікатори без прив'язки до послідовної нумерації записів і зручно працювати з ними на рівні прикладного коду.

Таблиця Users зберігає дані, необхідні для автентифікації та розмежування прав доступу. До неї входять ідентифікатор, електронна пошта, хеш пароля, ім'я, прізвище, роль, ознака блокування та баланс. Для балансу використано numeric(18,2), щоб зберігати грошові значення з фіксованою точністю. У поточній версії автентифікація працює через JWT-токен із терміном дії 7 діб, а refresh-токени не реалізовані.

Таблиця Properties описує об'єкти нерухомості. Вона містить назву, опис, адресу, місто, тип житла, ціну за добу, площу, кількість кімнат, поверх, координати, дату створення, URL-адреси зображень, ідентифікатор орендодавця та поля модерації. Окремо зберігається ознака HasGenerator, оскільки наявність альтернативного живлення є важливим критерієм для частини користувачів.

Для повідомлень використано таблицю Messages, у якій зберігаються відправник, отримувач, текст, дата надсилання та ознака прочитання. Таблиця Favorites пов'язує користувача з оголошенням, яке він додав до обраного. Для пари UserId і PropertyId налаштовано унікальне обмеження, щоб один користувач не міг додати одне й те саме оголошення кілька разів. Зв'язки між таблицями відображено на ER-діаграмі системи (рис. 2.1).

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						24

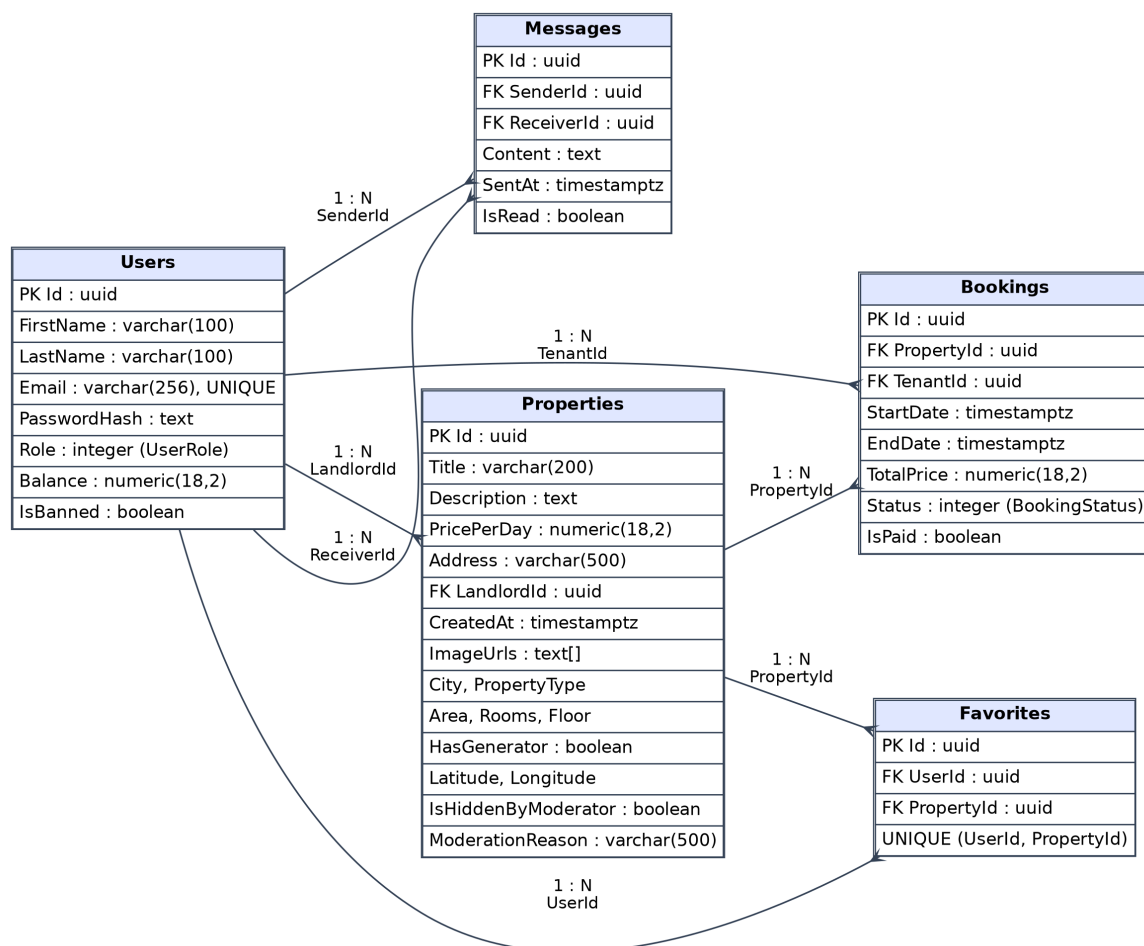


Рис. 2.1. ER-діаграма бази даних

Таблиця Bookings пов'язує об'єкт нерухомості з орендарем, який створив заявку. Вона містить зовнішні ключі на житло та користувача, дати початку й завершення оренди, підсумкову вартість, ознаку оплати та статус заявки. Статус використовується прикладною логікою для керування життєвим циклом бронювання. Основні поля сутності Bookings наведено в таблиці 2.1.

Таблиця 2.1

Фізична структура та обмеження таблиці замовлень Bookings

Назва стовпця	Тип даних у СКБД	Обмеження цілісності	Інженерне призначення поля
Id	uuid	PRIMARY KEY, NOT NULL	Глобальний унікальний ключ запису
PropertyId	uuid	FOREIGN KEY → Properties(Id), NOT NULL, ON DELETE RESTRICT	Зв'язок з об'єктом нерухомості
TenantId	uuid	FOREIGN KEY → Users(Id), NOT NULL, ON DELETE RESTRICT	Зв'язок з профілем орендаря

1.	2.	3.	4.
StartDate	timestamp with time zone	NOT NULL	Дата і час початку оренди
EndDate	timestamp with time zone	NOT NULL	Дата і час завершення оренди
TotalPrice	numeric(18,2)	NOT NULL	Підсумкова сума за весь термін
Status	integer	NOT NULL	Стан (0-Pending, 1-Confirmed, 2-Cancelled, 3-Completed)
IsPaid	boolean	NOT NULL	Прапорець фіксації оплати

Для полів `PropertyId` та `TenantId` у базі даних створено несунікальні індекси, що формуються на основі зовнішніх ключів і використовуються під час вибірки бронювань за об'єктом житла або орендарем.

Взаємодія серверного коду з базою даних реалізована через `Entity Framework Core 10` за підходом `Code-First`. Розробник працює з класами `C#`, а структура таблиць формується за допомогою міграцій. Доменні класи не містять інфраструктурних атрибутів на зразок `[Key]` або `[Table]`, що відповідає обраному поділу відповідальності між `Domain` та `Infrastructure`.

Конфігурація таблиць винесена в інфраструктурний шар і описана через `Fluent API` у методі `OnModelCreating` контексту `ApplicationDbContext`. Там задаються типи полів, значення за замовчуванням, зв'язки між сутностями та поведінка зовнішніх ключів. Наприклад, для `TotalPrice` встановлено тип `numeric(18,2)`, а для зв'язків бронювання з користувачем і об'єктом нерухомості використано `DeleteBehavior.Restrict`, щоб не втратити пов'язані записи про бронювання.

Для прискорення роботи з пов'язаними даними в міграціях створено індекси для зовнішніх ключів. У таблиці `Bookings` використовуються індекси за `PropertyId` і `TenantId`, у таблиці `Users` – унікальний індекс за `Email`, а в таблиці `Favorites` – унікальний складений індекс за `UserId` і `PropertyId`. Поле `HasGenerator` використовується у фільтрації каталогу, але окремий частковий індекс для нього в поточній версії проєкту не створено [17].

Отже, фізична структура бази даних відповідає основним процесам платформи: автентифікації, публікації оголошень, бронюванню, обміну повідомленнями та збереженню обраних об'єктів. Вона не містить зайвих таблиць для функцій, які не реалізовані в поточній версії, наприклад для онлайн-оплати або історії фінансових транзакцій. Це робить модель даних достатньо простою для супроводу і водночас повною для наявного функціоналу RentPlatform.

Зберігання URL-адрес зображень у полі ImageUrls пов'язане з тим, що файли зображень розміщуються окремо, а в базі даних достатньо зберігати шляхи до них. Це зменшує навантаження на базу даних і спрощує відображення фото в клієнтському застосунку. Такий підхід підходить для поточної версії системи, оскільки завданням було реалізувати робочий механізм завантаження та перегляду фото, а не окреме сховище медіафайлів із розширеними правилами доступу.

Для таблиці Bookings важливим є не тільки збереження дат і вартості, а й фіксація поточного статусу заявки. Статус зберігається як ціле число, що відповідає значенням переліку BookingStatus у серверному коді. Це дає змогу однаково трактувати стан бронювання в базі даних, прикладних сервісах і клієнтському інтерфейсі. Наприклад, заявка зі статусом Pending очікує рішення орендодавця, Confirmed означає підтвердження, Cancelled – скасування, а Completed – завершення періоду оренди.

Для збереження цілісності даних у схемі використовуються зовнішні ключі та обмеження. Унікальність електронної пошти в таблиці Users не дозволяє створити два облікові записи з однаковим логіном. Унікальне обмеження для пари UserId і PropertyId у таблиці Favorites запобігає дублюванню одного й того самого оголошення в обраному користувача. Такі обмеження важливі, оскільки частину помилок краще блокувати не лише на рівні інтерфейсу, а й на рівні бази даних.

Зв'язки між таблицями побудовано відповідно до ролей користувачів у системі. Один орендодавець може створити багато оголошень, тому між Users і

Properties використовується зв'язок один-до-багатьох. Один орендар може створити багато бронювань, а кожне бронювання стосується лише одного об'єкта нерухомості. Таблиця Messages має два зовнішні ключі на Users, оскільки кожне повідомлення має відправника й отримувача. Таблиця Favorites є проміжною, бо один користувач може додати багато оголошень до обраного, а одне оголошення може бути додане до обраного багатьма користувачами.

Під час проєктування бази даних окремо враховувалося, які дані повинні зберігатися як обов'язкові, а які можуть бути відсутніми. Наприклад, для об'єкта нерухомості обов'язковими є назва, опис, місто, ціна, адреса, координати та ідентифікатор орендодавця. Водночас поле поверху може бути необов'язковим, оскільки не всі типи житла або оголошень потребують цього параметра. Такий підхід дозволяє не перевантажувати користувача зайвими вимогами під час створення оголошення, але водночас зберігати достатній набір даних для пошуку і відображення житла в каталозі.

2.2. Проєктування та архітектурна структура серверної частини (REST API)

Серверна частина RentPlatform реалізована як REST API. Вона приймає HTTP-запити від клієнтського застосунку, перевіряє вхідні дані, виконує бізнес-логіку та звертається до шару збереження даних. API побудовано на ASP.NET Core 10.0 і розділено відповідно до обраної структури Clean Architecture.

Для обробки запитів у серверній частині використано контролери. Вони відповідають за автентифікацію, оголошення, бронювання, повідомлення, обране, профіль користувача та адміністрування. Окремий DevController має службове призначення і використовується під час розробки, тому не належить до основного бізнес-функціоналу. Контролери успадковуються від ControllerBase і позначені атрибутом [ApiController], який спрощує прив'язку моделей та повернення помилок 400 Bad Request у разі невалідних даних. Ключові маршрути REST API наведено в таблиці 2.2.

Таблиця 2.2

Специфікація ключових кінцевих точок REST API системи RentPlatform

Контролер	Метод	Маршрут (Route)	Вхідні дані (DTO / URL)	Коди відповіді	Функціональне призначення
AuthController	POST	/api/auth/register	RegisterRequest	201, 400, 409	Реєстрація нового користувача
AuthController	POST	/api/auth/login	LoginRequest	200, 400, 401, 403	Автентифікація та видача JWT-токена (403 - бан)
PropertyController	GET	/api/properties	Фільтри (City, HasGenerator та ін.)	200, 400, 401	Отримання каталогу з фільтрацією
PropertyController	POST	/api/properties	CreatePropertyRequest	201, 400, 401, 403	Публікація оголошення орендодавцем
BookingController	POST	/api/bookings	CreateBookingRequest	201, 400, 401, 404	Створення заявки на оренду житла
MessageController	GET	/api/messages/{partnerId}	Guid partnerId (URL)	200, 401	Вибірка історії чату з користувачем

Перевірка даних, які надходять від клієнта, виконується на двох рівнях. Спочатку DTO-моделі перевіряються за допомогою атрибутів DataAnnotations, наприклад [Required], [MaxLength] і [Range]. Потім у сервісах перевіряються бізнес-правила, які не можна описати простими атрибутами. До таких правил належать перевірка перетину дат бронювання, заборона бронювання власного житла та перевірка прав орендодавця на зміну статусу заявки [18].

Перетворення доменних сутностей у DTO виконується вручну в сервісах. У проєкті не використано AutoMapper або Mapster, оскільки для поточного обсягу системи ручний мапінг є зрозумілішим і простішим для контролю. Наприклад, сервіс може самостійно сформувати DTO оголошення, додавши до нього дані орендодавця або список зображень. Зворотне перетворення під час створення чи оновлення записів також виконується прямим присвоєнням полів.

Реєстрація залежностей зосереджена у файлі Program.cs, який виконує роль точки збирання застосунку. У ньому реєструються сервіси, репозиторії, контекст бази даних, параметри JWT-автентифікації, CORS-політика, Swagger і SignalR.

Для сервісів і репозиторіїв використано життєвий цикл *Scoped*, тобто об'єкт створюється в межах одного HTTP-запиту. Схему взаємодії основних компонентів серверної частини на прикладі роботи з оголошеннями подано у спрощеному вигляді, для збереження читабельності на ній показано лише ключові методи інтерфейсів, тоді як повний перелік операцій міститься у файлах *IPROPERTYService* та *IPROPERTYRepository* (рис. 2.2).

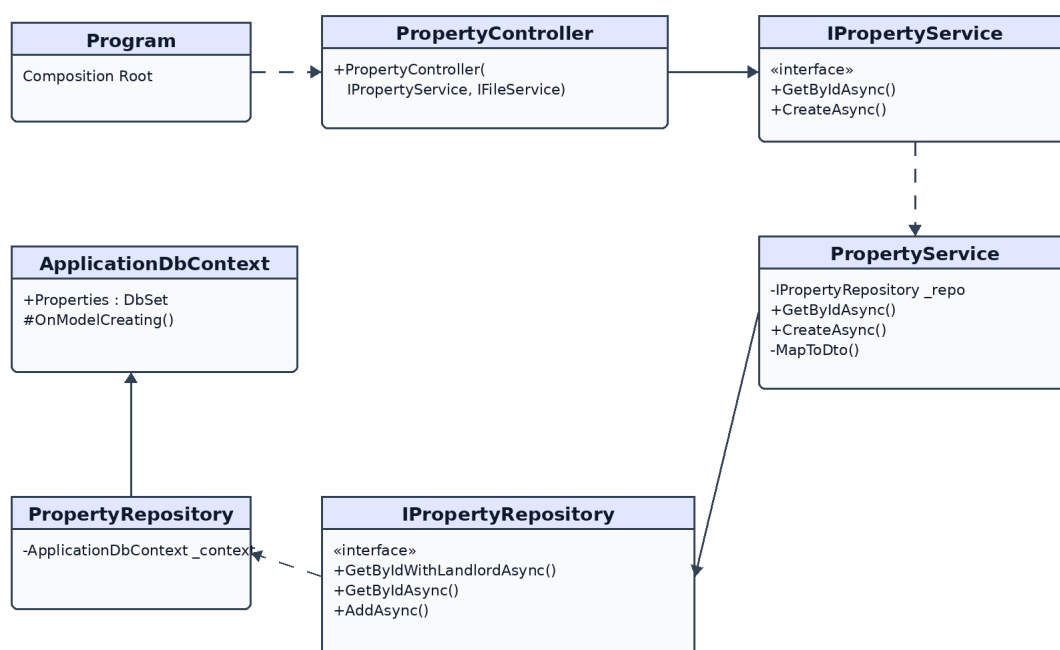


Рис. 2.2. Діаграма класів взаємодії компонентів серверної частини

У поточній версії платформи помилки обробляються локально. Прикладні сервіси генерують винятки, зокрема *KeyNotFoundException* або *InvalidOperationException*, а контролери перехоплюють їх і повертають відповідні HTTP-коди. Наприклад, відсутність запису може повертати 404 Not Found, а порушення бізнес-правила – 400 Bad Request або 409 Conflict. Централізований *middleware* для обробки винятків у цій версії не реалізовано, його можна додати під час подальшого розвитку системи.

Модуль повідомлень у реальному часі реалізовано за допомогою *SignalR*. Для цього в бекенді створено *ChatHub*, захищений атрибутом *[Authorize]*. Хаб доступний за маршрутом */chatHub* і має метод *SendMessage*, який приймає

ідентифікатор отримувача та текст повідомлення. Після збереження повідомлення через IMessageService сервер надсилає подію ReceiveMessage отримувачу та MessageSent відправнику. Для адресної доставки використовується прив'язка до ідентифікатора користувача через Clients.User(...) [19].

Загальну послідовність взаємодії компонентів серверної частини під час обробки типового HTTP-запиту (від отримання даних до формування DTO) наведено на UML-діаграмі послідовності (рис. 2.3).

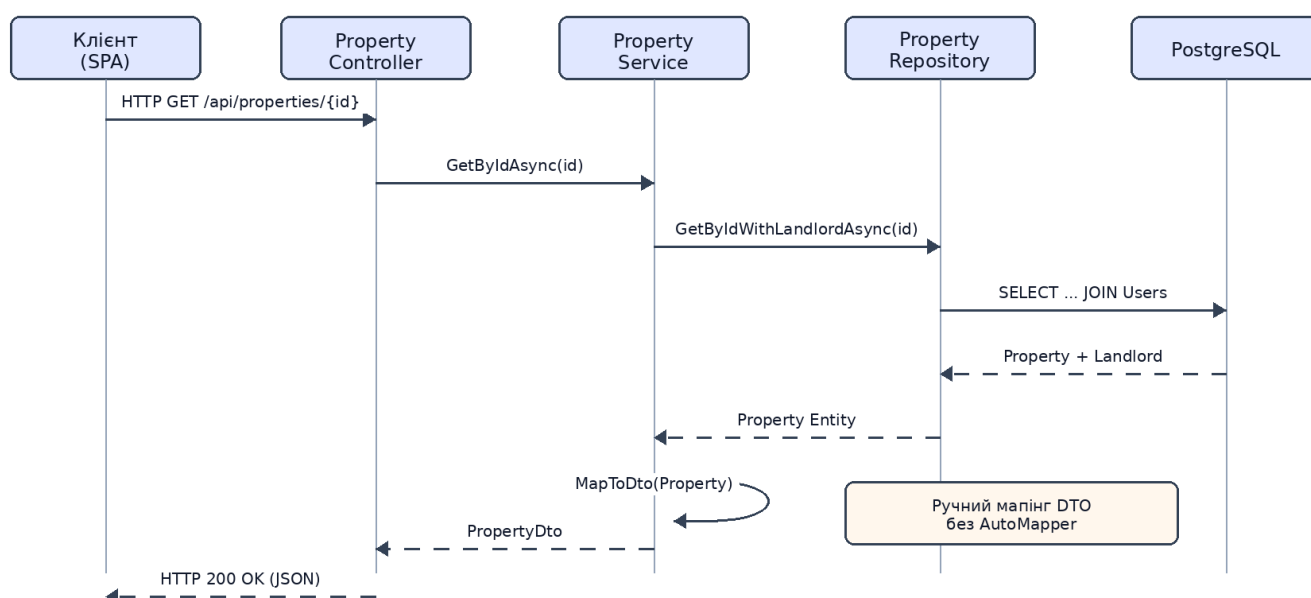


Рис. 2.3. Діаграма послідовності обробки HTTP-запиту в архітектурі

2.3. Проєктування алгоритмів роботи вебсистеми

Основні бізнес-процеси RentPlatform реалізовані в прикладному шарі. Найважливішими серед них є створення бронювання з перевіркою перетину дат, зміна статусів заявки та фільтрація каталогу оголошень. Саме ці алгоритми визначають, як система реагує на основні дії користувачів [20].

Створення бронювання виконується в методі CreateAsync сервісу BookingService. Спочатку перевіряються базові умови: дата початку не може бути в минулому, дата завершення має бути пізнішою за дату початку, а орендар не може бронювати власне житло. Якщо одна з умов порушена, сервіс генерує InvalidOperationException.

Після базової перевірки система завантажує об'єкт нерухомості та перевіряє, чи немає інших активних бронювань на обраний період. Для цього використовується метод `HasOverlapAsync`, у якому застосовано умову `b.StartDate < endDate && b.EndDate > startDate`. Скасовані заявки не враховуються, оскільки вони не займають календарний проміжок. Якщо перетин знайдено, створення заявки переривається. Якщо дати вільні, система розраховує вартість, створює заявку зі статусом `Pending` і зберігає її в базі даних. Логіку цього процесу подано на (рис. 2.4).

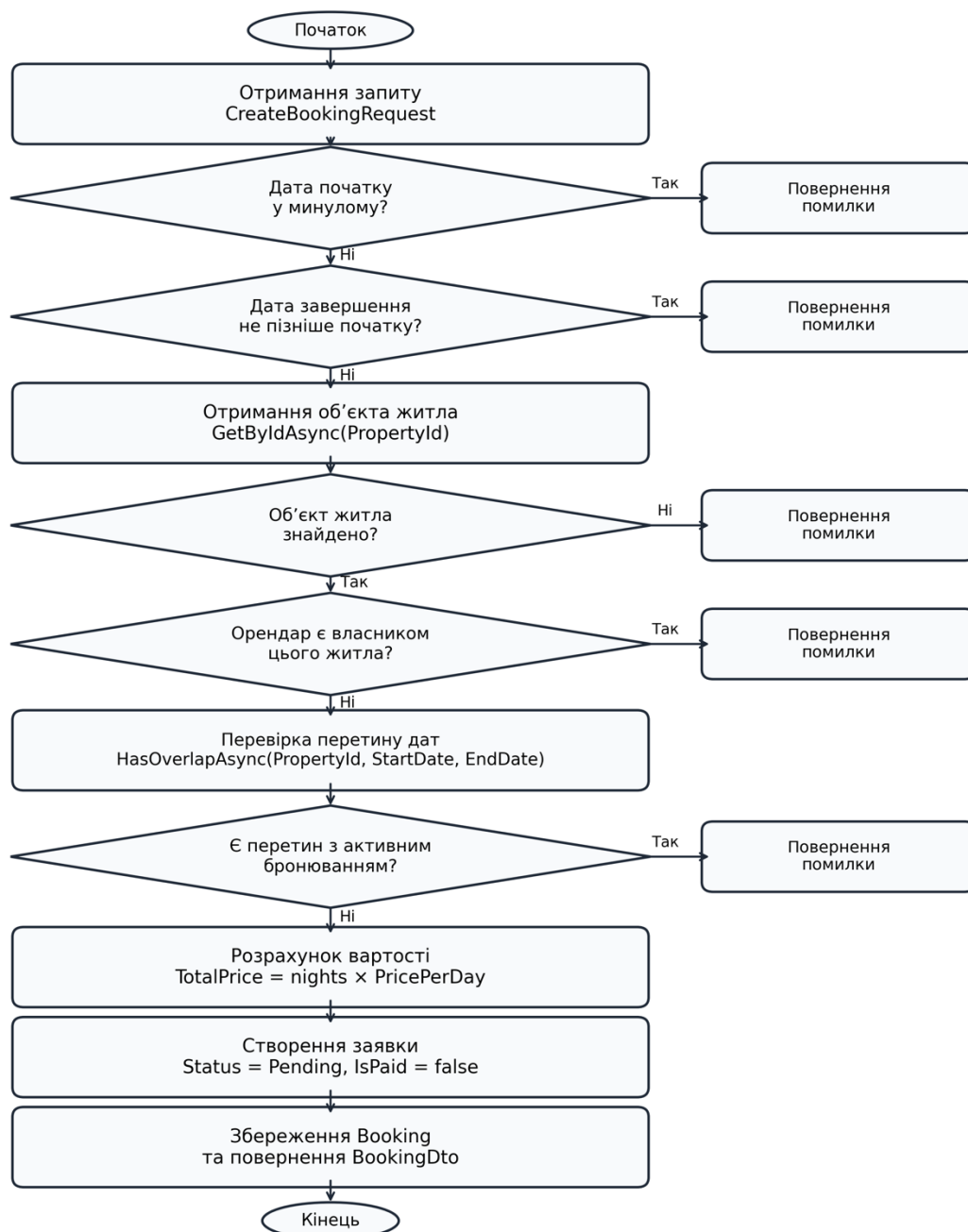


Рис. 2.4. Загальна схема алгоритму валідації та створення заявки на бронювання

Зміна статусів бронювання виконується в методі `UpdateStatusAsync`. Перед зміною система перевіряє, чи є користувач власником відповідного оголошення. Якщо статус намагається змінити інший користувач, сервіс генерує `UnauthorizedAccessException`. Це обмеження потрібне для того, щоб орендар або сторонній користувач не міг підтверджувати чи завершувати заявку замість орендодавця.

Дозволені переходи між статусами описані у функції `IsValidTransition` [21]. Зі стану `Pending` заявка може перейти в `Confirmed` або `Cancelled`. Зі стану `Confirmed` вона може перейти в `Completed` або `Cancelled`. Інші переходи вважаються невалідними. Коли бронювання переходить у статус `Completed`, сума `TotalPrice` додається до балансу орендодавця. Діаграму станів цього процесу наведено на (рис. 2.5).

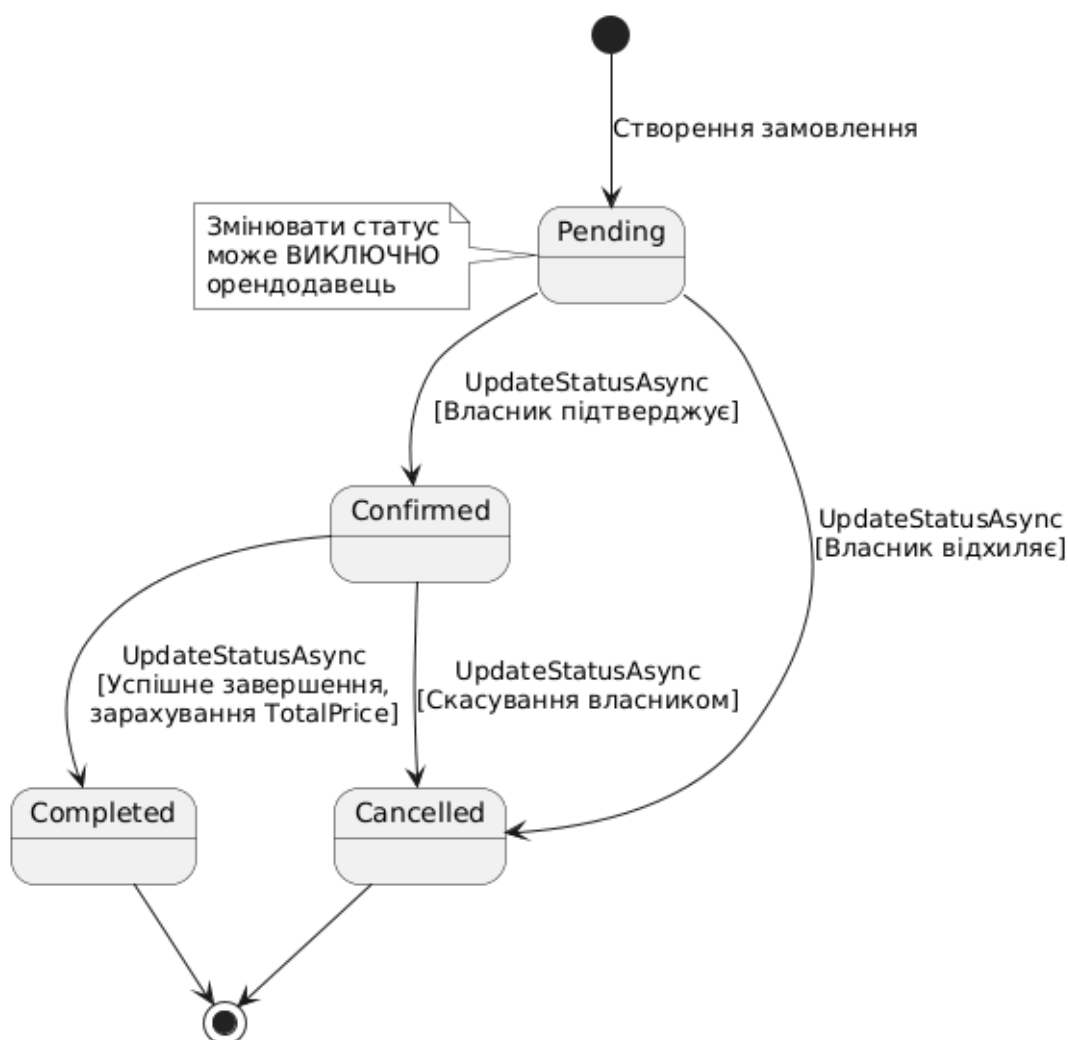


Рис. 2.5. UML-діаграма станів кінцевого автомата бронювання

Пошук оголошень реалізовано в методі GetAllWithLandlordAsync репозиторію. Запит формується поступово: до IQueryable додаються лише ті умови, які відповідають заповненим фільтрам у PropertyFilterParams [17, 22]. Першою умовою є відбір лише тих оголошень, які не приховані модератором.

Текстовий пошук за назвою виконується через EF.Functions.ILike, що дає змогу шукати без урахування регістру в PostgreSQL. Далі за потреби застосовуються фільтри за містом, типом нерухомості, мінімальною та максимальною ціною, кількістю кімнат і наявністю генератора. Кожен із цих фільтрів додається тільки тоді, коли користувач передав відповідний параметр.

Після застосування фільтрів система рахує загальну кількість знайдених оголошень, сортує результати та застосовує пагінацію через Skip і Take. За замовчуванням оголошення впорядковуються за датою створення, а розмір сторінки становить 9 об'єктів. На рівні сервісу результат формується у PagedResult<PropertyDto>, який містить самі оголошення та дані для навігації між сторінками. Алгоритм фільтрації й пагінації показано на (рис. 2.6).

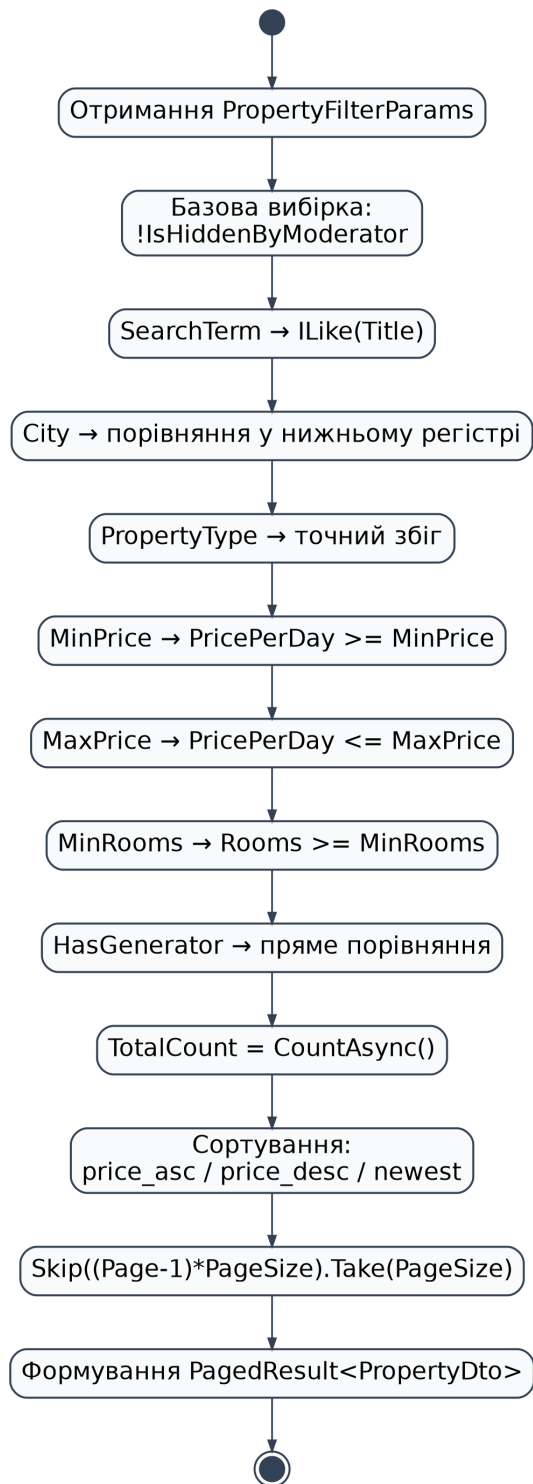


Рис. 2.6. Загальна схема алгоритму фільтрації та пагінації каталогу нерухомості

Описані алгоритми забезпечують основну логіку роботи платформи: створення заявки, контроль її статусу та пошук житла за фільтрами. Вони реалізовані в прикладному та інфраструктурному шарах і взаємодіють із клієнтською частиною через REST API.

2.4. Проектування та розробка клієнтської частини (фронтенду)

Клієнтську частину RentPlatform розроблено як односторінковий застосунок. Такий підхід дає змогу оновлювати інтерфейс без повного перезавантаження сторінки в браузері. Основою фронтенду є React 19 і TypeScript 5.7, а для збирання використано Vite 5. Під час розробки Vite швидко запускає локальний сервер і оновлює змінені модулі без повного перезапуску застосунку [9, 23].

Графічний інтерфейс стилізовано за допомогою Tailwind CSS 4 [24]. У проекті не використано готові набори компонентів на зразок Material UI або Ant Design. Натомість сторінки побудовані з власних компонентів і утилітарних класів Tailwind. Іконки реалізовані через бібліотеку lucide-react.

Картографічний модуль побудовано на Leaflet і react-leaflet. Він відображає об'єкти нерухомості за координатами, отриманими з API. Для маркерів використано власну SVG-іконку через L.DivIcon, що спрощує роботу зі статичними ресурсами у Vite. Центр карти обчислюється на основі координат переданих оголошень, а натискання на маркер відкриває Popur із короткою інформацією про житло. Код компонента карти наведено в Додатку А.

Стан клієнтської частини організовано без окремої великої бібліотеки керування станом. Значення форм і списки даних зберігаються у компонентах через useState. Дані авторизованого користувача зберігаються в localStorage, що є простим рішенням для MVP, але в продуктивній системі потребує додаткового захисту від XSS-ризиків. Для повідомлень використано UnreadContext, який зберігає SignalR-з'єднання та лічильник непрочитаних діалогів [13, 19].

Навігацію реалізовано через React Router 7. Для приватних сторінок створено компоненти PrivateRoute та AdminRoute. PrivateRoute перевіряє наявність токена і, якщо користувач не авторизований, перенаправляє його на сторінку входу. AdminRoute додатково перевіряє роль адміністратора. Фрагмент реалізації маршрутів наведено в Додатку Б.

Запити до серверного REST API виконуються через Axios. У файлі axios.ts створено спільний екземпляр клієнта з базовою адресою API та перехоплювачем

запитів. Перед кожним запитом перехоплювач бере JWT-токен із localStorage і додає його до заголовка Authorization за схемою Bearer. Завдяки цьому компоненти не формують авторизаційні заголовки вручну. Відповідний код наведено в Додатку В [25].

На стороні клієнта SignalR підключається в компоненті UnreadProvider. Під час створення з'єднання використовується accessTokenFactory, який передає актуальний JWT-токен для автентифікації WebSocket-каналу. Також увімкнено withAutomaticReconnect(), щоб з'єднання могло відновлюватися після короткочасної втрати мережі.

Для роботи з конкретним діалогом створено хук useChat. Він не відкриває нове SignalR-з'єднання, а використовує глобальне з'єднання з UnreadContext. Хук підписується на події ReceiveMessage і MessageSent, фільтрує повідомлення за ідентифікатором співрозмовника та викликає серверний метод SendMessage для надсилання тексту. При демонтуванні компонента обробники подій знімаються, щоб уникнути дублювання подій. Код хука наведено в Додатку Г, а загальну структуру клієнтської частини показано на (рис. 2.7).

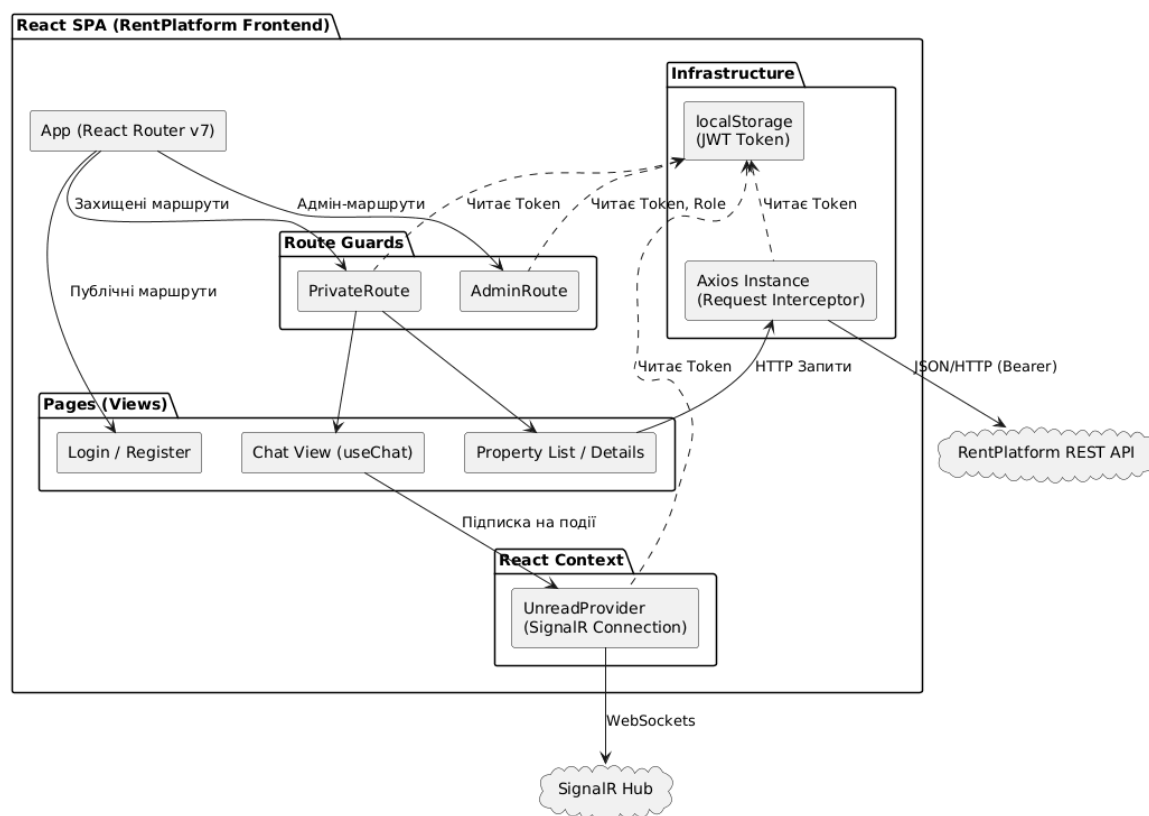


Рис. 2.7. Структурна діаграма компонентів клієнтської частини

Загалом клієнтська частина RentPlatform не є окремою самодостатньою системою, а працює як інтерфейс до серверного API. Її завдання полягає у зручному представленні даних, навігації між сторінками, передаванні запитів користувача та відображенні результатів. Такий поділ дозволяє зберігати бізнес-логіку на сервері, а фронтенд використовувати для швидкої та зрозумілої взаємодії з платформою.

Під час розробки фронтенду також враховувалася робота з помилками. Якщо сервер повертає повідомлення про помилку, клієнтський інтерфейс має показати користувачу зрозумілу реакцію, а не залишати сторінку в невизначеному стані. Для форм важливо очищувати або блокувати кнопки під час виконання запиту, щоб уникати випадкового повторного надсилання однакових даних. У поточній версії ці механізми реалізовані на рівні окремих сторінок і компонентів, що відповідає масштабу MVP.

Адміністративна частина клієнтського застосунку захищена окремою перевіркою ролі. Якщо користувач не має ролі Admin, він не може перейти до відповідних сторінок через звичайну навігацію. На цих сторінках адміністратор може переглядати інформацію для модерації, приховувати або відновлювати оголошення та блокувати користувачів. Така логіка відображає рольову модель, реалізовану на сервері, і не суперечить їй.

Для орендодавця реалізовано окремі сторінки керування власними оголошеннями та заявками. У цих розділах фронтенд відображає лише ті дії, які відповідають поточному стану об'єкта або бронювання. Наприклад, для заявки в очікуванні доступні дії підтвердження або скасування, а для завершених заявок керуючі кнопки не відображаються. Така поведінка робить інтерфейс зрозумілішим для користувача, хоча основна перевірка допустимості переходу все одно виконується на сервері.

Сторінка детального перегляду оголошення поєднує інформаційний блок, карусель або перелік зображень, карту, контактні дії та можливість створення бронювання. Важливо, що сам розрахунок і перевірка бронювання не залишаються лише на клієнті. Інтерфейс може попередньо перевірити дати та

розрахувати орієнтовну суму, але остаточне рішення про створення заявки приймає сервер після перевірки бізнес-правил. Це відповідає загальному принципу, за яким клієнтська частина не повинна бути єдиним місцем контролю важливих операцій.

На головній сторінці користувач працює з каталогом оголошень. Фронтенд передає до API параметри пошуку, міста, типу житла, мінімальної та максимальної ціни, кількості кімнат і наявності генератора. Після отримання відповіді клієнт відображає список об'єктів і метадані пагінації. Це дозволяє не завантажувати всі оголошення одразу, а працювати лише з тією частиною каталогу, яка потрібна користувачу на поточній сторінці.

Структура фронтенду поділена на сторінки, компоненти, API-модулі, контексти та користувацькі хуки. Сторінки відповідають за окремі сценарії взаємодії користувача із системою, наприклад перегляд каталогу, авторизацію, роботу з бронюваннями або чат. Компоненти використовуються для повторюваних елементів інтерфейсу, таких як навігаційна панель, картки оголошень, модальні вікна та блоки фільтрації. API-модулі ізолюють запити до серверної частини, тому сторінки не формують HTTP-запити вручну, а використовують підготовлені функції.

Висновки до другого розділу

У другому розділі описано проєктування та реалізацію основних частин RentPlatform. Було розглянуто структуру бази даних, серверний REST API, алгоритми бронювання і фільтрації, а також клієнтський інтерфейс. У результаті сформовано цілісний прототип вебплатформи для оренди житла.

База даних реалізована в PostgreSQL і містить п'ять основних сутностей: користувачі, об'єкти нерухомості, бронювання, обране та повідомлення. Між сутностями налаштовано зовнішні ключі, а для окремих таблиць використано унікальні індекси. Для взаємодії з базою даних застосовано Entity Framework Core 10 і підхід Code-First.

У системі використано UUID як тип первинних ключів. Для об'єктів нерухомості передбачено окреме поле HasGenerator, яке дозволяє фільтрувати житло за

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						39

наявністю альтернативного живлення. Конфігурація таблиць і зв'язків описана через Fluent API у методі OnModelCreating.

Серверна частина побудована як REST API на ASP.NET Core 10.0 за принципами Clean Architecture. Основні контролери відповідають за автентифікацію, оголошення, бронювання, повідомлення, обране, профіль користувача та адміністрування. DevController має службове призначення. Для перевірки даних використано DataAnnotations і додаткову бізнес-валідацію в сервісах, а перетворення між сутностями та DTO виконується вручну.

Бізнес-логіка системи зосереджена в прикладних сервісах. Алгоритм бронювання перевіряє доступність дат, кінцевий автомат обмежує допустимі переходи між статусами заявки, а пошуковий механізм формує динамічний IQueryable-запит із фільтрацією, сортуванням і пагінацією. Після завершення бронювання баланс орендодавця оновлюється на суму заявки.

Клієнтську частину реалізовано на React 19 і TypeScript 5.7 з використанням Vite 5 та Tailwind CSS 4. Для навігації застосовано React Router 7, для HTTP-запитів – Axios, а для повідомлень у реальному часі – SignalR. Такий набір технологій дозволив реалізувати каталог житла, сторінку деталей, обране, бронювання, чат, особисті кабінети й адміністративні функції.

Отже, у другому розділі показано, як проєктні рішення були реалізовані в коді. Описані компоненти створюють основу для запуску, тестування та подальшого розвитку платформи, що розглядається в третьому розділі роботи.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБПЛАТФОРМОЮ ОРЕНДИ ЖИТЛА

3.1. Порядок встановлення та налаштування параметрів системи

Для забезпечення коректного функціонування розподіленого клієнт-серверного застосунку RentPlatform необхідне попереднє налаштування середовища виконання та встановлення відповідних програмних залежностей. Процес розгортання та конфігурації системи логічно розділяється на два вектори: локальне налаштування для розробки та тестування (Local Development) та підготовка серверної інфраструктури до продуктивного використання у хмарному середовищі Microsoft Azure (Production).

Враховуючи обраний сучасний технологічний стек, базовими вимогами до апаратного та програмного забезпечення робочої станції адміністратора або розробника є наявність встановленого кросплатформного середовища виконання .NET SDK версії 10.0 для компіляції серверної частини, платформи Node.js для управління пакетами та збирання клієнтського застосунку через Vite, а також локального екземпляра реляційної СКБД PostgreSQL. У межах цієї версії проєкту свідомо не застосовувалися технології контейнеризації (наприклад, Docker), тому запуск та координація всіх компонентів здійснюється безпосередньо на хост-машині засобами командного рядка.

Архітектура фізичного розгортання системи передбачає гібридну модель. Серверний програмний інтерфейс REST API розміщено у хмарному середовищі Microsoft Azure, а база даних PostgreSQL функціонує як окремий віддалений ресурс. Клієнтська частина під час тестування та демонстрації запускається локально за допомогою Vite на робочій станції розробника та взаємодіє з віддаленим API через мережу Інтернет. Фізичну топологію вузлів системи та протоколи їхньої взаємодії проілюстровано на UML-діаграмі розгортання (рис. 3.1).

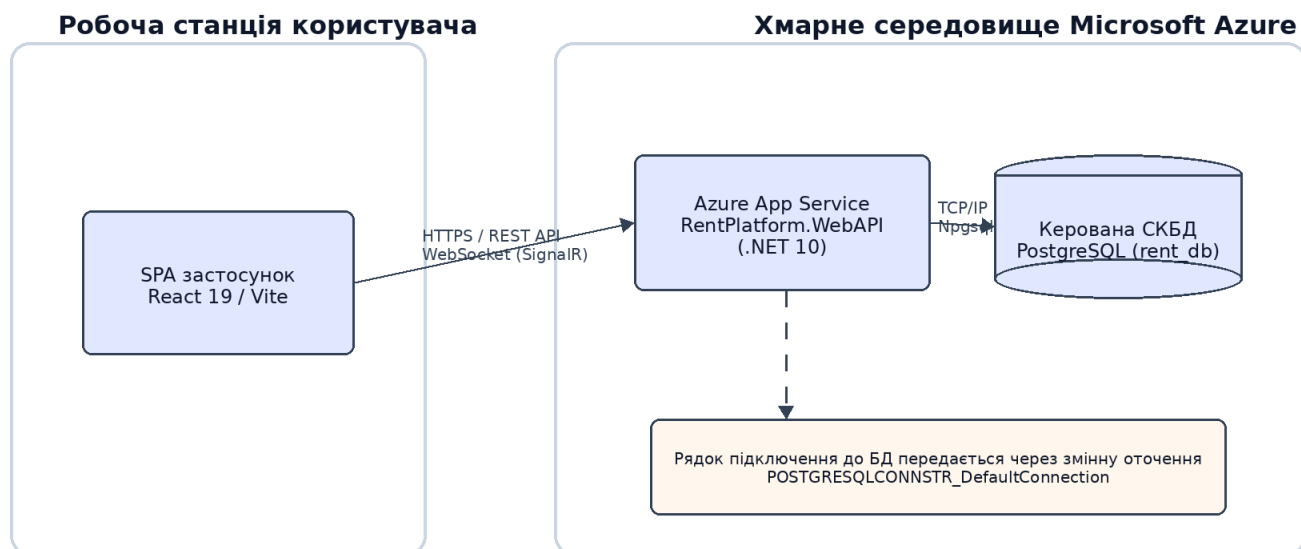


Рис. 3.1. UML-діаграма фізичного розгортання компонентів системи

Централізоване налаштування серверної частини виконується через конфігураційний файл `appsettings.json`, розташований у кореневій директорії проєкту `RentPlatform.WebAPI`. Цей файл містить ієрархічну структуру параметрів у форматі JSON, які десеріалізуються системою під час запуску та впроваджуються у сервіси через механізм `IOptions`. Для детального розуміння конфігураційних можливостей бекенду, опис ключових секцій та параметрів налаштування зведено у таблиці 3.1.

Таблиця 3.1

Специфікація ключових параметрів конфігурації серверної частини
(`appsettings.json`)

Назва секції / Параметра	Тип даних	Опис та функціональне призначення параметра
ConnectionStrings	Object	Блок налаштувань підключення до баз даних.
DefaultConnection	String	Рядок підключення до СКБД PostgreSQL (містить Host, Port, Database, Username, Password).
JwtSettings	Object	Блок конфігурації підсистеми автентифікації.
SecretKey	String	Криптографічний ключ для генерації та валідації симетричного підпису JWT-токенів.

1.	2.	3.
Issuer	String	Ідентифікатор довіреного видавця токенів (за замовчуванням: "RentPlatform").
Audience	String	Ідентифікатор цільової аудиторії токенів (за замовчуванням: "RentPlatformUsers").
ExpirationDays	Integer	Термін дії токена доступу в добах (встановлено значення 7).
Logging	Object	Рівні деталізації системного журналування (Information, Warning, Error).
AllowedHosts	String	Визначає дозволені значення HTTP-заголовка Host. У поточній конфігурації встановлено "*", що дозволяє обробляти запити з різними іменами хостів.

У продуктивному середовищі рядок підключення до бази даних передається через змінну оточення `POSTGRESQLCONNSTR_DefaultConnection`. Налаштування CORS, Swagger та SignalR виконуються програмно у файлі `Program.cs`, тому вони не винесені до таблиці параметрів `appsettings.json`.

Важливим інженерним аспектом кібербезпеки продуктивного середовища є механізм роботи з чутливими даними. Рядок підключення до віддаленої продуктивної бази даних у хмарі не зберігається у файлі `appsettings.json`, щоб уникнути витoku інформації через систему контролю версій Git. Натомість використовується механізм ін'єкції через змінні оточення (Environment Variables). Рядок підключення передається серверу на етапі ініціалізації через захищену змінну `POSTGRESQLCONNSTR_DefaultConnection` безпосередньо у панелі керування Azure App Service. Політики міждоменних запитів (CORS) також не винесені у конфігураційний файл, а імплементовані програмно на рівні коду `Program.cs` через політику `AllowReactApp`.

Для локального запуску клієнтської частини та перевірки взаємодії з віддаленим API розробник переходить до директорії `client`, встановлює залежності командою `"npm install"` та запускає локальний сервер Vite командою `"npm run dev"`. Серверна частина у поточній версії не запускається локально під час демонстрації, оскільки її опубліковано в Microsoft Azure разом із підключенням до віддаленої бази даних PostgreSQL. За потреби локального розгортання бекенду необхідно створити локальну базу `rent_db`, указати власний

рядок підключення у конфігурації, застосувати міграції Entity Framework командою “dotnet ef database update” і запустити WebAPI командою “dotnet run”.

У поточній архітектурі під час тестування фронтенд-застосунок запускається локально, проте конфігурація файлу axios.ts та параметри підключення HubConnectionBuilder для SignalR налаштовані на звернення до хмарного домену Azure. Такий підхід дозволяє перевіряти взаємодію локального клієнтського інтерфейсу з віддаленими серверними ресурсами та виявляти помилки, пов'язані з CORS, авторизацією, мережевими затримками й роботою WebSocket-з'єднання.

3.2. Структура інтерфейсу вебплатформи

Графічний інтерфейс вебплатформи RentPlatform спроектовано з урахуванням сучасних принципів юзабіліті (Usability) та User Experience (UX), що забезпечує інтуїтивну навігацію для користувачів різних ролей (орендарів, орендодавців, адміністраторів). Колірна гама, відступи та типографіка уніфіковані за допомогою утилітарних класів фреймворку Tailwind CSS, що дозволило створити адаптивний (Responsive) дизайн, який коректно відображається як на десктопних, так і на мобільних пристроях.

Базовим елементом навігації на всіх екранах системи є верхня панель (Navbar). Вона містить логотип платформи, бейдж поточної ролі користувача, аватар з ініціалом та індикатор непрочитаних повідомлень чату. Залежно від авторизованої ролі, склад навігаційних елементів динамічно змінюється: орендодавці бачать поточний фінансовий баланс, вкладки «Заявки» та «Мої оголошення», тоді як орендарям доступні розділи «Обране» та «Мої бронювання».

Першою точкою взаємодії користувача із системою є головна сторінка каталогу нерухомості (Home) (рис. 3.2). Вона структурно поділяється на панель розширених фільтрів та адаптивну сітку карток оголошень. Панель фільтрації (PropertyFilters) надає інструментарій для багатовимірного пошуку: рядок текстового пошуку за назвою, випадаючі списки вибору міста та типу житла, числові поля для задання цінового діапазону та мінімальної кількості кімнат.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						44

Окремо виділено стилізований тумблер (toggle-перемикач) для миттєвої фільтрації об'єктів з наявним альтернативним джерелом живлення («Генератор»).

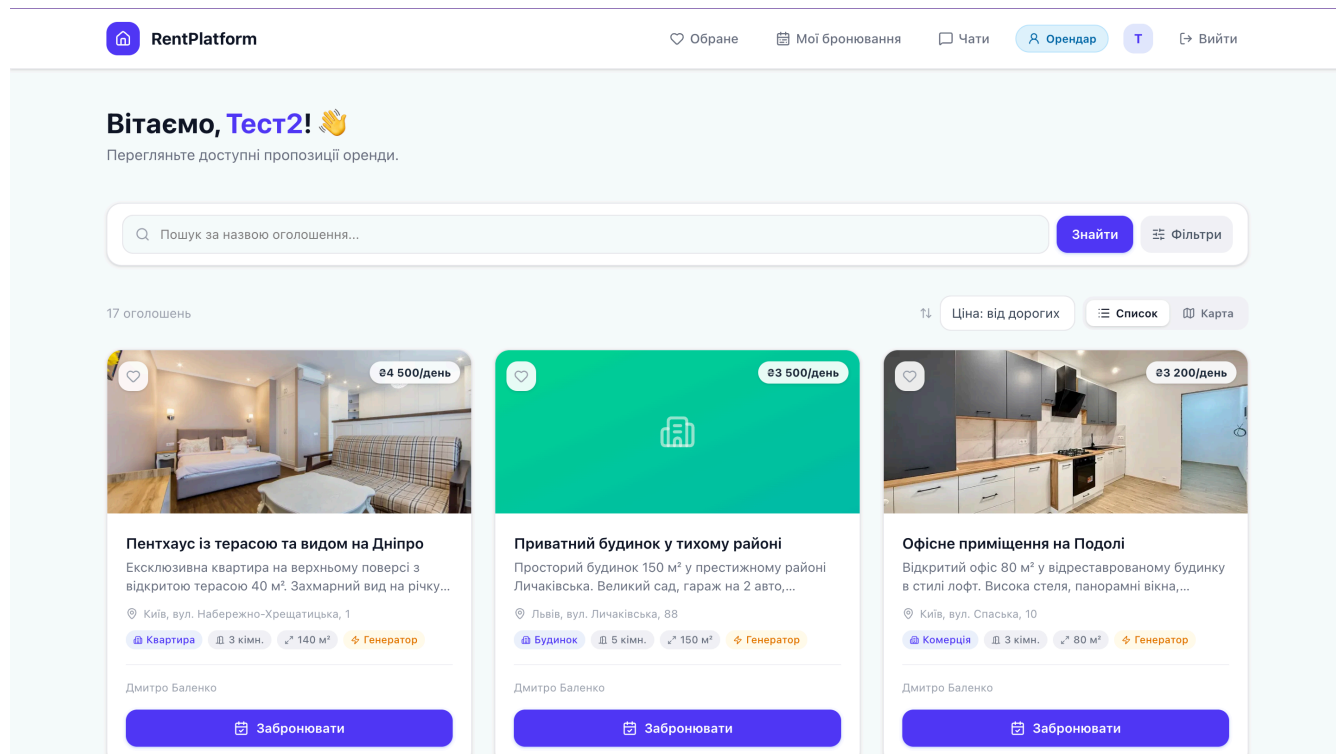


Рис. 3.2. Головна сторінка вебплатформи

Каталог підтримує два режими відображення: класичний список (Grid) та географічну карту. На короткій картці оголошення виводиться мініатюра фотографії, бейдж актуальної ціни за добу, текстовий опис та теги характеристик (площа, кількість кімнат, наявність генератора). Для орендаторів на картці передбачено кнопку додавання в обране («сердечко») з механізмом оптимістичного оновлення (Optimistic UI Update) та кнопку швидкого переходу до бронювання. Навігація між сторінками видачі реалізована через блок двосторонньої пагінації. Візуальну структуру головної сторінки представлено на (рис. 3.3).

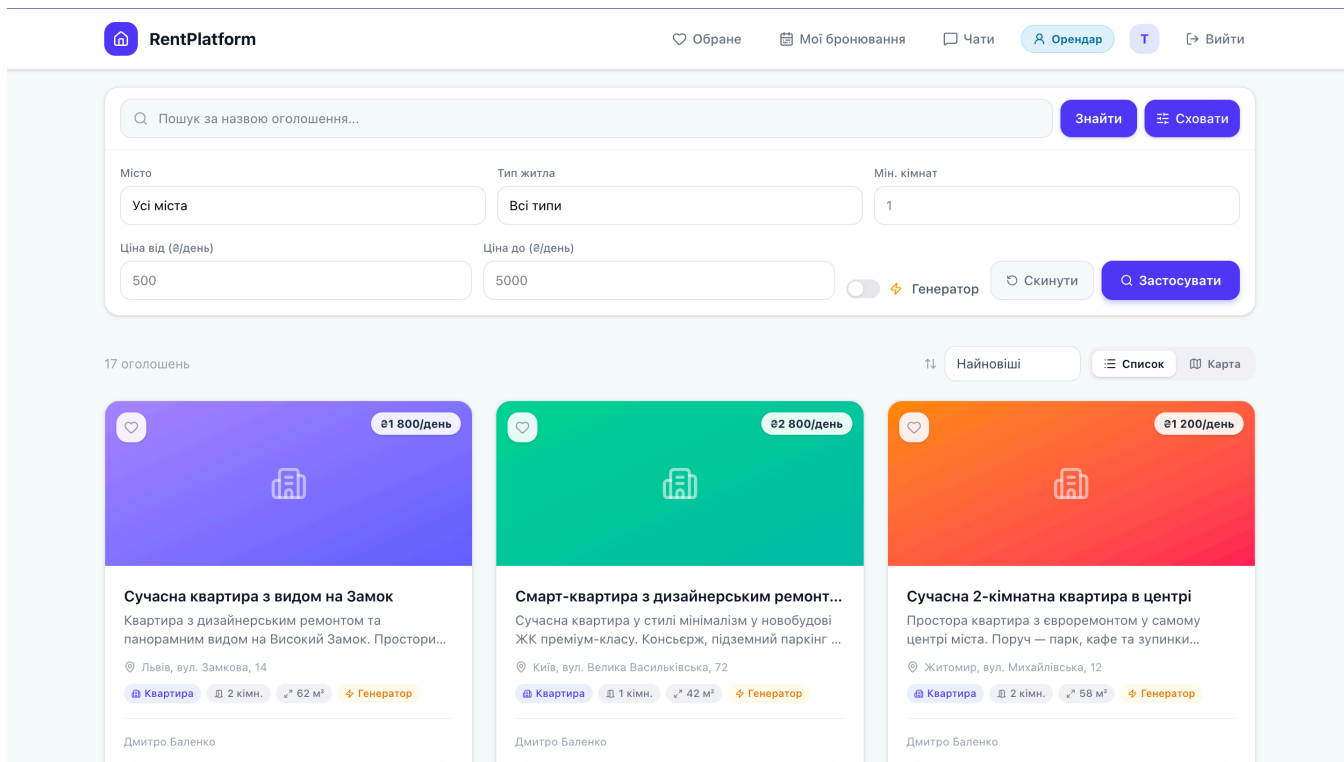


Рис. 3.3. Головна сторінка каталогу та панель фільтрів

При переході на сторінку конкретного оголошення (PropertyDetail) користувач отримує детальну інформацію про об'єкт у форматі розширеної картки на всю ширину контентної області. У верхній частині розташовано карусель фотографій (PropertyCarousel). Під нею виводиться повний опис, адреса та структурований блок характеристик у вигляді інформаційних плиток. Внизу сторінки розміщено цінову панель та дві ключові кнопки дій: «Написати» (для переходу в чат із власником) та «Забронювати». Для спілкування в чаті, або бронювання житла, користувачу необхідно авторизуватися в системі. Цей процес відбувається за допомогою відповідної форми на сайті, де користувач вводить свою електронну пошту та пароль в системі. Форма авторизації наведена на (рис. 3.4).

Рис. 3.4. Форма авторизації користувача

Процес ініціалізації оренди не вимагає переходу на окрему сторінку, а відбувається в контекстному модальному вікні (BookProperty). Форма бронювання використовує нативні елементи вибору дат HTML5 (`<input type="date">`), для яких встановлено строгі обмеження (атрибут `min`), що унеможлиблюють вибір дат у минулому або ситуацію, коли дата виїзду передуює даті заїзду. При виборі періоду система в реальному часі виконує автоматичний розрахунок кількості ночей та підсумкової вартості заявки. Інтерфейс вікна деталей та форми бронювання проілюстровано на (рис. 3.5).

Рис. 3.5. Сторінка детальної інформації про житло та форма бронювання

Внутрішня комунікація користувачів реалізована через модуль обміну повідомленнями. Архітектурно модуль розділено на дві навігаційно пов'язані сторінки: список активних діалогів (Chats) та вікно конкретного листування (Chat). На сторінці списку діалогів виводяться інтерактивні картки з аватаром співрозмовника, попередній перегляд останнього повідомлення та червоним бейджем кількості непрочитаних листів.

Вікно конкретного діалогу (рис. 3.6) побудовано за патерном класичного месенджера. У верхній частині екрана відображається статус з'єднання за протоколом WebSockets («Онлайн» / «Підключення...»). Центральна частина містить стрічку повідомлень, де власні репліки вирівняні за правим краєм (заливка кольором indigo), а повідомлення співрозмовника – по лівому (білі з рамкою). Стрічка автоматично позиціонується до останнього повідомлення при завантаженні або надходженні нового повідомлення. У нижній частині розташовано текстове поле вводу (<textarea>) з підтримкою перенесення рядка та кнопка надсилення, яка програмно блокується за відсутності тексту або при втраті підключення до сервера [26].

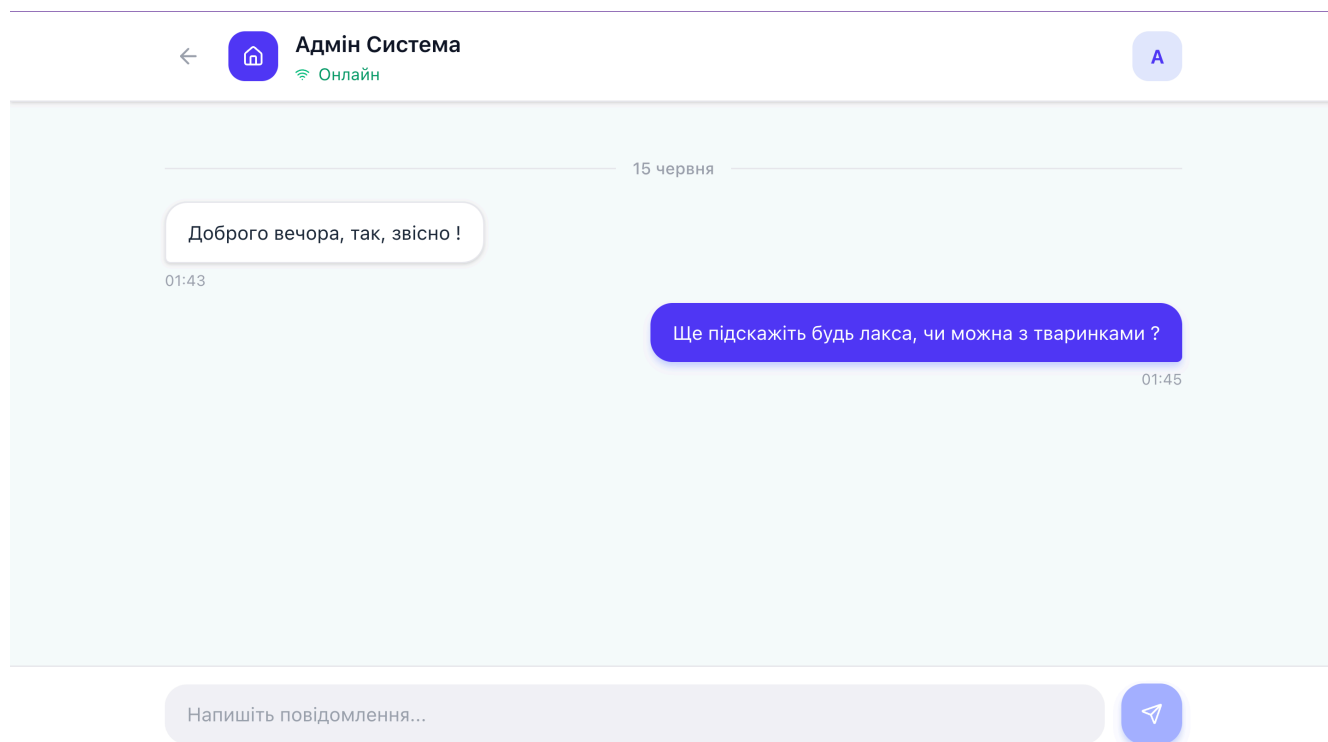


Рис. 3.6. Вікно внутрішнього чату та статуси з'єднання

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						48

Управління життєвим циклом заявок для орендодавця реалізовано на сторінці особистого кабінету (LandlordBookings). На відміну від класичних табличних структур, інтерфейс подано у вигляді списку інтерактивних карток (рис. 3.7), що суттєво покращує читабельність на мобільних пристроях [27]. У верхній частині сторінки розташовано панель аналітичної статистики: загальна кількість заявок, кількість в очікуванні, підтверджених та успішно завершених угод.

Кожна картка заявки містить фотографію об'єкта, кольоровий бейдж поточного статусу, контактні дані орендаря з кнопкою швидкого переходу до чату, обрані дати та фінальну суму. Логіка відображення кнопок керування на картці жорстко прив'язана до алгоритму кінцевого автомата (State Machine) на бекенді. Для заявок у статусі «Очікує» (Pending) доступні кнопки «Підтвердити» та «Скасувати». Для підтверджених заявок (Confirmed) активуються кнопки «Завершити» (що ініціює зарахування коштів на баланс орендодавця) або «Скасувати». Для заявок, що перейшли у термінальні стани (Completed або Cancelled), елементи керування ховаються і замінюються відповідною текстовою плашкою, запобігаючи невалідним переходам статусу.

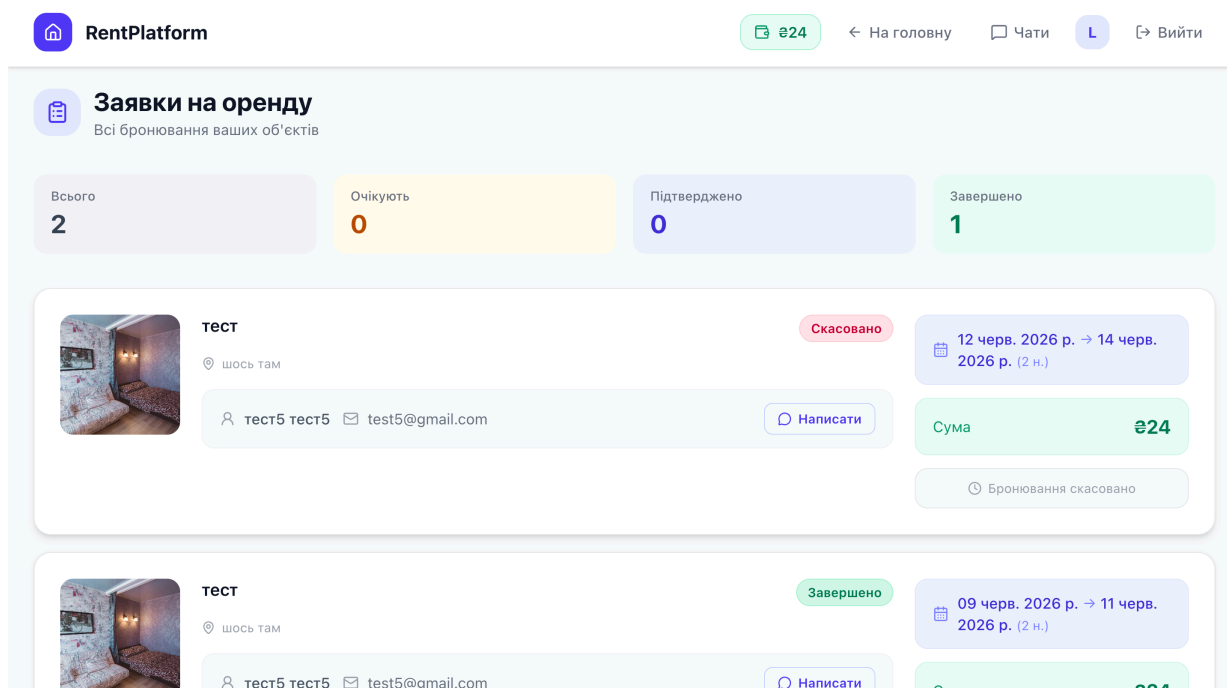


Рис. 3.7. Особистий кабінет орендодавця для управління заявками на бронювання

3.3. Тестування роботи програмного комплексу

Завершальним етапом життєвого циклу розробки програмного забезпечення є перевірка його працездатності, виявлення дефектів та підтвердження відповідності реалізованого функціоналу початковим вимогам. Зважаючи на статус проєкту як MVP (Minimum Viable Product), стратегію забезпечення якості (Quality Assurance) було побудовано на основі методів ручного функціонального та інтеграційного тестування (Manual Testing) [28, 29]. Впровадження інфраструктури автоматизованого модульного тестування (Unit Testing) визначено як пріоритетний напрям для наступних ітерацій розвитку платформи.

Процес тестування охоплював перевірку серверного програмного інтерфейсу, клієнтської частини та основних наскрізних бізнес-процесів. Такий поділ дав змогу окремо перевірити коректність REST API, роботу React-інтерфейсу та взаємодію між ролями орендаря, орендодавця й адміністратора.

Для ізольованого тестування бекенду без залучення клієнтського інтерфейсу використовувалася вбудована інтерактивна документація Swagger UI (OAS 3.0). Завдяки налаштуванню генератора Swagger у класі Program.cs система автоматично формує візуальний інтерфейс для взаємодії з усіма зареєстрованими кінцевими точками контролерів.

Під час тестування серверної частини через Swagger UI перевірялися маршрути отримання каталогу оголошень, вибірки власних бронювань, отримання статистики адміністратора, створення заявок і зміни статусів бронювання. Окремо перевірялися реакції API на невалідні DTO-моделі, відсутність JWT-токена та спроби звернення до захищених маршрутів без відповідної ролі. Після виконання запитів аналізувалися HTTP-коди відповідей, зокрема 200 OK, 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found і 409 Conflict, а також коректність повідомлень про помилки. Приклад роботи з інтерактивним інтерфейсом Swagger UI наведено на (рис. 3.8).

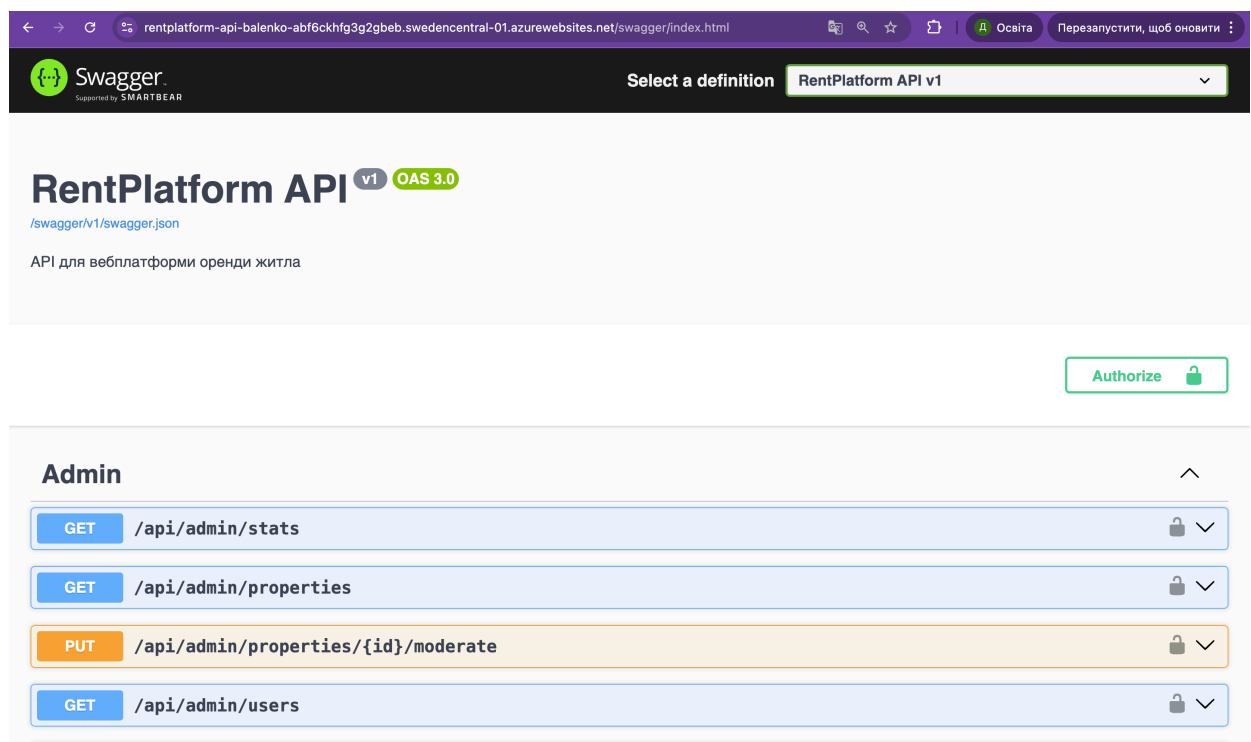


Рис. 3.8. Інтерактивний інтерфейс Swagger UI для тестування REST API

Тестування клієнтської частини та інтеграції. Перевірка коректності роботи клієнтського React-застосунку та його взаємодії з бекендом виконувалася за допомогою інструментів розробника у браузері (Chrome Developer Tools). Основна увага приділялася вкладці моніторингу мережі (Network) для аналізу вихідних XHR/Fetch запитів, які генерує бібліотека Axios [25]. Перевірялося коректне додавання JWT-токена перехоплювачем до заголовків запиту та правильність обробки CORS-політик. За допомогою вкладки Console відслідковувалася відсутність критичних помилок JavaScript під час рендерингу компонентів та роботи кастомних хуків (рис. 3.9). Такий підхід відповідає ручній перевірці інтеграційних сценаріїв у межах навчального MVP-проєкту [30].

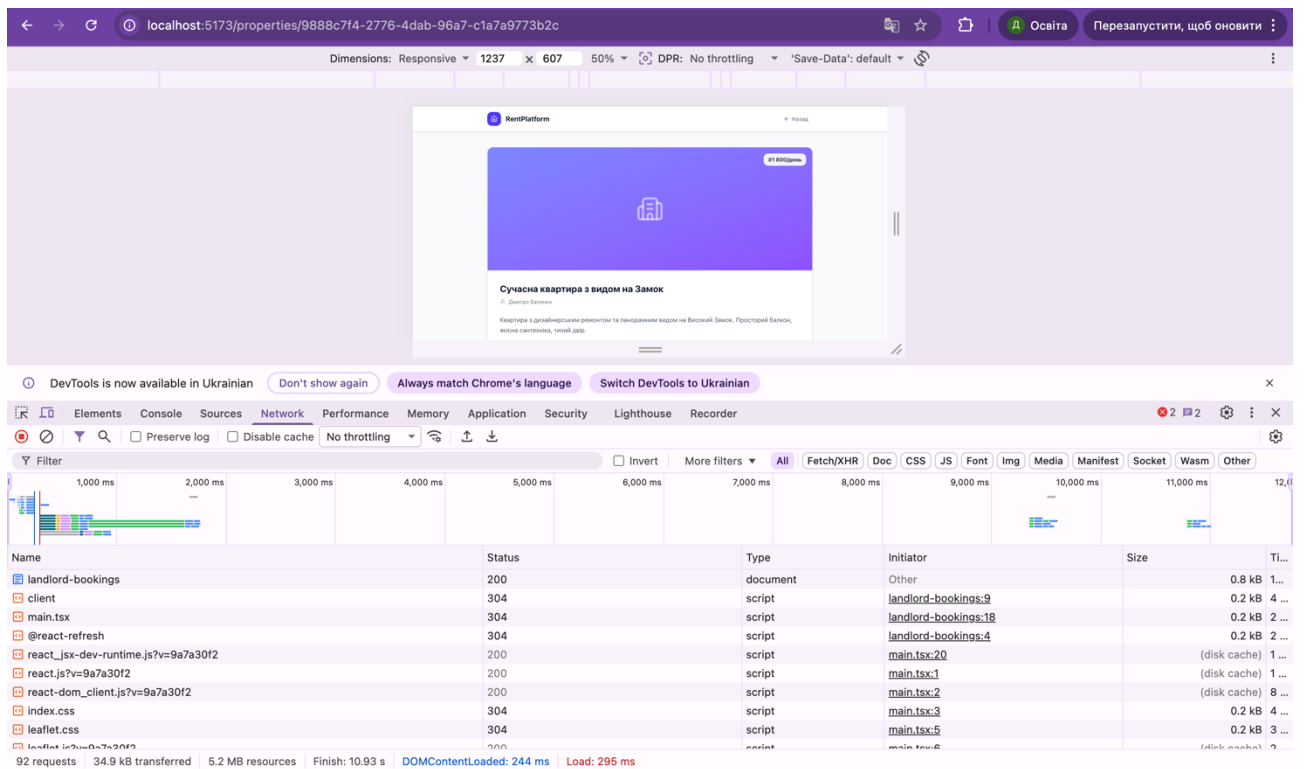


Рис. 3.9. Моніторинг мережевих запитів фронтенд-застосунку

Наскрізне тестування бізнес-процесів виконувалося в реальному інтерфейсі системи під обліковими записами орендаря, орендодавця та адміністратора. Під час перевірки орендодавців створював оголошення, додавав фотографії та перевіряв появу об'єкта в загальному каталозі. Орендар виконував пошук житла за фільтрами, відкривав сторінку деталей, додавав об'єкт в обране та створював заявку на бронювання. Окремо перевірялася спроба повторного бронювання на період, що перетинається з уже створеною заявкою, після чого система повертала повідомлення про недоступність вибраних дат. Також перевірялися підтвердження, скасування та завершення заявки орендодавцем, оновлення балансу, надсилання повідомлень у чаті та дії адміністратора з приховування оголошення і блокування користувача.

Основні сценарії ручного тестування системи наведено у таблиці 3.2.

Таблиця 3.2

Результати ручного функціонального тестування

Перевірений сценарій	Дії користувача	Очікуваний результат	Результат перевірки
Автентифікація користувача	Реєстрація та вхід під ролями орендаря й орендодавця; вхід під роллю адміністратора (акаунт створено через початкове наповнення).	Після входу система відкриває розділи, доступні для відповідної ролі.	Виконано коректно (ручна перевірка).
Створення оголошення	Орендодавець створює оголошення, додає характеристики житла та фотографії.	Оголошення зберігається і відображається в каталозі після оновлення даних.	Виконано коректно (ручна перевірка).
Пошук і фільтрація	Орендар застосовує фільтри за містом, ціною, кількістю кімнат і наявністю генератора.	Каталог оновлюється відповідно до заданих параметрів пошуку.	Виконано коректно (ручна перевірка).
Бронювання житла	Орендар створює заявку на вільні дати та повторно пробує обрати зайнятий період.	Заявка на вільні дати створюється, а спроба бронювання з перетином дат блокується.	Виконано коректно (ручна перевірка).
Керування заявкою	Орендодавець підтверджує, скасовує або завершує заявку.	Статус змінюється лише за дозволеними переходами, після завершення оновлюється баланс.	Виконано коректно (ручна перевірка).
Чат користувачів	Орендар і орендодавець надсилають повідомлення з різних облікових записів.	Повідомлення зберігаються, відображаються в діалозі та оновлюють лічильник непрочитаних.	Виконано коректно (ручна перевірка).
Адміністрування	Адміністратор приховує оголошення та блокує користувача.	Приховане оголошення не показується в каталозі, заблокований користувач не може увійти.	Виконано коректно (ручна перевірка).

Усі результати в таблиці 3.2 отримані під час ручного функціонального тестування. Автоматизовані unit- та integration-тести в поточній версії проєкту не реалізовані.

За результатами ручної перевірки основні користувацькі сценарії виконувалися коректно. Під час налагодження було усунуто дрібні недоліки інтерфейсу та окремі помилки обробки станів, після чого повторна перевірка

підтвердила працездатність ключових функцій платформи. Автоматизовані unit- та integration-тести в поточній версії проєкту не реалізовано, тому їх додавання визначено як один із напрямів подальшого розвитку системи.

Висновки до третього розділу

У третьому розділі кваліфікаційної роботи описано порядок налаштування та перевірки роботи вебплатформи RentPlatform. Наведено конфігурацію серверної частини через файл appsettings.json та описано фактичну інфраструктуру, у якій REST API і база даних PostgreSQL розгорнуті в Microsoft Azure, а клієнтська частина під час тестування запускається локально за допомогою Vite. Безпеку облікових даних до бази даних забезпечено застосуванням змінних оточення замість зберігання чутливих параметрів у репозиторії. Топологію системи та взаємодію її компонентів формалізовано за допомогою UML-діаграми розгортання.

Розглянуто структуру графічного інтерфейсу, побудованого за моделлю односторінкового застосунку (SPA) з використанням бібліотеки React та стилізованого за допомогою Tailwind CSS. Інтерфейс забезпечує навігацію для користувачів різних ролей та містить основні інтерактивні елементи: адаптивний каталог із фільтрами, картографічний модуль Leaflet, модуль комунікацій у реальному часі на базі SignalR та кабінет управління заявками. Контроль доступу та навігацію реалізовано засобами React Router.

Перевірку працездатності програмного забезпечення проведено методами ручного функціонального та інтеграційного тестування. За допомогою Swagger UI перевірено маршрути REST API, роботу авторизації та валідації запитів. Аналіз мережевої взаємодії через Chrome Developer Tools і проходження основних користувацьких сценаріїв підтвердили працездатність каталогу, бронювання, чату, обраного, адміністративної модерації та зміни статусів заявок. Автоматизовані тести не входять до поточної версії MVP і розглядаються як напрям подальшого вдосконалення.

ВИСНОВКИ

У кваліфікаційній роботі вирішено завдання з проєктування, програмної реалізації та перевірки розподіленої клієнт-серверної вебплатформи RentPlatform, призначеної для автоматизації процесів оренди житлової нерухомості. Розроблений програмний комплекс поєднує каталог оголошень, фільтрацію житла, бронювання дат, внутрішній чат, обране та адміністративну модерацію користувачів і оголошень.

Під час виконання роботи розроблено логічну та фізичну структуру реляційної бази даних у СКБД PostgreSQL з оптимізованими індексами, а також серверну частину на базі Clean Architecture під управлінням .NET Core 10.0. Безпеку та працездатність бізнес-логіки забезпечують алгоритми перевірки перетину дат і кінцевий автомат статусів бронювань. Клієнтський односторінковий застосунок реалізовано на React 19 та Tailwind CSS, а модуль комунікацій у реальному часі побудовано на основі технології SignalR. Захист приватних зон забезпечено безсесійною автентифікацією за допомогою stateless JWT-токенів. Практична цінність результатів підтверджена успішним ручним функціональним та інтеграційним тестуванням, а також хмарним розгортанням сервісів у середовищі Microsoft Azure App Service із механізмом ін'єкції чутливих даних через змінні оточення.

Перспективними напрямками подальшого вдосконалення розробленого програмного комплексу є покриття бізнес-логіки прикладного шару автоматизованими тестами xUnit, розширення підсистеми безпеки через впровадження механізму refresh-токенів, а також імплементація централізованого шару обробки помилок за допомогою Global Exception Middleware для автоматичного логування системних винятків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бублик В. В. Об'єктно-орієнтоване проектування програмних систем : навч. посіб. Київ : НТУУ «КПІ», 2015. 240 с.
2. Пасічник В. В., Резніченко В. А. Організація баз даних та знань. Київ : Видавнича група ВНУ, 2006. 384 с.
3. Кормен Т., Лейзерсон Ч., Рівест Р., Штайн К. Вступ до алгоритмів : пер. з англ. 3-тє вид. Київ : К.І.С., 2019. 1288 с.
4. Шпортько О. В. Веб-технології та проектування веб-застосунків : навч. посіб. Рівне : РДГУ, 2019. 182 с.
5. Lock A. ASP.NET Core in Action. 2nd ed. Shelter Island : Manning Publications, 2021. 824 p.
6. Airbnb. Vacation rentals, cabins, beach houses, & more. URL: <https://www.airbnb.com/> (дата звернення: 10.06.2026).
7. Booking.com. Official site. URL: <https://www.booking.com/> (дата звернення: 10.06.2026).
8. DIM.RIA. Вся нерухомість України. URL: <https://dom.ria.com/uk/> (дата звернення: 10.06.2026).
9. Vite. Guide. URL: <https://vite.dev/guide/> (дата звернення: 10.06.2026).
10. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol : O'Reilly Media, 2021. 280 p.
11. Мартин Р. Чиста архітектура. Мистецтво розроблення програмного забезпечення. Харків : Фабула, 2019. 368 с.
12. Freeman A. Pro ASP.NET Core 6: Develop Cloud-Ready Web Applications Using MVC, Blazor, and Razor Pages. 9th ed. Berkeley : Apress, 2022. 1204 p.
13. React Documentation. The library for web and native user interfaces. URL: <https://react.dev/> (дата звернення: 09.05.2026).
14. Create web APIs with ASP.NET Core. Microsoft Learn. URL: <https://learn.microsoft.com/en-us/aspnet/core/web-api/> (дата звернення: 12.05.2026).
15. Obe R. O., Hsu L. S. PostgreSQL: Up and Running. 3rd ed. Sebastopol : O'Reilly Media, 2017. 290 p.

					ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
						56

16. Cherny B. Programming TypeScript: Making Your JavaScript Applications Scale. Sebastopol : O'Reilly Media, 2019. 322 p.
17. Smith J. P. Entity Framework Core in Action. 2nd ed. Shelter Island : Manning Publications, 2021. 624 p.
18. Model validation in ASP.NET Core MVC and Razor Pages. Microsoft Learn. URL: <https://learn.microsoft.com/en-us/aspnet/core/mvc/models/validation> (дата звернення: 10.06.2026).
19. Introduction to ASP.NET Core SignalR. Microsoft Learn. URL: <https://learn.microsoft.com/en-us/aspnet/core/signalr/introduction> (дата звернення: 04.05.2026).
20. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2002. 560 p.
21. Wagner B. Effective C#: 50 Specific Ways to Improve Your C#. 3rd ed. Boston : Addison-Wesley Professional, 2016. 352 p.
22. Querying data in EF Core. Microsoft Learn. URL: <https://learn.microsoft.com/en-us/ef/core/querying/> (дата звернення: 10.06.2026).
23. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. 2nd ed. Sebastopol : O'Reilly Media, 2020. 336 p.
24. Install Tailwind CSS with Vite. Tailwind CSS Documentation. URL: <https://tailwindcss.com/docs/installation/using-vite> (дата звернення: 10.06.2026).
25. Axios. Interceptors. URL: <https://axios-http.com/docs/interceptors> (дата звернення: 10.06.2026).
26. Купер А., Рейман Р., Кронін Д. Інтерфейс. Основи проєктування взаємодії. Київ : Фабула, 2020. 656 с.
27. Tidwell J., Brewer C., Valencia-Brooks A. Designing Interfaces: Patterns for Effective Interaction Design. 3rd ed. Sebastopol : O'Reilly Media, 2020. 578 p.
28. Myers G. J., Sandler C., Badgett T. The Art of Software Testing. 3rd ed. Hoboken : John Wiley & Sons, 2011. 256 p.
29. Black R., Veenendaal E., Graham D. Foundations of Software Testing: ISTQB Certification. 4th ed. London : Cengage Learning, 2019. 320 p.

					ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
						57

30. Глинський Я. М., Ряжська В. А. Тестування програмного забезпечення : навч. посіб. Львів : Десятий форт, 2020. 208 с.

ДОДАТКИ

					ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						59

Програмний код компонента відображення карти (PropertiesMap.tsx)

```

import L from 'leaflet';
import { useNavigate } from 'react-router-dom';
import { MapContainer, TileLayer, Marker, Popup } from 'react-leaflet';
import { Building2 } from 'lucide-react';
import { BACKEND_BASE, getPropertyTypeLabel, type Property } from '../api/propertyApi';
// Кастомна іконка маркера – inline SVG будиночка у круглому контейнері indigo.
// Використовуємо L.DivIcon замість стандартного L.Icon, щоб уникнути проблем
// із підвантаженням PNG-файлів у середовищі Vite.
const propertyMarkerIcon = new L.DivIcon({
  className: "",
  iconSize: [36, 36],
  iconAnchor: [18, 18],
  popupAnchor: [0, -20],
  html: `
<div style="
width:36px; height:36px;
background:#4f46e5;
border-radius:50%;
border:2.5px solid #fff;
box-shadow:0 2px 8px rgba(0,0,0,0.35);
display:flex; align-items:center; justify-content:center;
color:#fff;
">
<svg xmlns="http://www.w3.org/2000/svg" width="18" height="18"
viewBox="0 0 24 24" fill="none" stroke="currentColor"
stroke-width="2.5" stroke-linecap="round" stroke-linejoin="round">
<path d="M3 9l9-7 9 7v11a2 2 0 0 1-2 2H5a2 2 0 0 1-2-2z"/>
<polyline points="9 22 9 12 15 12 15 22"/>
</svg>
</div>
`,
  });
interface Props {
  properties: Property[];
}
// Обчислює центр карти як середнє значення координат усіх оголошень.
function computeCenter(props: Property[]): [number, number] {
  if (props.length === 0) return [50.2547, 28.6586];

```

```

const lat = props.reduce((sum, p) => sum + p.latitude, 0) / props.length;
const lng = props.reduce((sum, p) => sum + p.longitude, 0) / props.length;
return [lat, lng];
}

export default function PropertiesMap({ properties }: Props) {
  const navigate = useNavigate();
  const center = computeCenter(properties);
  return (
    <div className="rounded-2xl overflow-hidden border border-gray-200 shadow-sm">
      <MapContainer
        center={center}
        zoom={properties.length === 1 ? 14 : 10}
        style={{ height: '620px' }}
        scrollWheelZoom
      >
        <TileLayer
          url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
          attribution='&copy; <a
            href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>'
          />
          {properties.map((p) => (
            <Marker key={p.id} position={[p.latitude, p.longitude]}
              icon={propertyMarkerIcon}>
                <Popup minWidth={220} maxWidth={260}>
                  <div className="font-sans text-sm">
                    {/* Мініатюра фото */}
                    {p.imageUrls.length > 0 ? (
                      <img
                        src={p.imageUrls[0].startsWith('http') ? p.imageUrls[0] :
                          `${BACKEND_BASE}${p.imageUrls[0]}`
                        alt={p.title}
                        className="w-full h-28 object-cover rounded-lg mb-2"
                      />
                    ) : (
                      <div className="w-full h-28 bg-gradient-to-br from-indigo-400 to-violet-500
                        rounded-lg mb-2 flex items-center justify-center">
                        <Building2 size={28} className="text-white/70" />

```

```

</div>
))}
{ /* Назва та тип */}
<p className="font-semibold text-gray-900 leading-tight truncate">{p.title}</p>
<p className="text-xs text-gray-500 mt-0.5">
{getPropertyTypeLabel(p.propertyType)} · {p.city}
</p>
{ /* Ціна */}
<p className="text-indigo-600 font-bold mt-1.5">
{p.pricePerDay.toLocaleString()} <span className="text-xs font-normal text-gray-400">/день</span>
</p>
{ /* Характеристики */}

<p className="text-xs text-gray-400 mt-0.5">
{p.rooms} кімн. · {p.area} м²
{p.hasGenerator && ' · ⚡ Генератор'}
</p>
{ /* Посилання на деталі */}
<button
onClick={() => navigate(`/properties/${p.id}`)}
className="mt-2 w-full py-1.5 bg-indigo-600 hover:bg-indigo-700 text-white text-
xs font-semibold rounded-lg transition"
>
Переглянути →
</button>
</div>
</Popup>
</Marker>
))}
</MapContainer>
</div>
);
}

```

Програмний код конфігурації захищених маршрутів застосунку (App.tsx)

```
// Маршрут для авторизованих користувачів будь-якої ролі.
function PrivateRoute({ children }: { children: JSX.Element }) {
  const token = localStorage.getItem('token');
  return token ? children : <Navigate to="/login" replace />;
}

// Маршрут виключно для адміністраторів.
function AdminRoute({ children }: { children: JSX.Element }) {
  const token = localStorage.getItem('token');
  const role = localStorage.getItem('userRole');
  if (!token) return <Navigate to="/login" replace />;
  if (role !== 'Admin') return <Navigate to="/" replace />;
  return children;
}

export default function App() {
  return (
    <BrowserRouter>
      { /* UnreadProvider знаходиться всередині BrowserRouter, щоб NavChatButton
        міг використовувати useNavigate. */ }
      <UnreadProvider>
        <Routes>
          <Route path="/login" element={<Login />} />
          <Route path="/register" element={<Register />} />
          <Route
            path="/"
            element={
              <PrivateRoute>
                <Home />
              </PrivateRoute>
            }
          />
        </Routes>
      </UnreadProvider>
    </BrowserRouter>
  );
}
```

Програмний код конфігурації HTTP-клієнта з перехоплювачами (axios.ts)

```

import axios from 'axios';
const api = axios.create({
  baseURL: 'https://rentplatform-api-balenko-abf6ckhfg3g2gbeb.swedencentral-
01.azurewebsites.net/api',
  headers: { 'Content-Type': 'application/json' },
});
api.interceptors.request.use((config) => {
  const token = localStorage.getItem('token');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});
export default api;

```

					ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
						64

Програмний код кастомного хука для взаємодії з SignalR (useChat.ts)

```

import { useCallback, useEffect, useRef } from 'react';
import * as signalR from '@microsoft/signalr';
import type { MessageDto } from '../api/messageApi';
import { useUnread } from '../context/UnreadContext';
// Хук підключається до глобального SignalR-з'єднання з контексту UnreadProvider.
// Це виключає дублювання підключень при переходах між сторінками.
export function useChat(partnerId: string, onNewMessage: (msg: MessageDto) => void) {
  const { connection, isConnected } = useUnread();
  const callbackRef = useRef(onNewMessage);
  // Зберігаємо актуальний callback у ref, щоб useEffect не перезапускався при
  // кожному рендері.
  useEffect(() => {
    callbackRef.current = onNewMessage;
  }, [onNewMessage]);
  useEffect(() => {
    if (!connection) return;
    // Відображаємо лише повідомлення, що стосуються поточного діалогу.
    const handleReceive = (msg: MessageDto) => {
      if (msg.senderId === partnerId) callbackRef.current(msg);
    };
    // Підтвердження власного надісланого повідомлення від сервера.
    const handleSent = (msg: MessageDto) => {
      if (msg.receiverId === partnerId) callbackRef.current(msg);
    };
    connection.on('ReceiveMessage', handleReceive);
    connection.on('MessageSent', handleSent);
    // Знімаємо обробники при виході зі сторінки чату.
    return () => {
      connection.off('ReceiveMessage', handleReceive);
      connection.off('MessageSent', handleSent);
    };
  }, [connection, partnerId]);
  const sendMessage = useCallback(
    async (receiverId: string, content: string) => {
      if (!connection || connection.state !== signalR.HubConnectionState.Connected) {
        throw new Error("З'єднання з чатом не встановлено. Спробуйте ще раз.");
      }
    }
  );
}

```

```
await connection.invoke('SendMessage', receiverId, content);
},
[connection],
);
return { sendMessage, isConnected };;
```