

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:

«Онлайн-платформа оренди житла»

Виконала студентка групи ПЗ-21-2
ВАТИЛЬ Вікторія Валеріївна

Керівник роботи:
БАКАЛЮК Тетяна Анатоліївна

Рецензент:
ВЛАСЕНКО Олег Васильович

Житомир – 2025

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
«14» лютого 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ВАТИЛЬ Вікторія Валеріївна**

Керівник роботи: **ВАКАЛЮК Тетяна Анатоліївна**

Тема роботи: **«Онлайн-платформа оренди житла»**,

затверджена Наказом закладу вищої освіти від **«14» лютого 2025 р., №61/с**

Вихідні дані для роботи: об'єктом дослідження є інформаційні технології для реалізації оголошень та бронювання житла; предметом дослідження є використання сучасних підходів, технології та інструментів розробки веб-застосунків.

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Вакалюк Т.А.	14.02.2025р.	14.02.2025р.
2	Вакалюк Т.А.	14.03.2025р.	14.03.2025р.
3	Вакалюк Т.А.	14.04.2025р.	14.04.2025р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з веб-платформи для оренди житла та пояснювальної записки. Пояснювальна записка до випускної роботи містить текстову частину, викладену на 69 сторінках друкованого тексту, список використаних джерел містить 20 найменувань і займає 2 сторінки. В роботі наведено 44 рисунки та 14 таблиць. В роботі є 8 сторінок додатків. Загальна кількість сторінок пояснювальної записки становить 76.

Метою роботи є розробка веб-платформи з використанням сучасних інформаційних технологій, який забезпечуватиме користувачам можливість публікації оголошень про оренду житла, здійснення пошуку та бронювання об'єктів нерухомості з урахуванням зручності взаємодії, масштабованості та доступності.

У роботі визначено основні завдання, необхідні для розробки веб-платформи, здійснено аналіз існуючих веб-застосунків, аналогічних за функціональністю та представлено детальний сценарій функціонування розробленого програмного забезпечення. Наведено огляд загальної архітектури веб-застосунку, описано алгоритми взаємодії між основними компонентами системи, надано опис сценаріїв використання та об'єктної структури програмного забезпечення. Використано клієнт-серверну архітектуру, для розробки серверної частини – Next.js, а для клієнтської – React.js, у якості системи управління базами даних – PostgreSQL.

Здійснено тестування веб-застосунку з метою оцінки ефективності користувацького інтерфейсу, функціональності системи та загального досвіду взаємодії користувача з платформою.

КЛЮЧОВІ СЛОВА: ВЕБ-ПЛАТФОРМА, КЛІЄНТ-СЕРВЕР, ОRENDA ЖИТЛА, TYPESCRIPT, POSTGRESQL.

					ІПЗ.КР.Б – 121 – 25 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Онлайн-платформа оренди житла	Літ.	Арк.	Аркушів
Розроб.		Ватиль В.В.					3	76
Керівник.		Вакалюк Т.А.						
Рецензент		Власенко О.В.				Житомирська політехніка, група ІПЗ-21-2		
Зав. каф.		Вакалюк Т.А.						

ABSTRACT

A bachelor's thesis consists of a web-based housing rental platform and an explanatory note. The explanatory note to the graduation thesis contains a text part set out on 69 pages of printed text, the list of references contains 20 titles and occupies 2 pages. The work contains 44 figures and 14 tables. The paper has 8 appendices. The total number of pages of the explanatory note is 76.

The aim of the work is to develop a web platform using modern information technologies that will provide users with the ability to publish rental ads, search and book real estate objects, taking into account the convenience of interaction, scalability and accessibility.

The paper identifies the main tasks required for the development of a web platform, analyzes existing web applications that are similar in functionality, and presents a detailed scenario of the functioning of the developed software. An overview of the overall architecture of the web application is given, the algorithms of interaction between the main components of the system are described, and the use cases and object structure of the software are described. The client-server architecture was used, Next.js was used to develop the server side, and React.js was used for the client side, and PostgreSQL was used as the database management system.

The web application was tested to evaluate the effectiveness of the user interface, system functionality, and the overall user experience with the platform.

KEYWORDS: WEB PLATFORM, CLIENT-SERVER, RENTAL HOUSING, TYPESCRIPT, POSTGRESQL.

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ЗМІСТ	5
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП.....	7
РОЗДІЛ 1 АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ОНЛАЙН-ПЛАТФОРМИ ОРЕНДИ ЖИТЛА	9
1.1 Постановка задачі	9
1.2 Аналіз аналогів програмного продукту	11
1.3 Вибір архітектури для веб-орієнтованої системи для оренди житла	15
1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	18
Висновки до першого розділу	21
РОЗДІЛ 2 ПРОЕКТУВАННЯ ОНЛАЙН-ПЛАТФОРМИ ОРЕНДИ ЖИТЛА	22
2.1 Визначення варіантів використання та об'єктно-орієнтованої структури	22
2.2 Розробка бази даних системи	28
2.3 Проектування та реалізація алгоритмів роботи системи.....	34
2.4 Реалізація онлайн-платформи оренди житла	37
Висновки до другого розділу	45
РОЗДІЛ 3 ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ ОНЛАЙН-ПЛАТФОРМИ ОРЕНДИ ЖИТЛА	46
3.1 Порядок встановлення та налаштування параметрів платформи	46
3.2 Структура інтерфейсу платформи	47
3.3 Тестування платформи	62
Висновки до третього розділу	64
ВИСНОВКИ	65
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	66
ДОДАТКИ.....	68

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – програмне забезпечення

API – прикладний програмний інтерфейс

REST – передача репрезентативного стану

БД – база даних

CRUD – основні функції для управління даними (створення, читання, оновлення, видалення)

SPA – односторінковий додаток

ВСТУП

У сучасному світі, де цифрові технології стрімко розвиваються і змінюють різні сфери життєдіяльності, питання гнучкого та тимчасового проживання набуває великої важливості.

Останнім часом все більшу частину ринку займають приватні апартаменти, мікроквартири, колівінги та інші нестандартні варіації розміщення, що складають суттєву конкуренцію із традиційними готелями та орендою довгострокових помешкань. Власники таких об'єктів шукають нові шляхи залучення клієнтів, надаючи особливу увагу цифровим інструментам та платформам, що дозволяють легко бронювати житло, порівнювати пропозиції та адаптувати послуги до запитів користувачів.

Саме малий бізнес у сфері короткострокової оренди потребує ефективних і доступних цифрових сервісів простими у використанні, що забезпечує зручне керування об'єктами, бронювання без посередників та оптимізацією операційних витрат. Створення такої платформи для онлайн-оренди надає можливість для відкриття нових перспектив для власників житла, допомагаючи залишатись конкурентоспроможними для широкої аудиторії.

Актуальність теми зумовлена динамічним розвитком цифрових технологій та зростанням попиту на альтернативні форми тимчасового проживання. В умовах глобалізації та урбанізації платформи змінюють традиційний ринок нерухомості, пропонуючи гнучкість, доступність та економічну вигоду для різних соціальних груп.

Важливість даної теми спричинена не лише стрімким розвитком інформаційних технологій, а й зміною споживчих тенденцій. Користувачі все частіше надають перевагу швидким і зручним онлайн-платформам, що дають можливість самостійно обирати житло, порівнювати варіанти, ціни, умови та відгуки, контролюючи кожен етап бронювання. Це дозволяє спростити процес пошуку тимчасового розміщення, заощадити кошти та уникнути зайвих комісій.

Крім того, останні глобальні зміни підкреслюють необхідність та важливість адаптивних та незалежних платформ, що будуть оперативно реагувати на нові

умови ринку. У цих реаліях цифрові сервіси стають не лише інструментом для користувачів, а й підтримкою малого бізнесу, що допомагає місцевим власникам залишатись конкурентоздатними.

Таким чином, розробка онлайн-платформи оренди житла відповідає актуальним потребам як орендодавців, так і користувачів. Це рішення сприятиме підвищенню ефективності ринку нерухомості та розвитку цифрового середовища.

Метою випускної кваліфікаційної роботи бакалавра є розробка онлайн-платформи оренди житла у вигляді веб-застосунку. Сервіс має забезпечувати безперебійну роботу, високу стійкість до перевантажень, кросплатформеність на різних типах пристроїв та відповідати вимогам ефективності та зручності у використанні.

Поставлена мета обумовлює необхідність виконання наступних завдань:

1. Аналіз теоретичних і функціональних аспектів проєктування веб-платформ.
2. Аналіз існуючих веб-платформ за обраною тематикою з метою виявлення їх функціональних особливостей та визначення унікальності власної розроблюваної платформи.
3. Вибір технологічних засобів для реалізації веб-платформи.
4. Проєктування бази даних для збереження інформації та відображення її на платформі.
5. Опис методів взаємодії з веб-системою.
6. Визначення об'єктно-орієнтованої структури веб-платформи.

Об'єктом дослідження є інформаційні технології, що використовуються для розробки та функціонування веб-застосунку.

Предметом дослідження є використання сучасних методів і технологій у розробці веб-застосунків та їх взаємодія між компонентами системи.

РОЗДІЛ 1 АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ОНЛАЙН-ПЛАТФОРМИ ОРЕНДИ ЖИТЛА

1.1 Постановка задачі

У сучасних умовах цифрової трансформації ринку нерухомості онлайн-платформи оренди житла забезпечують новий рівень доступності та прозорості, поєднуючи інструменти пошуку, бронювання та управління об'єктами нерухомості в одному єдиному інтерфейсі. Зростання мобільності населення та популяризація спільного (sharing) підходу до користування активами створюють підґрунтя для широкого впровадження подібних платформ, які об'єднують орендодавців і орендарів у єдиному інформаційному просторі.

Завдяки інтеграції геолокаційних сервісів, алгоритмів рекомендацій і платежів у реальному часі, користувачі отримують можливість оперативно знаходити житло, яке відповідає їхнім індивідуальним потребам, та здійснювати транзакції безпосередньо через веб-інтерфейс або мобільний додаток.

Більше того, сучасні платформи оренди житла виступають своєрідними цифровими спільнотами, які формують клієнтську екосистему із відгуків, рейтингів та рекомендацій. Використання функціональних модулів для зворотного зв'язку й обміну досвідом створює додаткову цінність як для власників нерухомості, так і для орендарів, сприяючи довгостроковому співробітництву та лояльності.

Метою розробки є створення інтерактивної веб-платформи оренди житла, що забезпечуватиме зручний пошук, бронювання та управління об'єктами нерухомості.

Для досягнення поставленої мети потрібно виконати такі завдання.

Спочатку потрібно спроектувати архітектуру веб-платформи, визначити оптимальний технологічний стек, що забезпечить масштабованість, продуктивність і безпеку системи та обрати відповідний фреймворк для швидкої розробки та полегшення підтримки та якості коду.

Далі спроектувати та розробити структуру бази даних: створити таблиці для зберігання інформації про нерухомості, оголошення, бронювання, відгуки, користувачів і т.п., впровадження міжтабличних зв'язків для забезпечення цілісності та ефективного доступу до даних.

Наступним етапом стане реалізація функціональності та можливості веб-платформи для відображення та збереження різної інформації.

Важливим моментом розробки є створення інтерфейсу користувача для виконання різних операцій на платформі. Адміністратор отримає інструменти для моніторингу та керування всіма оголошеннями й користувачами для оперативного вирішення організаційних і безпекових питань, модератор отримає спеціалізовану панель для перегляду, верифікації та редагування оголошень, що гарантує відповідність контенту внутрішнім політикам, орендодавець матиме зручний інтерфейс для реєстрації та редагування об'єктів нерухомості, публікації оголошень та перегляду історії бронювань, а орендатор – інтуїтивно зрозумілий адаптивний інтерфейс для пошуку та фільтрації пропозицій, оформлення бронювання і управління особистим кабінетом.

Далі здійснюватиметься тестування функціональності розробленої веб-платформи та виправлення помилок, які будуть виявлені, проведення навантажувального та стрес-тестування для оцінки продуктивності й відмовостійкості системи під різними сценаріями експлуатації, вдосконалення роботи платформи.

Також важливе значення матимуть розгортання платформи на сервері та запуск його в експлуатацію, передбачення постійної підтримки платформи шляхом регулярних оновлень функціональних модулів і компонентів безпеки, оптимізація інфраструктури під зростаюче навантаження та впровадження поліпшень на основі зворотного зв'язку користувачів з метою забезпечення стабільності роботи та актуальності платформи.

Результатом реалізації поставлених завдань стане створення веб-платформи оренди житла, яка забезпечить користувачам ефективний пошук, бронювання та управління нерухомістю через інтуїтивно зрозумілий інтерфейс.

1.2 Аналіз аналогів програмного продукту

У сегменті онлайн-платформ оренди житла вже функціонує значна кількість веб-платформ, що надають користувачам можливість пошуку, бронювання та управління об'єктами нерухомості.

Для аналізу найефективніших онлайн-платформ оренди житла обрано такі сервіси: DimRia, Flatfy, Booking.com та Airbnb .

DimRia (стара назва – DomRia) [1] – це один із найпопулярніших українських онлайн-сервісів для пошуку та оренди нерухомості, який надає користувачам можливість переглядати оголошення про здачу квартир, будинків та комерційної нерухомості по всій території України. Сервіс пропонує зручні фільтри для пошуку за параметрами (тип житла, ціна, розташування, тощо), інтерактивну карту, перевірку оголошень, окремі інструменти для власників і ріелторів. Інтерфейс DimRia представлено на рис. 1.1.

До переваг DimRia можна віднести зручний інтерфейс, широке покриття ринку, функцію перевірених оголошень та наявність мобільного застосунку для доступу до сервісу з різних пристроїв.

До недоліків DimRia можна віднести наявність платних послуг для просування оголошень, обмежений функціонал для користувачів без реєстрації.

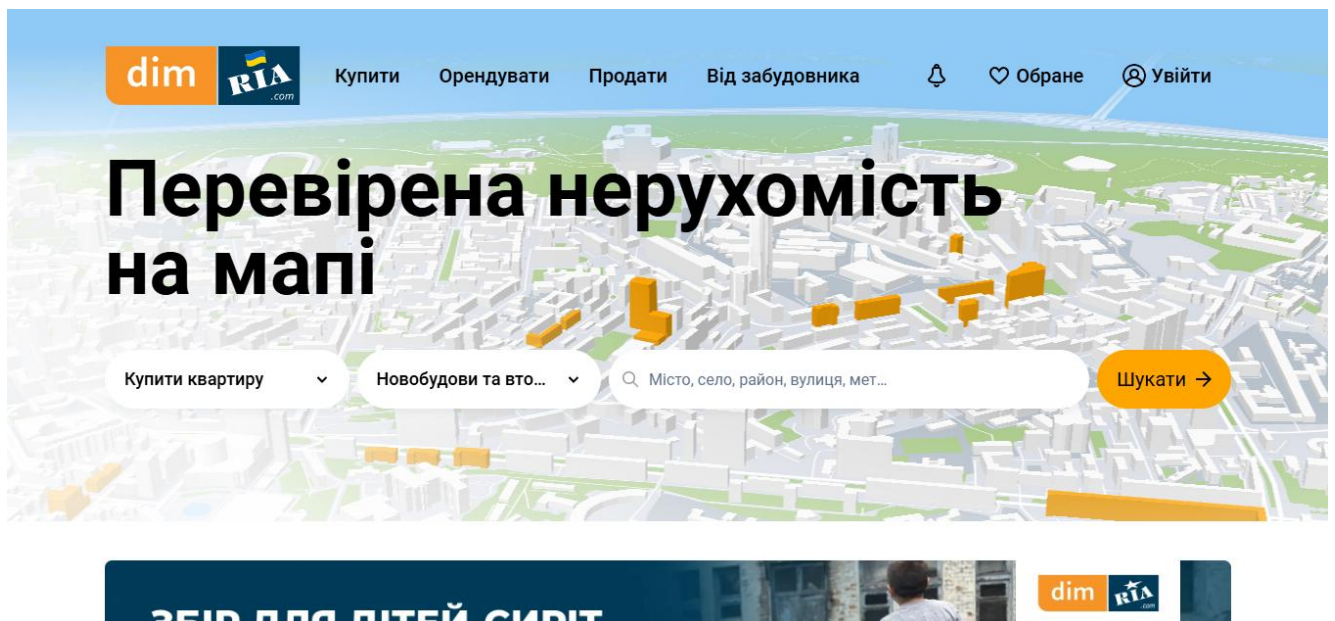


Рисунок 1.1 – Інтерфейс DimRia

Flatfy [2] – це міжнародна платформа для пошуку нерухомості, яка працює в понад 30 країнах, в тому числі в Україні. Сервіс агрегує оголошення з різних джерел і надає користувачам змогу здійснювати зручний пошук житла для оренди або купівлі. Flatfy має простий та зрозумілий інтерфейс, дозволяє фільтрувати результати за розташуванням, типом нерухомості, вартістю, кількістю кімнат та іншими параметрами. Інтерфейс Flatfy показаний на рис. 1.2.

До переваг Flatfy можна віднести великий доступ до великої кількості оголошень, автоматичну перевірку на дублікати, підтримку кількох мов та інтуїтивну навігацію по сайту.

До недоліків Flatfy належить обмежений набір функцій для власників житла, відсутність повноцінного мобільного додатку для деяких платформ та залежність якості оголошень від сторонніх сайтів-джерел.

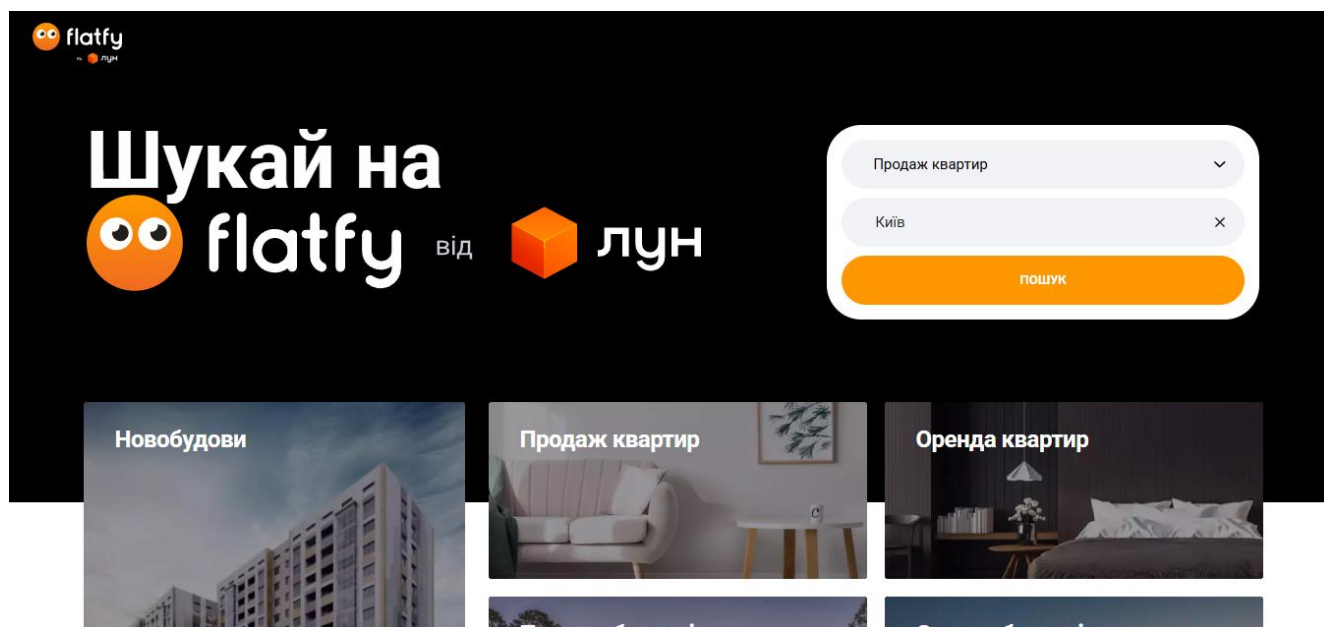


Рисунок 1.2 – Інтерфейс Flatfy

Booking.com [3] – міжнародний онлайн-сервіс для бронювання житла, який має широкий спектр об'єктів – від готелів і апартаментів до приватних будинків та хостелів. Платформа доступна у понад 220 країнах світу включно з Україною. Користувачі можуть здійснювати пошук за різноманітними параметрами, переглядати фотографії, читати відгуки, порівнювати ціни та бронювати житло безпосередньо через веб-сайт або мобільний застосунок. Інтерфейс Booking.com показаний на рис. 1.3.

До переваг Booking.com належать велика база об'єктів, наявність детальної інформації про кожне житло, підтримка багатьох мов, можливість безкоштовного скасування бронювання та високий рівень довіри користувачів.

До недоліків Booking.com можна віднести високі комісії для власників житла, складність у видимості нових оголошень серед великої кількості пропозицій, можливі випадки неактуальності інформації у деяких оголошеннях.

Рисунок 1.3 – Інтерфейс Booking.com

Airbnb [4] – це світова платформа для оренди житла, яка з'єднує орендодавців із орендарями в понад 190 країнах світу, включаючи Україну. Сервіс дозволяє розміщувати та бронювати найрізноманітніші типи помешкань – від кімнат і квартир до будинків, вілл і унікальних об'єктів (човни, будинки на деревах). Користувачі можуть ознайомитися з описом житла, переглянути фото, прочитати відгуки та забронювати помешкання через веб-сайт або мобільний застосунок. Інтерфейс Airbnb показаний на рис. 1.4.

До переваг Airbnb можна віднести зручний і сучасний інтерфейс, можливість спілкування між орендодавцем і орендарем через чат, широкий вибір житла, наявність рейтингової системи довіри.

До недоліків Airbnb належать сервісні збори, які в деяких випадках підвищують загальну вартість бронювання, потенційні ризики у разі порушення правил з боку користувачів, обмежена підтримка в окремих регіонах.

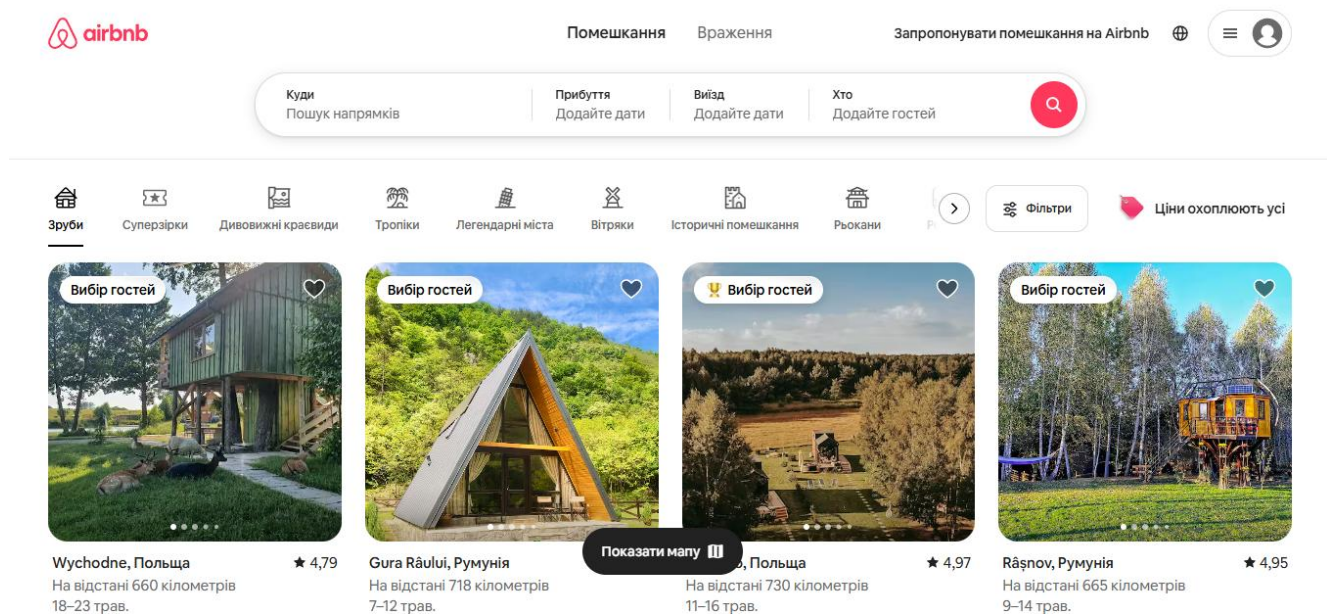


Рисунок 1.4 – Інтерфейс Airbnb

Для наочності порівняння отримані результати подані в табл. 1.1.

Таблиця 1.1

Порівняльна характеристика аналогів платформ

Назва платформи	Зручність інтерфейсу	Широта бази оголошень	Наявність перевірених оголошень	Можливість спілкування	Комісійна політика
DimRia	+	+	+	-	+/-
Flatfy	+	+	+/-	-	+
Booking.com	+	+	+/-	+	-
Airbnb	+	+	+	+	-

Після аналізу функціональних можливостей онлайн платформ оренди житла можна визначити основні вимоги до розроблювальної платформи: можливість додавати оголошення та нерухомості, система фільтрації та пошуку за параметрами, карта для візуалізації об'єктів нерухомості, адаптивний інтерфейс для різних пристроїв, можливість реєстрації та автентифікації користувачів, перегляд та модерація відгуків, блокування користувачів, бронювання нерухомості, можливість залишати відгуки для орендарів, керування особистою інформацією, редагування профілю.

1.3 Вибір архітектури для веб-орієнтованої системи для оренди житла

Вибір архітектури веборієнтованої системи є важливим етапом проектування програмного забезпечення, оскільки архітектурні рішення визначають логічну організацію системи, спосіб взаємодії її компонентів, можливості масштабування, підтримуваності та ефективності подальшого розвитку.

Архітектура повинна враховувати специфіку предметної області, вимоги до продуктивності, безпеки, адаптивності інтерфейсу та інтеграції з зовнішніми сервісами. Успішно обрана архітектурна забезпечує стабільну роботу системи у реальному часі, спрощує процес її модифікації, тестування та розгортання.

Якщо коротко, то архітектура є фундаментом, на якому базується вся логіка функціонування платформи.

Перед тим, як обрати правильну архітектуру, розглянемо найпопулярніші з них.

Монолітна архітектура – найдавніша та найпоширеніша архітектура програмного забезпечення у сфері веб-орієнтованих систем. У межах цієї архітектурної моделі всі компоненти системи реалізуються як єдиний цілісний додаток, який функціонує як один процес.

Типовим прикладом є веб-застосунок, що містить у собі користувацький інтерфейс, логіку бізнес-процесів, доступ до бази даних та інші функціональні модулі в межах єдиної кодової бази.

Основною перевагою монолітної архітектури є її відносна простота реалізації та розгортання. Централізована структура полегшує початкову розробку системи, забезпечує узгодженість між компонентами, спрощує налагодження та тестування. Цей підхід часто обирають на етапі створення прототипу або для невеликих проєктів, де немає потреби в складному масштабуванні або багаторівневій взаємодії між модулями.

Зі зростанням обсягу функціональності та кількості користувачів монолітна архітектура поступово втрачає свою ефективність. Ускладнюється процес внесення змін до окремих частин системи без впливу на інші модулі, виникають труднощі з масштабуванням окремих підсистем, збільшується ризик помилок при розгортанні

оновлень. Крім того, монолітні системи в деяких випадках обмежують гнучкість розробки, особливо у разі паралельної роботи над проєктом кількох команд розробників.

Клієнт-серверна архітектура є базою і водночас найпоширенішим підходом до побудови веборієнтованих систем. У цій моделі застосунок розподіляється на дві основні частини – клієнтську та серверну.

Клієнтська частина, її ще називають фронтендом (frontend), відповідає за забезпечення інтерфейсу взаємодії з кінцевим користувачем. Вона виконується безпосередньо на пристрої користувача – у веб-браузері або мобільному застосунку – і реалізує логіку відображення, обробку введення, валідацію форм, навігацію між сторінками, тощо. Клієнт ініціює запити до сервера, надсилаючи дані у стандартизованому вигляді (наприклад, у форматі JSON) через протокол HTTP або HTTPS. Сучасні фронтед-технології, такі як React, Angular, Vue, Next.js, дозволяють створювати динамічні, адаптивні та високопродуктивні інтерфейси.

Серверна частина або бекенд (backend) виконує функції обробки бізнес-логіки, автентифікації користувачів, авторизації доступу, управління базами даних і взаємодії з іншими сервісами. Саме тут реалізується логіка, яка визначає правила взаємодії між елементами системи, обробку транзакцій, формування відповідей на клієнтські запити. Сервер може бути побудований з використанням різноманітних мов і фреймворків (наприклад, Node.js, ASP.NET, Django), може бути масштабований горизонтально або вертикально залежно від потреб системи. Надійність і продуктивність серверної частини є важливим забезпеченням стабільної роботи веб-застосунку, особливо в умовах високого навантаження.

Перевагою клієнт-серверної архітектури є чіткий розподіл відповідальності між клієнтом і сервером, що забезпечує структурованість системи та спрощує її обслуговування. Можливість створювати масштабовані й продуктивні системи, де сервер може централізовано обробляти запити великої кількості клієнтів, забезпечуючи водночас єдину логіку доступу до даних.

Недоліком клієнт-серверної архітектури є залежність від стабільного з'єднання з сервером та обмежену автономність клієнтської частини. У разі

високого навантаження на сервер або його недоступності, функціональність усієї системи може бути порушена, є загроза у відкритих мережах, якщо передача даних відбувається без шифрування.

Мікросервісна архітектура – поширена модель, яка відзначається високим рівнем модульності та гнучкості. Архітектура системи складається з набору незалежних сервісів, кожен з яких відповідає за окрему бізнес-функцію та функціонує як окремий процес. Кожен мікросервіс має власну логіку, базу даних або доступ до неї, інтерфейс взаємодії (REST або gRPC API) і може бути розгорнутий, масштабований та оновлений незалежно від інших компонентів системи.

Основною перевагою мікросервісної архітектури є високий рівень масштабованості, стійкість до збоїв, можливість незалежної розробки та оновлення окремих частин системи, особливо для великих, динамічних або багатофункціональних проєктів. Розробники можуть зосередитися на окремих модулях, забезпечуючи більш ефективне управління складністю системи та пришвидшуючи цикл розгортання нових функцій.

Мікросервісна архітектура має декілька недоліків. При розгортанні чи підтримці інфраструктури зростає її складність, з'являється необхідність у системах моніторингу, логування, управління конфігураціями та забезпечення взаємодії між сервісами, ускладнюється налагодження взаємозв'язків та забезпечення консистентності даних між розподіленими компонентами.

Остаточний вибір архітектури було здійснено на основі аналізу функціональних і нефункціональних вимог платформи, з урахуванням принципів масштабованості, гнучкості, розширюваності та підтримки. У результаті обґрунтованим рішенням стало використання клієнт-серверної архітектури.

Як вже було проаналізовано, архітектура розмежовує обов'язки між клієнтською частиною, яка відповідає за взаємодію з користувачем, та серверною частиною, яка обробляє запити, реалізує бізнес-логіку та забезпечує доступ до бази даних. В цьому випадку, розробка платформи буде організованою, полегшить

підтримку коду, дасть змогу масштабувати окремі компоненти незалежно та адаптувати систему до змінних навантажень.

Клієнт-серверна архітектура забезпечує достатній рівень безпеки, дає змогу ефективно реалізовувати механізми автентифікації, авторизації, спрощує реалізацію веб-інтерфейсу, що є важливим для користувацького досвіду в системах подібного типу. Вона підтримується сучасними технологіями з великою наявністю інструментів для розробки, що являється найбільш доцільним для реалізації поставлених завдань.

1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Враховуючи специфіку проєкту, сформовані завдання та обрану архітектуру потрібно обрати інструментальні засоби для ефективної реалізації основних складових веборієнтованого застосунку. Оскільки платформа є веборієнтованою було обрано наступні сучасні інструменти.

Для реалізації серверної та клієнтської частин системи було обрано кросплатформене середовище виконання, базоване на мові програмування JavaScript – Node.js [4].

Node.js – середовище виконання JavaScript, побудоване на рушії V8 від Google, який безпосередньо компілює вихідний код у машинні інструкції для досягнення високої продуктивності. Платформа організована навколо одного поточного циклу обробки подій із неблокуючою моделлю вводу-виводу, що дозволяє ефективно керувати численними одночасними з'єднаннями. Структура модулів CommonJS забезпечує розподіл коду на ізольовані блоки, інстальовані й оновлювальні через менеджер пакетів npm. Екосистема npm пропонує широкий вибір бібліотек і утиліт із відкритим вихідним кодом, вбудовані API надають інструменти для роботи з файловою системою, мережевими протоколами та потоками даних.

Для виконання етапу кодування було обрано мову програмування TypeScript [5].

TypeScript – мова програмування із статичною типізацією та розширеним синтаксисом JavaScript, яка виконує транспіляцію в JavaScript сучасних версій ECMAScript. Статична типізація дозволяє виявляти помилки ще до етапу компіляції, тим самим підвищуючи зрозумілість коду. Підтримка інтерфейсів, класів і модулів полегшує організовувати велику кодову базу. Інтеграція з редакторами коду гарантує автодоповнення й статичний аналіз під час розробки, деклараційні файли забезпечують сумісність із зовнішніми бібліотеками. Є можливість поступово впроваджувати типізацію в існуючі проекти JavaScript без втрати функціональності.

У якості системи збереження даних обрано реляційна СУБД PostgreSQL [6]. Система підтримує стандартизовану мову SQL із розширеннями для складних аналітичних та транзакційних запитів, забезпечує повноцінну реалізацію ACID-транзакцій і високий рівень безпеки. Наявні механізми реплікації та розподіленого зберігання даних для масштабованості та відмовостійкості.

Для організації шару доступу до даних з серверної частини обрано Prisma ORM [7]. Prisma ORM виконує генерацію типобезпечних моделей із вихідного опису схеми бази даних, спрощує формулювання складних SQL-запитів через декларативний API та підтримує автоматичну міграцію структури даних відповідно до визначених моделей.

Для реалізації клієнтської частини веб-орієнтованої системи обрано фреймворк Next.js [8].

Next.js представляє собою фреймворк для побудови веб-застосунків на базі React, розроблений компанією Vercel з урахуванням сучасних вимог до продуктивності та SEO. Основним призначенням фреймворку вважається забезпечення можливості виконання серверного рендерингу й попередньої генерації сторінок, завдяки чому час початкового завантаження значно скорочується, а контент стає доступним пошуковим ботам.

Архітектурною основою Next.js є бібліотека React.js [9] – технологія, яка пропонує декларативний компонентний підхід і віртуальний DOM для оптимізації оновлення інтерфейсу користувача. React-компоненти в межах Next.js можуть

використовувати сучасні можливості JavaScript (ES6+), а їхня взаємодія опрацьовується через чітко визначені життєві цикли.

Механізм серверного рендерингу (SSR) оброблює React-компоненти на сервері під час отримання запиту, після чого відправляється готовий HTML. Альтернативою слугує генерація статичних сторінок (SSG), під час якої HTML створюється під час збірки проєкту. Обидва методи знижують затримки при навігації та підвищують оцінки сторінок пошуковими системами.

Система маршрутизації у Next.js базується на файловій структурі директорій – файли в папці `pages` відповідають URL-шляхам, динамічні сегменти позначаються через квадратні дужки в імені файлу, організація маршрутів є інтуїтивно зрозумілою, додавання нових ендпоінтів не потребує додаткових налаштувань.

API-шари реалізуються шляхом створення серверних функцій у межах директорії `pages/api`, які обробляють HTTP-запити подібно до традиційних бекенд-ендпоінтів, що спрощує розробку middleware-логіки і зберігає єдину структуру проєкту.

Для візуалізації геопросторових даних обрана бібліотека React Leaflet [10], яка забезпечує інтеграцію картографічного движка Leaflet у середовище React. Бібліотека надає набір компонентів для відображення маркерів, шарів і маршрутів на інтерактивній карті, дозволяє точно позиціонувати різні об'єкти за координатами. Модульна архітектура React Leaflet надає змогу додавати плагіни та налаштовувати зовнішній вигляд картографічних елементів із урахуванням специфіки проєкту.

Обробка платежів здійснюється за допомогою платіжної системи LiqPay [11], яка підтримує інтеграцію через RESTful API та відповідає вимогам безпеки платіжних даних згідно з стандартом PCI DSS. Інтерфейсні модулі для взаємодії із LiqPay реалізують генерацію платіжних форм, обробку callback-повідомлень і верифікацію цифрових підписів гарантуючи цілісність і конфіденційність фінансових операцій користувачів.

Для стилізації інтерфейсу обрано утилітарний CSS-фреймворк Tailwind CSS [12], який забезпечує швидке створення адаптивних макетів за допомогою набору готових класів, інтеграцію з PostCSS та можливість гнучкого налаштування через конфігураційний файл.

Для розгортання веб-орієнтованої системи обрана платформа Vercel [13]. Vercel має вбудований конвеєр безперервної інтеграції та доставки, автоматичному масштабуванню та оптимізації швидкості завантаження ресурсів. Інтеграція з репозиторієм на GitHub забезпечує автоматичну побудову та розгортання при кожному коміті, а глобальна мережева доставка контенту (CDN) гарантує мінімальні затримки для кінцевих користувачів.

Система повинна гарантувати оптимальну роботу на різних пристроях, таких як комп'ютери, смартфони, планшети та інші. Тому мінімальні вимоги для браузерів є наступними:

- 4024 Мб оперативної пам'яті;
- 300 Мб вільного дискового простору;
- Процесор з тактовою частотою 1GHz;
- Операційна система Windows 10, Mac OS X 11.5, Ubuntu 18.04, Android 11, IOS 18.

Висновки до першого розділу

Розглянуто функціональні можливості та обмеження провідних онлайн-платформ оренди житла, після чого сформульовано перелік ключових вимог до розроблювальної.

Було обрано клієнт-серверну архітектуру з RESTful API та MVC-патерном для чіткого розподілу відповідальностей між шарами системи. Інструментарій реалізації включає: серверну частину на Node.js із TypeScript, Prisma ORM та PostgreSQL для збереження даних; клієнтську частину на Next.js (React) з Tailwind CSS для побудови адаптивного інтерфейсу; бібліотеку React Leaflet для картографічної візуалізації; сервіс LiqPay для обробки платежів; платформу Vercel для автоматизованого розгортання та CDN-доставки контенту.

РОЗДІЛ 2 ПРОЕКТУВАННЯ ОНЛАЙН-ПЛАТФОРМИ ОРЕНДИ ЖИТЛА

2.1 Визначення варіантів використання та об'єктно-орієнтованої структури

Для забезпечення реалізації ключових функціональних можливостей веборієнтованої системи оренди житла та її відповідності базовим вимогам користувачів і технічним умовам, сформульовано узагальнені сценарії використання, які визначають типову поведінку системи при взаємодії з різними категоріями користувачів. Такі сценарії дозволяють структурувати функціональність, виявити основні об'єкти предметної області та встановити їхні зв'язки. На основі цього буде побудовано об'єктно-орієнтовану модель, яка відображає логіку системи та слугуватиме фундаментом для подальшої реалізації її компонентів.

Вимоги користувачів:

1. Можливість переглядати, шукати, фільтрувати та сортувати оголошення;
2. Можливість переглядати оголошення та відгуки;
3. Можливість бронювати нерухомість з різними параметрами;
4. Можливість редагувати особисті дані користувачем;
5. Можливість перегляду власних бронювань користувачем та орендаром;
6. Можливість зміни та відновлення паролю;
7. Можливість створювати нерухомості та оголошення, їх перегляд;
8. Можливість редагування нерухомостей та оголошень;
9. Можливість модерації оголошень адміністратором та модератором;
10. Можливість блокування та зміни ролі адміністратором.

Для кращого сприйняття інформації основні функціональні та нефункціональні вимоги було представлено у табл. 2.1 - 2.2.

Таблиця 2.1

Основні функціональні вимоги

Визначення	Опис
Авторизація у системі	В системі має бути реалізований функціонал створення облікового запису, що забезпечує реєстрацію нових користувачів та подальший доступ до можливостей платформи;
Збереження та редагування даних	Система має забезпечувати збереження даних з можливістю їх перегляду, редагування та управління у межах функціоналу.

Таблиця 2.2

Основні нефункціональні вимоги

Визначення	Опис
Сприйняття	Час, необхідний для ознайомлення звичайних користувачів із функціональністю системи – 10-15 хвилин, для досвідчених користувачів – 5-8 хвилин;
	Час відгуку для типових задач – не більше 20 секунд, для складних завдань – не більше 30 секунд
	Інтерфейс повинен бути інтуїтивно зрозумілим, із наявністю контекстних підказок
Надійність	Середній час безвідмовної роботи – 15 робочих днів;
	Максимальна норма помилок та дефектів в роботі системи – 1 помилка на десять тисяч рядків коду
Доступність	Час, що витрачається на обслуговування системи не повинно перевищувати 5% від загального часу роботи
Продуктивність	Система повинна підтримувати мінімум 100 одночасно працюючих користувачів, пов'язаних із загальною базою даних.

Масштабування	Система повинна мати можливість збільшувати потужності зі збільшенням користувачів без погіршення швидкодії, стабільності або доступності функціональних можливостей.
---------------	---

Аналіз функціональних потреб та очікувань кінцевих користувачів став підґрунтям для визначення варіантів використання онлайн-платформи оренди житла. На основі цих даних було сформовано діаграму варіантів використання, яка відображає взаємодію з основним функціоналом веборієнтованої системи залежно від ролі користувача (рис. 2.1). У системі передбачено такі ролі: неавторизований користувач, орендатор, орендодавець, модератор та адміністратор. Кожна з ролей має доступ до певного набору функцій, відповідно до своїх повноважень та завдань у межах роботи з системою.

Неавторизований користувач має доступ до функціоналу реєстрації та авторизації, відновлення паролю, можливість пошуку, сортування та фільтрування оголошень, їх перегляд.

Орендатор має доступ до функціональних можливостей, аналогічних тим, що доступні неавторизованому користувачу, проте функціональність його взаємодії із системою є значно розширеною. Він отримує доступ до процесу бронювання об'єктів нерухомості, може редагувати власні персональні дані, переглядати список своїх бронювань із можливістю їх скасування або здійснення оплати, змінювати пароль до особистого облікового запису.

Орендодавець має доступ до функціоналу, дозволяючи створювати об'єкти нерухомості та публікувати відповідні оголошення, редагувати та видаляти власні об'єкти нерухомості та пов'язані з ними оголошення, можливість переглядати бронювання, здійснені користувачами щодо його об'єктів, з подальшою можливістю підтвердження, скасування або фіксації факту оплати у формі готівкового розрахунку.

Модератор має обмежену кількість функцій, основною з яких є огляд оголошень, які ще не були опубліковані, та перебувають на стадії перевірки. Йому

надається можливість підтвердити публікацію оголошення або відхилити, залежно від відповідності встановленим вимогам та політикам платформи.

Адміністратор має ті дії, що й модератор, але з додатковою функцією зміни ролі та блокування користувачів для ефективного управління доступом до системи та забезпечення її безпеки.

У результаті аналізу функціональних вимог веборієнтованої системи для оренди житла було визначено сутності, взаємозв'язки між ними та відповідні атрибути, які забезпечують повноцінну реалізацію основного функціоналу платформи. На основі отриманих результатів побудовано діаграму класів (рис. 2.2), яка відображає структуру об'єктно-орієнтованої моделі системи. Дана діаграма слугує основою для подальшої реалізації бізнес-логіки та взаємодії компонентів системи.



Рисунок 2.1 – Діаграма варіантів використання платформи

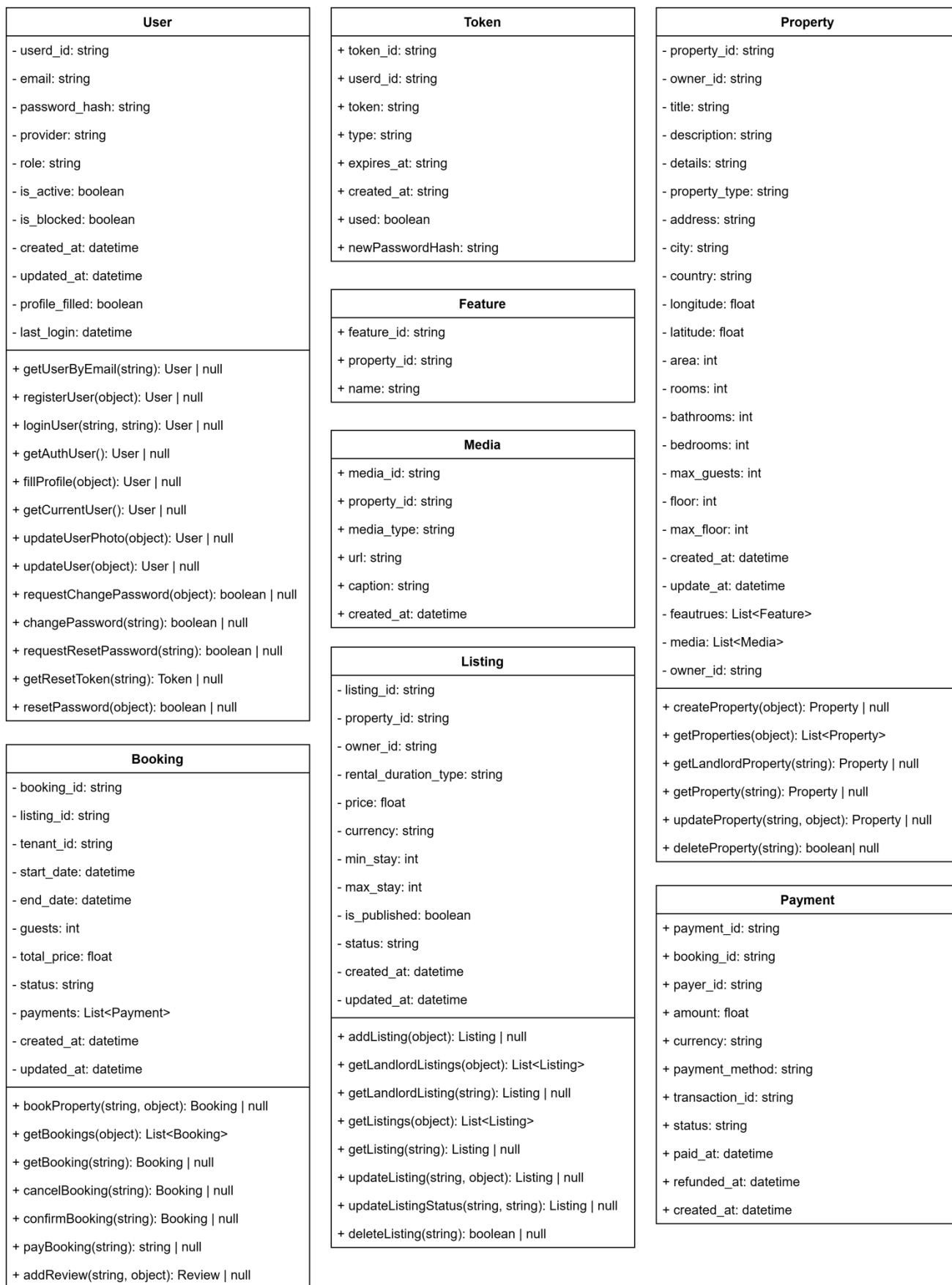


Рисунок 2.2 – Діаграма класів

У класі користувачів User getUserByEmail() відповідає за отримання користувача по електронній пошті. registerUser() відповідає за реєстрацію користувача, loginUser() – за авторизацію користувача, а getAuthUser () відповідає за отримання авторизованого користувача. fillProfile() відповідає за заповнення профіля користувача, getCurrentUser() відповідає за отримання поточного користувача. updateUserPhoto() оновлює фото користувача, а updateUser() – дані користувача. requestChangePassword() потрібна для запиту зміни пароллю, в той час як changePassword() відповідає за безпосередню зміну пароллю; requestResetPassword() ініціює запит скидання пароллю, getResetToken() отримує токен скидання, а resetPassword() відповідає за скидання пароллю.

Property – це клас нерухомості. createProperty() створює нерухомість, getProperties() відповідає за отримання списку доступних нерухомостей, а getLandlordProperty() – за отримання нерухомості конкретного орендодавця. getProperty() відповідає за отримання інформації про нерухомість, updateProperty() оновлює дані нерухомості, а deleteProperty() відповідає за видалення її видалення.

У класі оголошень Listing. addListing() відповідає за додавання оголошення, getLandlordListings() – за отримання оголошень конкретного орендодавця, getLandlordListing() – за отримання одного конкретного оголошення даного орендодавця. getListings() надає загальний список усіх оголошень, в той час як getListing() відповідає за отримання одного певного оголошення. updateListing() відповідає за оновлення оголошення, updateListingStatus() оновлює статус оголошення, а deleteListing() – видаляє оголошення.

Booking – це клас бронювань. bookProperty() виконує бронювання оголошення. getBookings() відповідає за отримання загального списку бронювань, а getBooking() відповідає за отримання одного бронювання. cancelBooking() скасовує бронювання, а confirmBooking() – підтверджує. payBooking() відповідає за оплату бронювання, в той час як addReview() відповідає за додавання відгуку про бронювання.

2.2 Розробка бази даних системи

Для збереження даних, необхідних для функціонування веб-платформи з оренди житла, використовується PostgreSQL – реляційна система управління базами даних, яка підтримує розширені можливості роботи з транзакціями, складними зв'язками між таблицями з високим рівнем надійності зберігання, реалізовує чітко визначені зв'язки між об'єктами через структуровану схему.

База даних платформи складається з 10 таблиць:

1. User – таблиця, у якій зберігається інформація про користувачів системи;
 2. UserProfile – таблиця, у якій зберігається детальна інформація про користувачів;
 3. Property – таблиця, у якій зберігається інформація про нерухомість;
 4. Listing – таблиця, у якій зберігається інформація про оголошення;
 5. Media – таблиця, у якій зберігається інформація про медіа-файли нерухомості;
 6. Feature – таблиця, у якій зберігається інформація про зручності нерухомості;
 7. Token – таблиця, у якій зберігається інформація про токени користувачів;
 8. Booking – таблиця, у якій зберігається інформація про бронювання;
 9. Payment – таблиця, у якій зберігається інформація про платежі;
 10. Review – таблиця, у якій зберігається інформація про відгуки;
- Опис кожної з таблиці бази даних наведений у табл. 2.1 - 2.10.

Таблиця 2.3

Опис таблиці User

Поле	Тип даних	Опис
user_id	uuid	Ідентифікатор користувача
email	varchar	Електронна пошта
password_hash	varchar(255)	Хешований пароль
provider	varchar(30)	Провайдер
role	varchar(30)	Роль користувача

продовж. табл. 2.3

is_active	boolean	Активний користувач
is_blocked	boolean	Заблокований користувач
created_at	datetime	Дата створення користувача
profile_filled	boolean	Заповнено частково профіль
last_login	datetime	Дата останнього входу користувача

Таблиця 2.4

Опис таблиці UserProfile

Поле	Тип даних	Опис
profile_id	uuid	Ідентифікатор профілю
user_id	uuid	Ідентифікатор користувача
first_name	varchar(50)	Ім'я користувача
last_name	varchar(50)	Прізвище користувача
phone	varchar(20)	Телефон користувача
show_phone	boolean	Показувати телефон
avatar_url	varchar	Аватар користувач
created_at	datetime	Дата створення профілю
update_at	datetime	Дата оновлення профілю

Таблиця 2.5

Опис таблиці Property

Поле	Тип даних	Опис
property_id	uuid	Ідентифікатор нерухомості
owner_id	uuid	Ідентифікатор власника
title	varchar(255)	Назва нерухомості
description	text	Опис нерухомості
details	text	Деталі нерухомості
property_type	varchar(30)	Тип нерухомості
address	varchar(255)	Адреса нерухомості
city	varchar(100)	Місто нерухомості

продовж. табл. 2.5

country	varchar(100)	Країна нерухомості
longitude	float	Розташування нерухомості (довгота)
latitude	float	Розташування нерухомості (широта)
area	int	Площа нерухомості
rooms	int	Кількість кімнат
bathrooms	int	Кількість ванних кімнат
bedrooms	int	Кількість спалень
max_guests	int	Кількість гостей
floor	int	Поверх нерухомості
max_floor	int	Максимальний поверх розташування
created_at	datetime	Дата додавання нерухомості
updated_at	datetime	Дата оновлення нерухомості

Таблиця 2.6

Опис таблиці Listing

Поле	Тип даних	Опис
listing_id	uuid	Ідентифікатор оголошення
property_id	uuid	Ідентифікатор нерухомості
owner_id	uuid	Ідентифікатор власника
rental_duration_type	varchar(30)	Тип оренди
price	float	Ціна оренди
currency	varchar(30)	Валюта
min_stay	int	Мінімальна кількість перебування
max_stay	int	Максимальна кількість перебування
is_published	boolean	Опубліковано оголошення
status	varchar(30)	Статус оголошення
created_at	datetime	Дата додавання оголошення
updated_at	datetime	Дата оновлення оголошення

Таблиця 2.7

Опис таблиці Media

Поле	Тип даних	Опис
media_id	uuid	Ідентифікатор медіа
property_id	uuid	Ідентифікатор нерухомості
media_type	varchar(20)	Тип медіа
url	varchar	Посилання на ресурс
caption	varchar(255)	Заголовок медіа
created_at	datetime	Дата додавання медіа

Таблиця 2.8

Опис таблиці Feature

Поле	Тип даних	Опис
feature_id	uuid	Ідентифікатор зручності
property_id	uuid	Ідентифікатор нерухомості
name	varchar(50)	Назва зручності

Таблиця 2.9

Опис таблиці Token

Поле	Тип даних	Опис
token_id	uuid	Ідентифікатор токєну
user_id	uuid	Ідентифікатор користувача
token	varchar	Токєн
type	varchar(30)	Тип токєну
used	boolean	Використаний токєн
newPasswordHash	varchar(255)	Новий хешований пароль
Expires_at	datetime	Кінцева дата придатності токєну
Created_at	datetime	Дата додавання токєну

Таблиця 2.10

Опис таблиці Booking

Поле	Тип даних	Опис
booking_id	uuid	Ідентифікатор бронювання
listing_id	uuid	Ідентифікатор оголошення
tenant_id	uuid	Ідентифікатор орендатора
start_date	datetime	Дата початку оренди
end_date	datetime	Дата завершення оренди
guests	int	Кількість гостей
total_price	float	Повна ціна
status	varchar(30)	Статус оренди
created_at	datetime	Дата створення бронювання
updated_at	datetime	Дата оновлення бронювання

Таблиця 2.11

Опис таблиці Payment

Поле	Тип даних	Опис
payment_id	uuid	Ідентифікатор платежу
booking_id	uuid	Ідентифікатор бронювання
payer_id	uuid	Ідентифікатор платника
amount	float	Ціна платежу
currency	varchar(30)	Валюта платежу
payment_method	varchar(30)	Метод платежу
transaction_id	varchar(255)	Ідентифікатор транзакції
status	varchar(30)	Статус платежу
paid_at	datetime	Дата платежу
refunded_at	datetime	Дата повернення платежу
created_at	datetime	Дата створення платежу

Опис таблиці Review

Поле	Тип даних	Опис
review_id	uuid	Ідентифікатор відгуку
booking_id	uuid	Ідентифікатор бронювання
property_id	uuid	Ідентифікатор нерухомості
user_id	uuid	Ідентифікатор користувача
rating	varchar(30)	Рейтинг користувача
comment	varchar(30)	Відгук користувача
created_at	varchar(255)	Дата створення відгуку
updated_at	varchar(30)	Дата оновлення відгуку

На основі визначених таблиць побудовано діаграму структури бази даних, що відображає взаємозв'язки між основними сутностями системи (рис. 2.3).

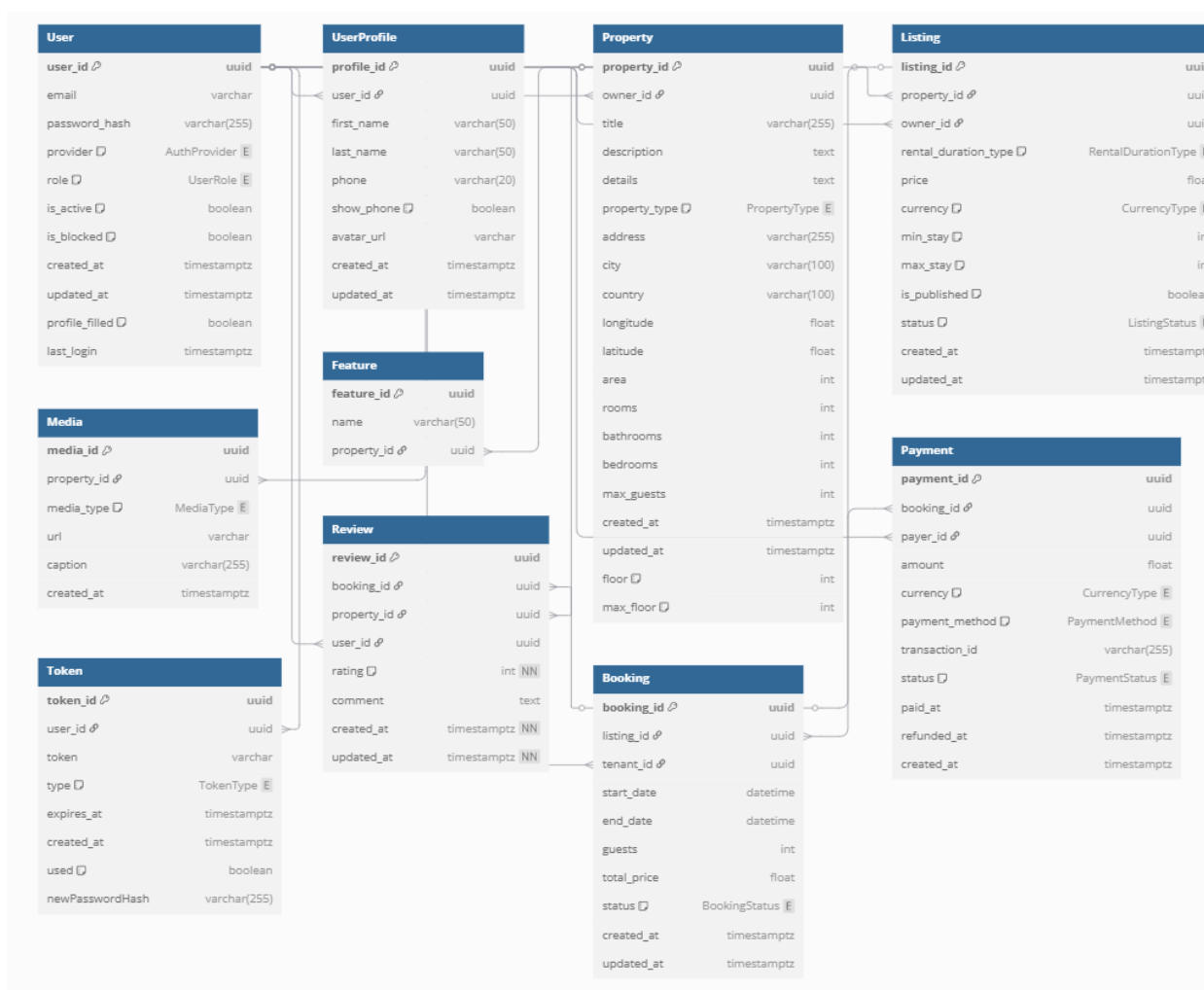


Рисунок 2.3 – Діаграма структури бази даних

2.3 Проектування та реалізація алгоритмів роботи системи

З огляду на функціональне призначення системи, веб-застосунок характеризується інтуїтивно зрозумілим інтерфейсом та доступністю основних дій дозволяючи користувачам ефективно взаємодіяти з платформою без необхідності створення складних алгоритмів. На рис. 2.4-2.6 наведено діаграму активності, яка відображає загальну послідовність дій користувачів у межах роботи платформи, а нижче наведено опис ключових етапів взаємодії з системою.

Основні можливості користувачів відповідно до їх ролей на платформі наведено в табл. 2.13.

Таблиця 2.13

Основний функціонал користувачів залежно від ролі

Роль користувача	Функціонал користувача
Неавторизований користувач	Користувач може авторизуватися або зареєструватися на платформі
	Користувач може відновити пароль
	Користувач може авторизуватися або зареєструватися через Google
	Користувач може шукати, сортувати та фільтрувати оголошення на головній сторінці
	Користувач може переглядати конкретне оголошення
Авторизований (роль орендатор)	Користувач може шукати, сортувати та фільтрувати оголошення на головній сторінці
	Користувач може переглядати конкретне оголошення
	Користувач може бронювати нерухомість по оголошенню
	Користувач може редагувати персональну інформацію
	Користувач може переглядати власні бронювання
	Користувач може скасовувати або оплачувати бронювання
	Користувач може залишити відгук щодо бронювання
	Користувач може змінювати пароль

Авторизований (роль орендодавець)	Користувач може шукати, сортувати та фільтрувати оголошення на головній сторінці
	Користувач може переглядати конкретне оголошення
	Користувач може редагувати персональну інформацію
	Користувач може додавати нерухомість
	Користувач може переглядати власні нерухомості
	Користувач може редагувати власні нерухомості
	Користувач може видаляти власні нерухомості
	Користувач може додавати оголошення
	Користувач може переглядати власні оголошення
	Користувач може редагувати власні оголошення
	Користувач може видаляти власні оголошення
	Користувач може переглядати бронювання
	Користувач може підтвердити або скасувати бронювання
	Користувач може фіксувати оплату бронювання
Авторизований (роль модератор)	Користувач може шукати, сортувати та фільтрувати оголошення на головній сторінці
	Користувач може переглядати конкретне оголошення
	Користувач може редагувати персональну інформацію
	Користувач може підтверджувати або відхиляти публікацію оголошення
Авторизований (роль адміністратор)	Користувач може шукати, сортувати та фільтрувати оголошення на головній сторінці
	Користувач може переглядати конкретне оголошення
	Користувач може редагувати персональну інформацію
	Користувач може підтверджувати або відхиляти публікацію оголошення

	Користувач може переглядати всіх зареєстрованих користувачів
	Користувач може змінювати ролі іншим користувачам
	Користувач може блокувати або розблоковувати інших користувачів

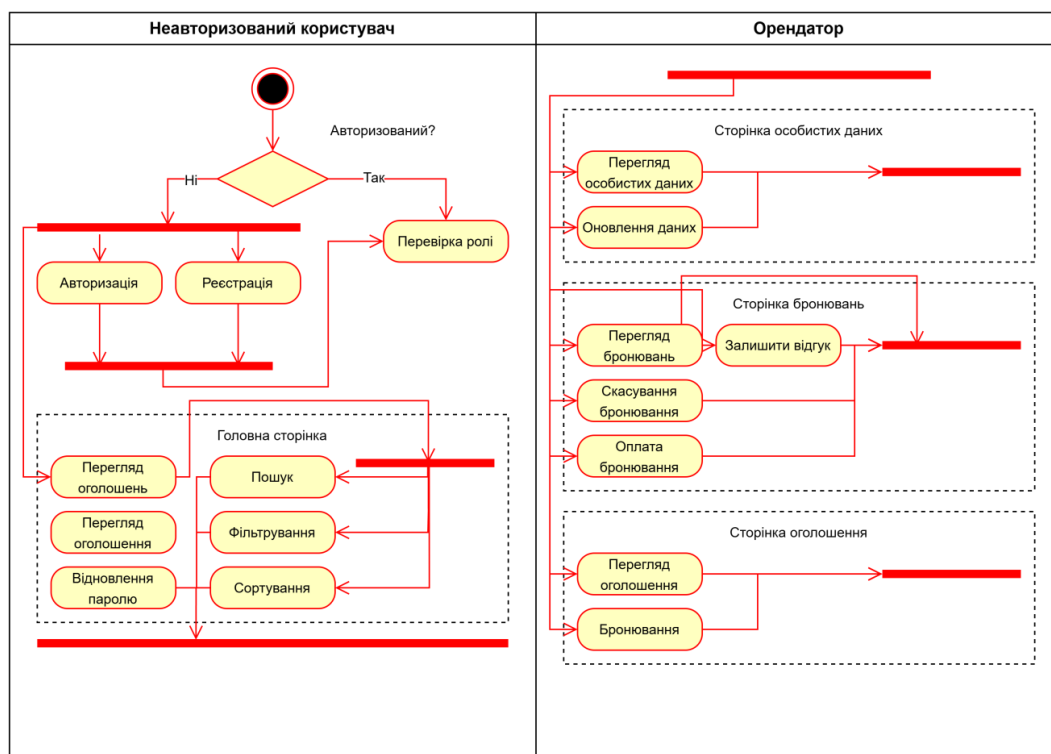


Рисунок 2.4 – Діаграма активності неавторизованого користувача та орендатора

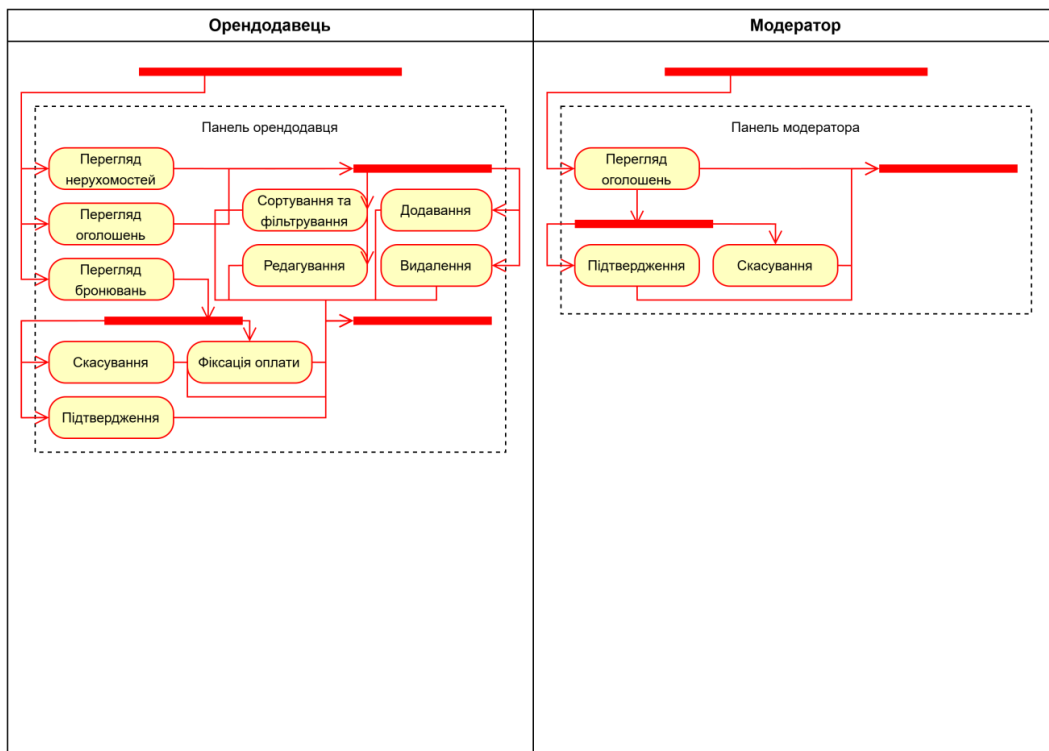


Рисунок 2.5 – Діаграма активності орендодавця та модератора



Рисунок 2.6 – Діаграма активності адміністратора

2.4 Реалізація онлайн-платформи оренди житла

Розглянемо повну реалізацію створення нерухомості.

Відповідно перед тим, як щось додавати нам потрібна схема збереження даних. Код схеми наведений на рис. 2.7.

Детальний опис схеми відповідає таблиці Property, яка була описана вище в попередньому підрозділі.

Після формування схеми даних потрібно перевіряти коректність введення інформації від користувача перед її обробкою та збереженням у базі даних. З метою забезпечення валідації вхідних даних було використано бібліотеку Zod, яка описує схеми типів та перевіряє відповідність отриманих значень заданим вимогам.

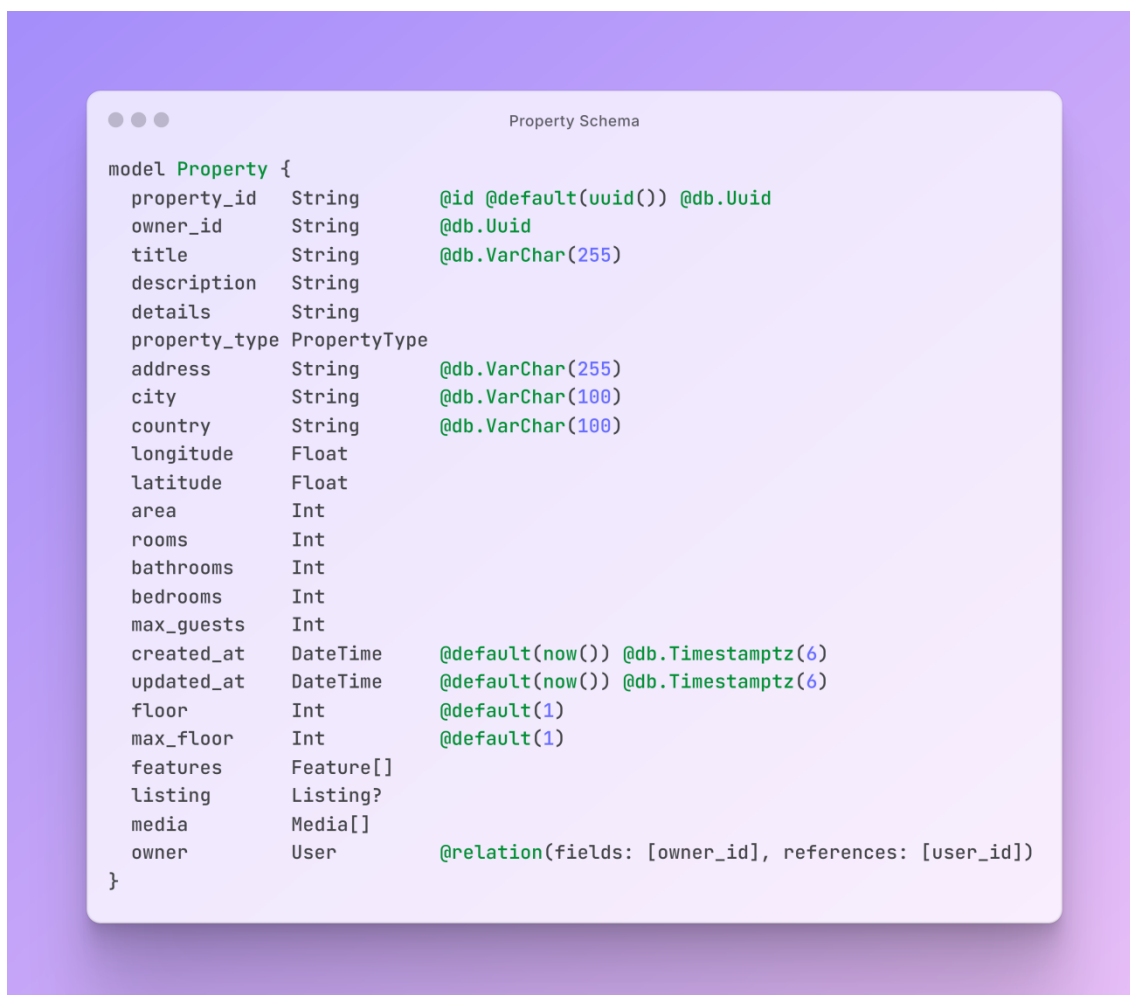


Рисунок 2.7 – Схема Property

Zod є декларативною бібліотекою для валідації та парсингу даних у JavaScript та TypeScript. Вона визначає строгі типи та правила перевірки забезпечуючи високий рівень надійності при роботі з формами, API-запитами та іншими джерелами вхідних даних. Синтаксис бібліотеки орієнтований на читабельність і

тісно інтегрується з системою типів TypeScript з метою зменшення кількості помилок під час розробки.

Код валідації даних наведений на рис. 2.8.

Даний код описує схему валідації даних для створення об'єкта нерухомості. Кожне з полів, яке користувач повинен заповнити під час додавання нового об'єкта, перевіряється на відповідність заданим умовам.



Рисунок 2.8 – Код валідації даних нерухомості

Після створення валідаційної схеми можна реалізовувати дію, яка буде додавати нову нерухомість до бази даних.

Відповідний код додавання нерухомості наведений на рис. 2.9



Рисунок 2.9 – Код для додавання нерухомості до бази даних

Код виконує перевірку вхідних даних за допомогою схеми, створеної в попередньому кроці. Якщо валідація проходить успішно, здійснюється перевірка авторизації користувача через функцію `getAuthUser()`. У разі неуспішної авторизації або виникнення будь-якої помилки на будь-якому етапі, вона буде

виведена у консолі, а клієнту буде повернуто null. Після підтвердження прав користувача виконується завантаження медіафайлів (зображень або відео), прикріплених до об'єкта нерухомості. Для цього використовується сторонній сервіс Cloudinary – хмарне рішення для зберігання та обробки мультимедійного контенту, яке забезпечує масштабоване та оптимізоване збереження зображень та відео. Файли завантажуються через API `upload_stream` з обробкою потоків файлів у форматі `arrayBuffer`. Кожен завантажений файл класифікується відповідно до свого MIME-типу як фотографія або відео. У результаті формується масив об'єктів із посиланнями на завантажені ресурси та зазначенням їхнього типу. В кінці виконується створення запису в таблиці `Property` бази даних через `Prisma ORM`. У запиті вказуються основні дані об'єкта, що пройшли валідацію, зв'язок між об'єктом і користувачем (власником) через поле `owner`, масив мультимедійних елементів, які створюються в пов'язаній таблиці `Media`, набір зручностей (`features`), які також створюються як окремі записи та зв'язуються з конкретним об'єктом нерухомості. Після успішного виконання всієї процедури функція повертає створений запис об'єкта нерухомості разом із пов'язаними медіа та зручностями.

Для реалізації взаємодії клієнта та сервера для логіки додавання нерухомості можна використати два основні підходи: традиційний API-інтерфейс або інтеграцію через мутації.

У першому випадку створюється окремий маршрут у директорії `api`, до якого клієнтський інтерфейс надсилає запит разом із відповідними даними. Ці дані надходять до функції `createPropertyAction`, яка обробляє запит, здійснює збереження об'єкта нерухомості в базі даних і повертає відповідь клієнту.

Підхід є універсальним і добре масштабованим, однак він потребує додаткових проміжних рівнів логіки, перевірки автентифікації користувача, авторизації доступу, обробки помилок, логування та іншого, що збільшує час розробки та ускладнює загальну структуру коду.

Альтернативою є використання мутацій, які забезпечують виклик серверних дій безпосередньо з клієнта. У контексті фреймворку `Next.js`, де доступна концепція `server actions`, мутації дозволяють напямую звертатися до цих функцій, минаючи

необхідність створення окремого API-ендпоінту. Таким чином, функція `createPropertyAction` може бути викликана з клієнтського коду як серверна дія, при цьому всі необхідні операції (валідація, збереження медіа, створення запису у базі даних, тощо) виконуються на сервері, а результат повертається до клієнта.

Для спрощення реалізації мутацій на стороні клієнта використовується бібліотека `RTK Query` – інструмент з екосистеми `Redux Toolkit`, призначений для ефективного управління запитами до серверу. `RTK Query` дозволяє оголошувати мутації у вигляді окремих функцій, які автоматично обробляють відправку даних, кешування відповідей, повторні запити та обробку помилок. Додатково зменшується обсяг коду та покращується взаємодія між клієнтом і сервером.

Код створення нерухомості у вигляді мутації наведений на рис. 2.10.

В коді використовується хук `useMutation` із бібліотеки `React Query` з автоматичним керуванням станами запиту та обробкою помилок. Задається основна функція мутації `mutationFn`, яка приймає дані форми `CreatePropertyFormData` і передає їх до серверної функції `createPropertyAction`.

Параметр `onSuccess` визначає дію, яка виконується у разі успішного завершення мутації. Якщо у відповідь повертається `null` (тобто створення не відбулося), відображається повідомлення про помилку за допомогою бібліотеки `Toastify`, яка відображає попап повідомлення. Якщо ж об'єкт нерухомості було успішно створено, виводиться повідомлення про успішне створення, оновлюються (інвалідуються) відповідні кешовані запити: список усіх нерухомостей та конкретно створена нерухомість. У випадку виникнення помилки під час виконання запиту, спрацьовує функція `onError`, яка відображає відповідне повідомлення про помилку через `Toastify`. На стороні клієнта можна викликати асинхронну функцію `createPropertyMutation` для створення нерухомості. Додатково витягнуто з хуку `useMutation` - `isPending` для перевірки стану виконання мутації (активна чи ні), використовується на клієнті для показу індикатора завантаження.



Рисунок 2.10 – Код додавання нерухомості у вигляді мутації

На клієнтській стороні для обробки введених користувачем даних та відправки форми використовується бібліотека React Hook Form – інструмент для керування станом форм у React-додатках з високою продуктивністю і мінімальною кількістю повторних рендерів.

Відповідний код хука для валідації, відправки форми та функції обробки відправленої форми наведено на рис. 2.11.



Рисунок 2.11 – Код валідації та відправки форми

У коді через `useForm<CreatePropertyFormData>` виконується ініціалізація форми з типізацією даних та встановленням режиму валідації "all" (перевірка відбувається під час будь-якої зміни значення або фокусу елемента), механізм валідації – `zodResolver`.

Функція `onSubmit` викликається після успішного проходження валідації. У ній здійснюється виклик асинхронної функції `createProperty` – мутація, яка була створено у попередньому кроці. У разі успішного завершення виконання мутації орендодавець перенаправляється на сторінку зі списком власних нерухомостей.

Додатково було реалізовано серверні функції, які виконуються щодня з метою скасування бронювань у випадках, коли термін бронювання завершився або

орендар не здійснив оплату протягом 24 годин з моменту підтвердження орендодавцем.

Детальний код роботи цих функцій наведено в додатку А.

Як уже зазначалося, оплата здійснюється через платіжну систему LiqPay, яка надає зручний у використанні API. У програмному коді реалізовано логіку оплати за допомогою двох API-маршрутів: маршрут для ініціалізації платежу та callback-маршрут для обробки успішно здійсненої транзакції.

Детальний код роботи маршрутів наведений в додатку Б.

Висновки до другого розділу

Під час проектування онлайн-платформи оренди житла було здійснено аналіз основних сценаріїв взаємодії користувачів з системою, а також сформульовано вимоги до функціональних та нефункціональних характеристик програмного забезпечення. На основі проведеного аналізу було розроблено структурну модель системи, яка охоплює ключові компоненти клієнтської та серверної частин.

Було спроектовано базу даних для забезпечення зберігання основної інформації, зокрема даних користувачів, об'єктів нерухомості, бронювань та іншого контенту у вигляді реляційної моделі, реалізованої на основі PostgreSQL.

Було описано основну логіку взаємодії користувачів із системою, а саме ключові дії, що реалізують типові сценарії використання веб-платформи.

Було детально розглянуто основний функціонал веб-застосунку, зокрема програмну реалізацію процесу створення об'єктів нерухомості, механізм завантаження медіафайлів на віддалений сервер, логіку взаємодії клієнтської та серверної частин за допомогою мутацій, а також валідацію та обробку даних на стороні клієнта.

РОЗДІЛ 3 ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ ОНЛАЙН-ПЛАТФОРМИ ОРЕНДИ ЖИТЛА

3.1 Порядок встановлення та налаштування параметрів платформи

Порядок розгортання та налаштування онлайн-платформи оренди житла є важливою складовою процесу його впровадження, оскільки від правильної конфігурації залежить стабільність роботи системи та ефективність взаємодії між її компонентами. На етапі розгортання передбачається інсталяція клієнтських і серверних елементів, а також налаштування зовнішніх служб, зокрема бази даних, хмарного сховища медіафайлів та платіжного сервісу.

На рис. 3.1 наведено діаграму розгортання (Deployment Diagram), що описує архітектуру платформи.

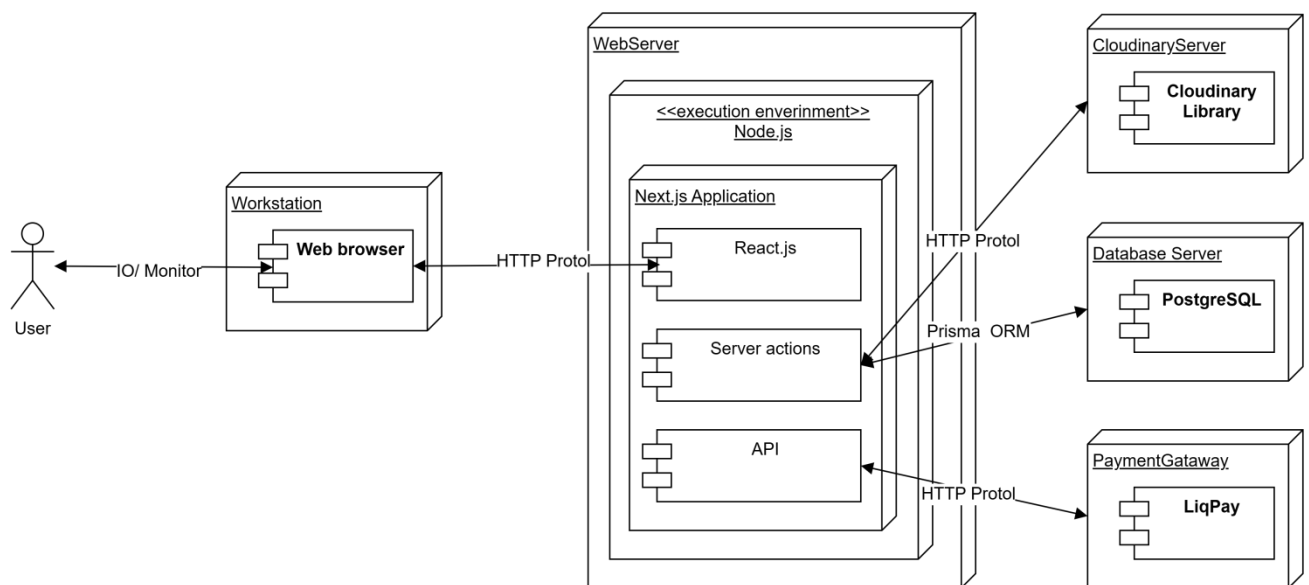


Рисунок 3.1 – Діаграма розгортання платформи

Опис архітектури діаграми розгортання:

1. Workstation – клієнтський пристрій, з якого користувач здійснює доступ до веб-браузера.
2. Web browser – веб-браузер користувача.
3. Web Server – серверна частина веб-застосунку реалізована у середовищі Node.js на основі фреймворку Next.js. Доступ до неї клієнт здійснює за допомогою протоколу HTTP. Клієнтська частина представлена окремим React.js-додатком, що виконується у веб-браузері користувача.

4. Server Actions – серверні функції, за допомогою яких здійснюється отримання, обробка та збереження даних. Для взаємодії з базою даних використовується Prisma ORM, для зберігання медіафайлів – віддалений сервер.

5. CloudinaryServer – сервер, призначений для зберігання медіафайлів, до якого здійснюється доступ через Cloudinary Library.

6. Database Server – сервер, що забезпечує зберігання усіх даних, отриманих із веб-сервера. Дані зберігаються у реляційній базі даних PostgreSQL.

7. PaymentGateway – платіжний шлюз, через який здійснюється оплата бронювання, використовується платіжна система LiqPay.

3.2 Структура інтерфейсу платформи

На головній сторінці неавторизований користувач має можливість переглядати, здійснювати пошук, сортування та фільтрацію оголошень. На рис. 3.2 представлено головну сторінку веб-застосунку для неавторизованого користувача.

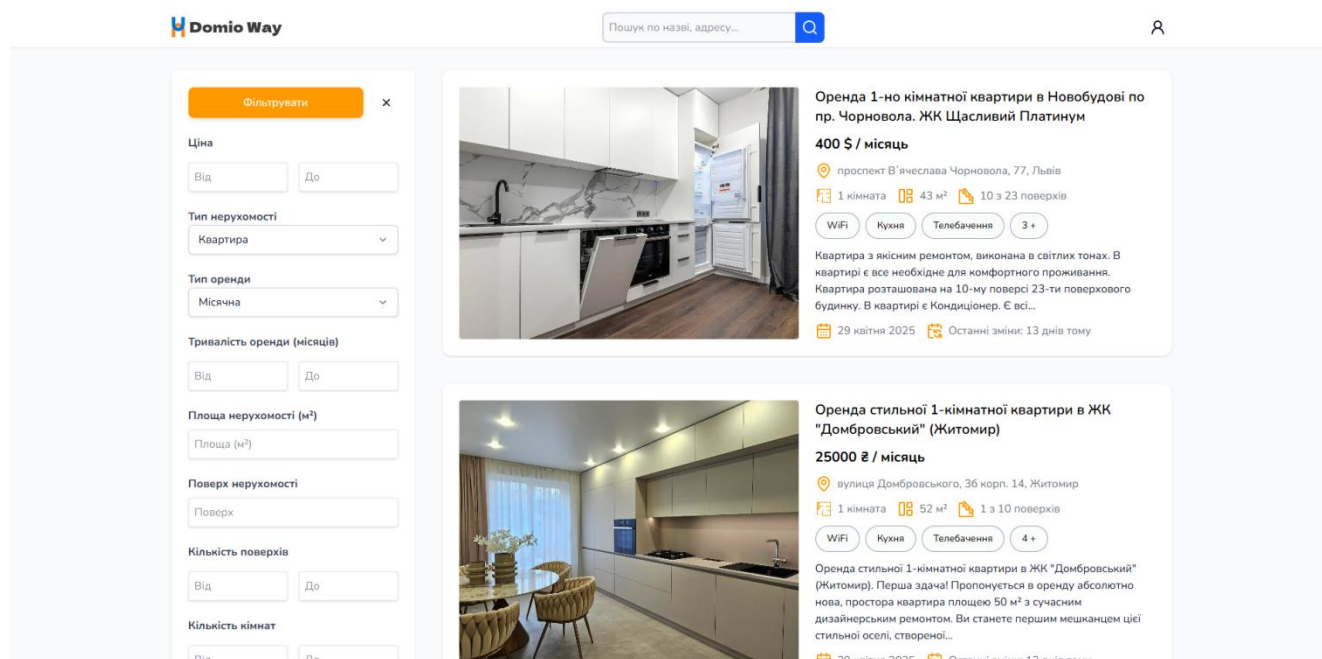


Рисунок 3.2 – Головна сторінка платформи

Після натискання користувачем на кнопку з іконкою профілю, відкривається форма автентифікації, яка передбачає вхід за допомогою електронної пошти та пароля або через обліковий запис Google. У разі відсутності облікового запису користувач може скористатися посиланням “Зареєструватися”, після чого йому буде запропоновано заповнити відповідну реєстраційну форму. Додатково

передбачено можливість реєстрації через Google. На рис. 3.3 зображено вікно входу на платформу.

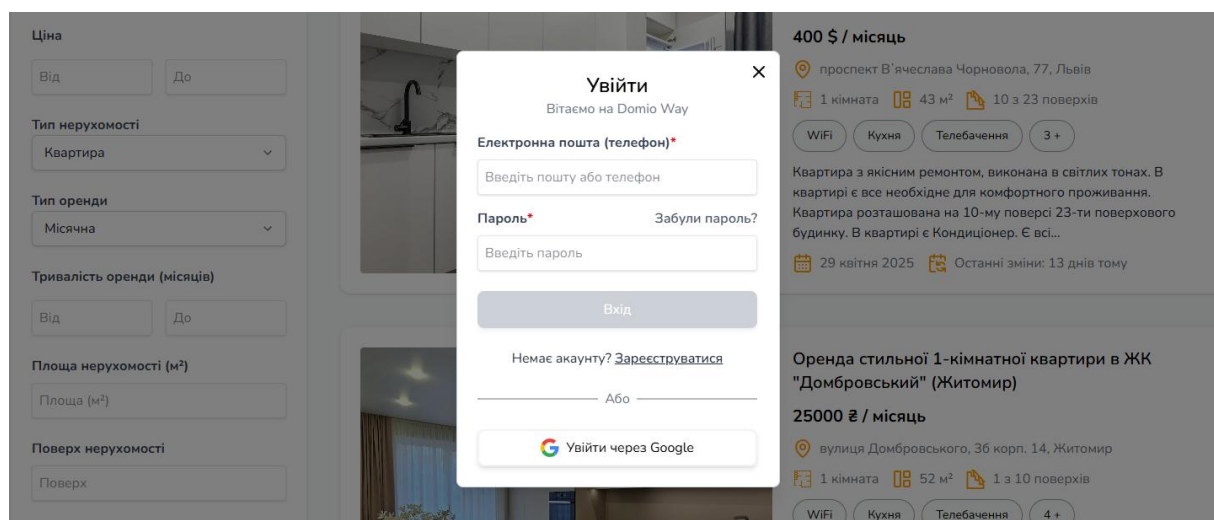


Рисунок 3.3 – Вікно входу на платформу

У разі втрати пароля користувач має можливість його відновити шляхом зміни. Для цього необхідно натиснути на посилання “Забули пароль?”, після чого відкриється форма, в якій слід вказати адресу електронної пошти, прив’язаної до облікового запису. На рис. 3.4 зображено вікно відновлення паролю.

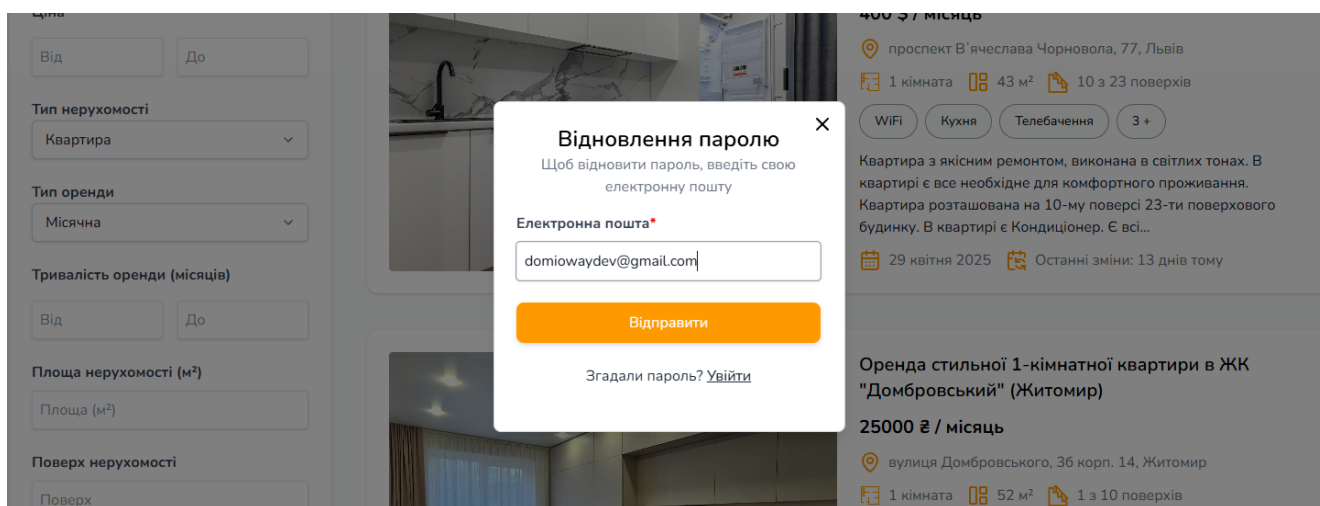


Рисунок 3.4 – Вікно відновлення паролю

Після натискання кнопки “Відправити” користувач побачить повідомлення щодо перевірки власної електронної пошти. На рис. 3.5 наведено повідомлення.

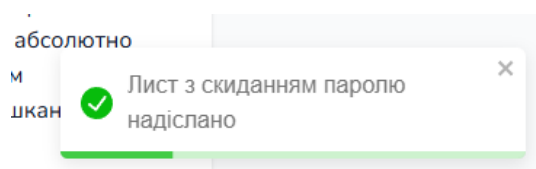


Рисунок 3.5 – Вікно з повідомленням

Після перевірки електронної скриньки та переходу за надісланим посиланням користувачу буде відкрито нове вікно з формою для введення нового пароля. На рис. 3.6 наведено вікно відновлення паролю.

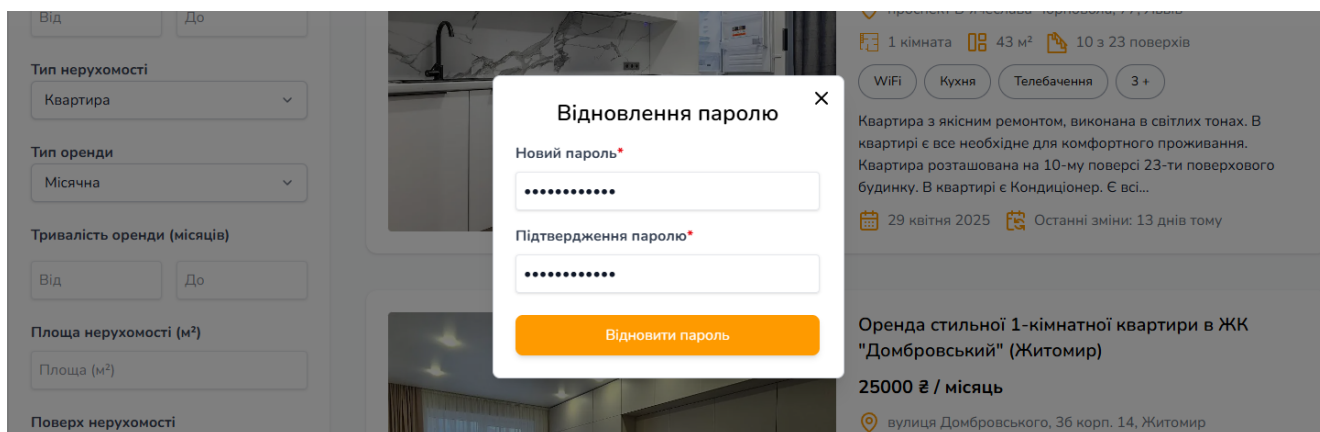


Рисунок 3.6 – Вікно відновлення паролю

Після натискання кнопки “Відновити пароль” пароль буде успішно змінено, після чого з’явиться повідомлення про успішне скидання паролю. На рис. 3.7 наведено повідомлення успішного скидання паролю.

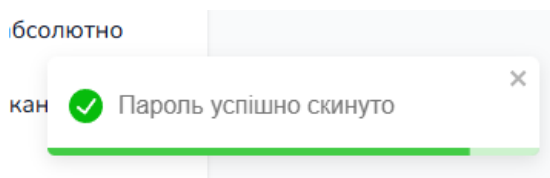


Рисунок 3.7 – Успішне скидання паролю

Увійшовши під обліковим записом орендатора, користувач отримує доступ до функціональності, доступної неавторизованим користувачам, але з розширеними можливостями. Користувач може перейти в свій профіль. В профілі користувача відображається інформація про користувача, а також додаткові відомості, що залежать від ролі. В лівій частині екрана розташоване навігаційне меню профілю. Якщо користувач бажає змінити свої особисті дані, він може скористатися пунктом меню “Особисті дані”. На рис. 3.8 наведено сторінку редагування особистих даних.

The screenshot shows the 'Особисті дані' (Personal Data) page of the Domio Way application. On the left is a sidebar with a user profile card for 'Іван Петренко' (Ivan Petrenko) and a menu with options: 'Профіль', 'Бронювання', 'Особисті дані' (highlighted), 'Зміна паролю', and 'Вихід'. The main content area contains a form with the following fields: 'Прізвище*' (Last Name) with the value 'Петренко', 'Ім'я*' (First Name) with the value 'Іван', 'Електронна пошта*' (Email) with the value 'tenant@gmail.com', and 'Номер телефону*' (Phone Number) with the value '+380631556522'. An 'Оновити' (Update) button is located at the top right of the form. The footer includes the Domio Way logo, contact information, and social media links.

Рисунок 3.8 – Сторінка редагування особистих даних

Додатково є можливість зміни паролю. Якщо користувач бажає змінити пароль, він може скористатися пунктом меню “Зміна паролю”. На рис. 3.9 наведено сторінку зміни паролю.

The screenshot shows the 'Зміна паролю' (Change Password) page of the Domio Way application. The sidebar is identical to the previous screenshot, but the 'Зміна паролю' option is highlighted. The main content area contains a form with the following fields: 'Поточний пароль*' (Current Password) with the placeholder 'Введіть поточний пароль', 'Новий пароль*' (New Password) with the placeholder 'Введіть новий пароль', and 'Підтвердження паролю*' (Password Confirmation) with the placeholder 'Введіть новий пароль'. An 'Оновити' (Update) button is located at the top right of the form. The footer is identical to the previous screenshot.

Рисунок 3.9 – Сторінка зміни паролю

Перейдемо безпосередньо до процесу орендування нерухомості. Для цього користувачу необхідно відкрити головну сторінку веб-застосунку та обрати оголошення, що його зацікавило. Після цього слід натиснути на відповідне

оголошення, щоб перейти на сторінку з детальною інформацією про об'єкт оренди. На рис. 3.10 наведено сторінку оголошення.

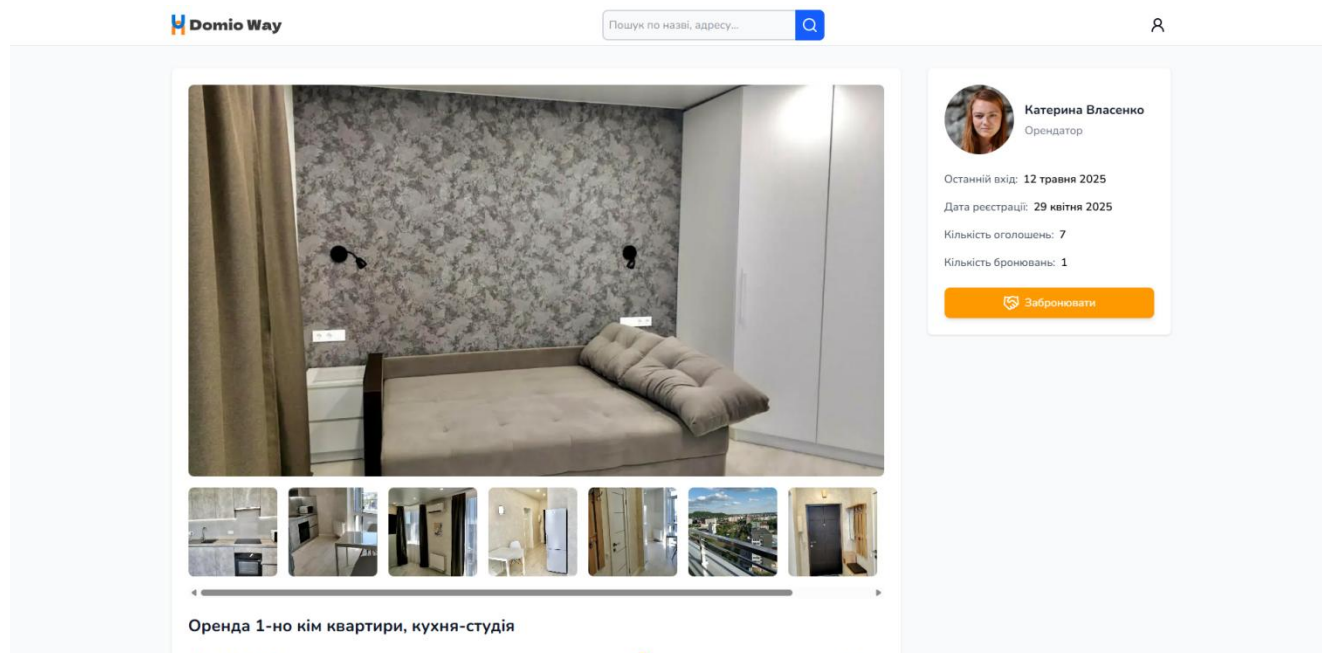


Рисунок 3.10 – Сторінка оголошення

На сторінці оголошення користувач може детально ознайомитися з інформацією про нерухомість, зокрема переглянути фотографії, опис, технічні характеристики, перелік зручностей, а також розташування об'єкта на карті. У правій частині сторінки відображається інформація про орендодавця, який опублікував оголошення: час останнього входу, дата реєстрації, кількість опублікованих об'єктів та активних бронювань. Основна функція, що нас цікавить, – це кнопка “Забронювати”. Після її натискання користувач перенаправляється на сторінку бронювання нерухомості. На рис. 3.11 наведено сторінку бронювання нерухомості.

На сторінці оформлення бронювання з лівого боку відображається інформація про користувача, орендодавця, який опублікував відповідне оголошення, а також про об'єкт нерухомості, що пропонується в оренду. Праворуч розміщено форму з параметрами бронювання, яку необхідно заповнити. Наприклад, користувач може обрати оренду терміном на 4 місяці, починаючи з поточної дати. Після підтвердження обраного періоду користувач натискає кнопку “Бронювати нерухомість”. Після цього його буде автоматично переадресовано на

сторінку з історією бронювань, де доступна детальна інформація про кожне з них. На рис. 3.12 наведено сторінку бронювань користувача.

Domio Way Пошук по назві, адресу...

1 1

Зручності нерухомості

WiFi Кухня Телебачення Душ Шафа Лівко

Розташування нерухомості

Країна: Україна Місто: Львів

Адреса: проспект В'ячеслава Чорновола, 77

Ціна оренди: 400 USD

Мінімальна кількість місяців оренди: 1

Максимальна кількість місяців оренди: 12

Дата початку оренди*: 13.05.2025

Кількість місяців (від 1 до 12)*: 4

Дата закінчення оренди: 13.09.2025

Кількість осіб*: 1

Загальна ціна оренди: \$1600

[Бронювати нерухомість](#)

Рисунок 3.11 – Сторінка бронювання нерухомості

Domio Way Пошук по назві, адресу...

Бронювання

Пошук по ID

Іван Петренко [Орендар](#)

[Профіль](#) [Бронювання](#) [Особисті дані](#) [Зміна паролю](#) [Вихід](#)

Оренда 1-но кімнатної квартири в Новобудові по пр... 400 \$ / місяць

проспект В'ячеслава Чорновола, 77, Львів 1 кімната 43 м² 1 з 5 поверхів

Бронювання буде скасовано автоматично, якщо ви не оплатите його частину або орендодавець не підтвердить його протягом 24 годин

Орендатор: Катерина Власенко +380976442033

Орендар: Іван Петренко +380631556522

Дата початку оренди	13 травня 2025 р.
Дата закінчення оренди	13 вересня 2025 р.
Кількість місяців оренди	4
Кількість гостей	1
Статус оплати	В очікуванні підтвердження
Разом	1600 \$

#13f18f6d-3c0d від 13 травня 2025 р. [Очікує підтвердження](#)

#112ac528-8ed6 від 29 квітня 2025 р. [Скасовано](#)

#8c64cb1b-ad24 від 26 квітня 2025 р. [Підтверджено](#)

Рисунок 3.12 – Сторінка бронювань користувача

Як видно, користувач успішно здійснив бронювання нерухомості, однак підтвердження з боку орендодавця ще не відбулося. Після підтвердження бронювання орендодавцем, на сторінці оновиться інформація про бронювання та

з'явиться додаткова кнопка для здійснення оплати. На рис. 3.13 наведено сторінку з успішним підтвердженням бронювання від орендодавця.

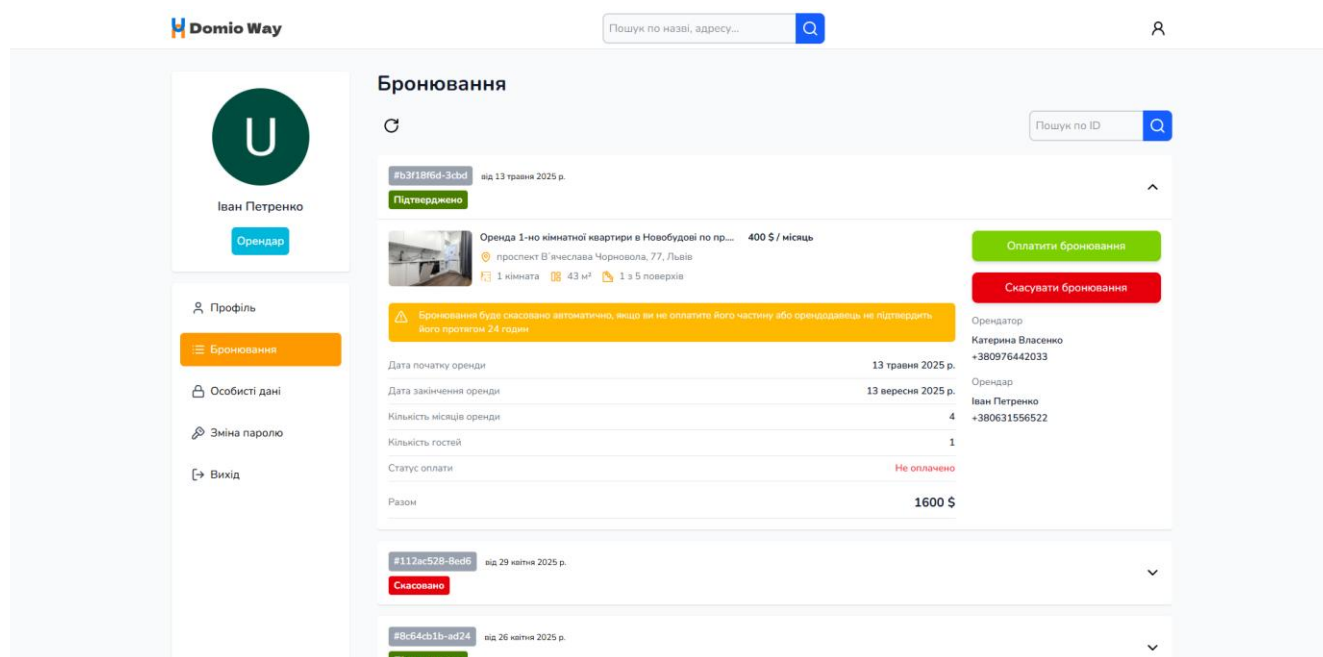


Рисунок 3.13 – Оновлена сторінка з успішним підтвердженням бронювання

Оплатити бронювання можна одним із двох способів: онлайн за допомогою банківської картки або готівкою. У першому випадку користувач натискає кнопку “Оплатити бронювання”, після чого відкривається вікно, де можна вказати суму для часткової або повної оплати оренди. У другому випадку передбачено фіксацію готівкової оплати, яку підтверджує орендодавець. Для цього він також використовує кнопку “Оплатити” на своїй панелі, після чого з’являється форма для введення суми, яка була сплачена орендарем. Наприклад, користувач здійснює оплату карткою. Користувач натискає на кнопку “Оплатити бронювання”, після чого відкривається вікно (рис. 3.14), у якому необхідно ввести кількість днів (або місяців), які потрібно оплатити. Після введення кількості, користувач підтверджує дію натисканням на кнопку “Оплатити”, після чого його буде переадресовано на сторінку проведення платежу. На рис. 3.15 наведено сторінку проведення платежу.

На сторінці проведення платежу користувач має можливість або скасувати оплату та повернутися на сторінку бронювань, або продовжити процес і здійснити оплату. Для цього необхідно ввести особисті платіжні дані та натиснути кнопку підтвердження оплати. Після успішного завершення операції користувач автоматично перенаправляється на сторінку своїх бронювань. Наприклад, якщо

користувач обирає здійснення оплати, він вводить необхідні дані, натискає кнопку «Оплатити» та після підтвердження транзакції отримує можливість завантажити квитанцію про оплату (рис. 3.16) та повернутися до списку власних бронювань.

На рис. 3.17 наведено здійснену часткову оплату бронювання користувачем. Додатково, користувач має можливість скасувати бронювання за власним бажанням, проте у такому випадку питання повернення коштів вирішується безпосередньо з орендодавцем.

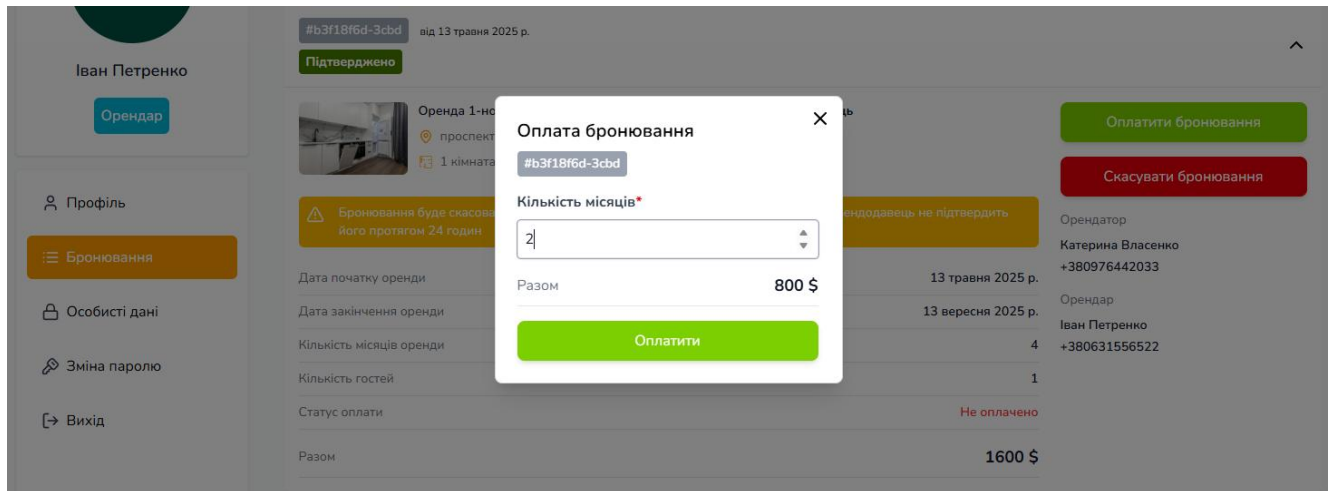


Рисунок 3.14 – Вікно оплати бронювання

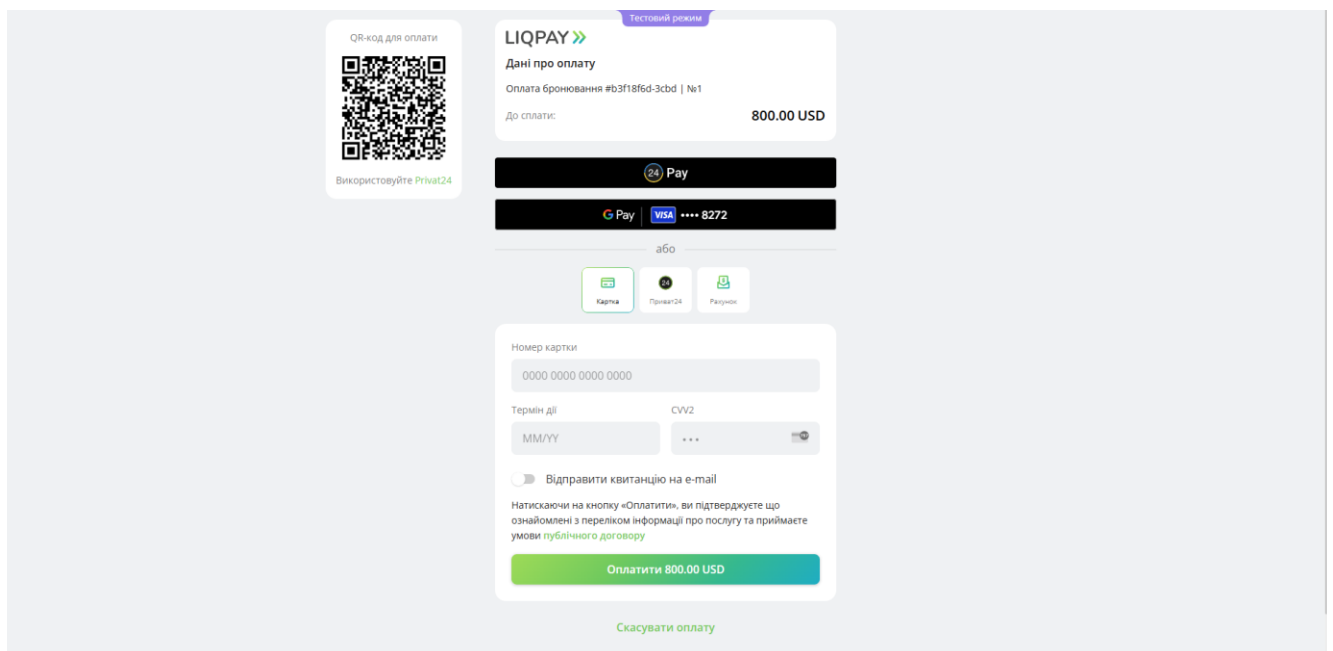


Рисунок 3.15 – Сторінка проведення платежу

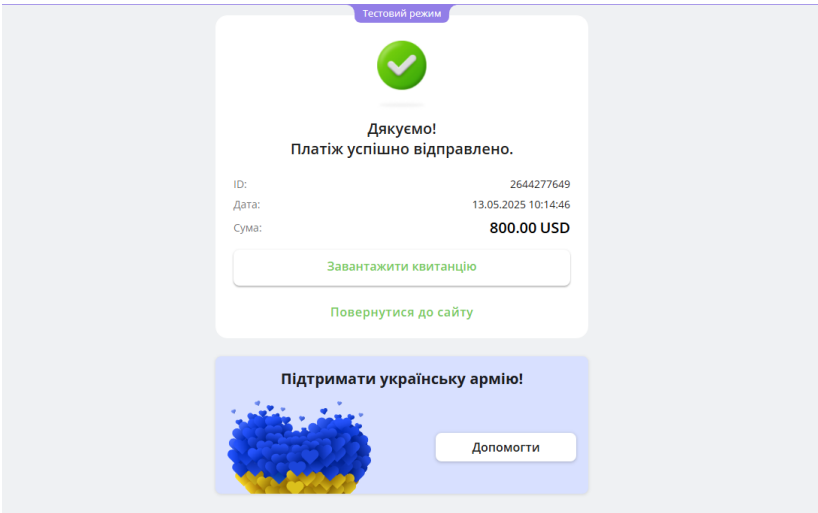


Рисунок 3.16 – Успішна часткова оплата бронювання

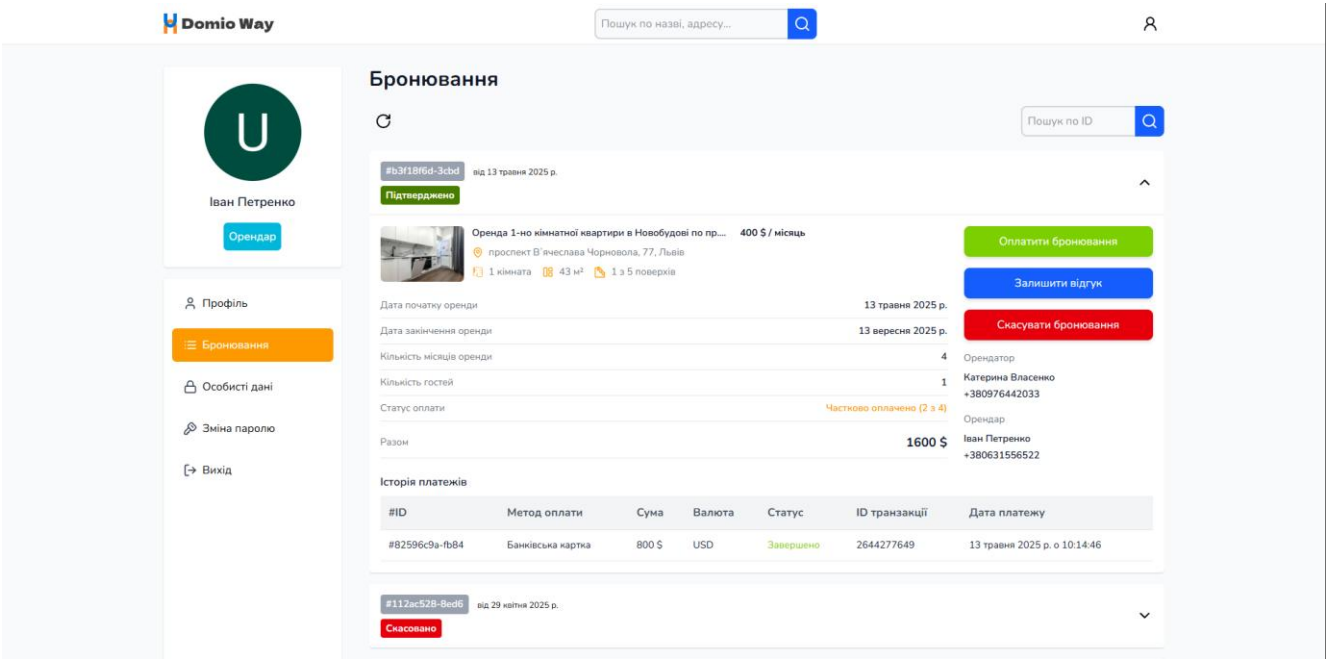


Рисунок 3.17 – Оновлена інформація бронювання користувача

Після здійснення часткової або повної оплати користувач отримує доступ до історії платежів, пов’язаної з конкретним бронюванням та можливість залишити відгук. При натисканні на кнопку “Залишити відгук“ відкривається форма (рис. 3.18), у якій необхідно обрати рейтинг та вказати коментар. Після заповнення полів і натискання кнопки «Надіслати», відгук буде збережено та прив’язано до відповідного оголошення. Відображення оголошення на платформі стає доступним після завершення періоду оренди поточним користувачем. Поточний користувач може переглянути відгуки на окремій сторінці бронювання, де відображається

повна інформація про оголошення, для цього потрібно натиснути на назву оголошення в бронюванні, що розташовано біля ціни, після чого відкриється сторінка оголошення в нижній частині якого можна ознайомитися з усіма наявними відгуками. На рис. 3.19 наведено сторінку оголошення з доданим відгуком.

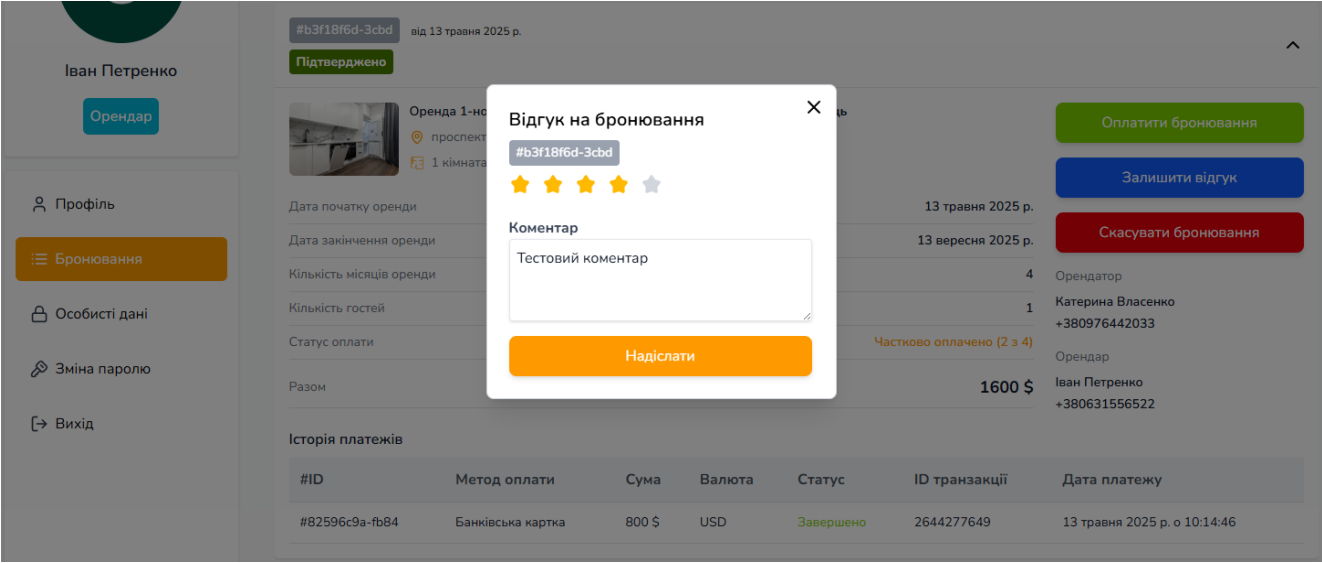


Рисунок 3.18 – Додавання відгука

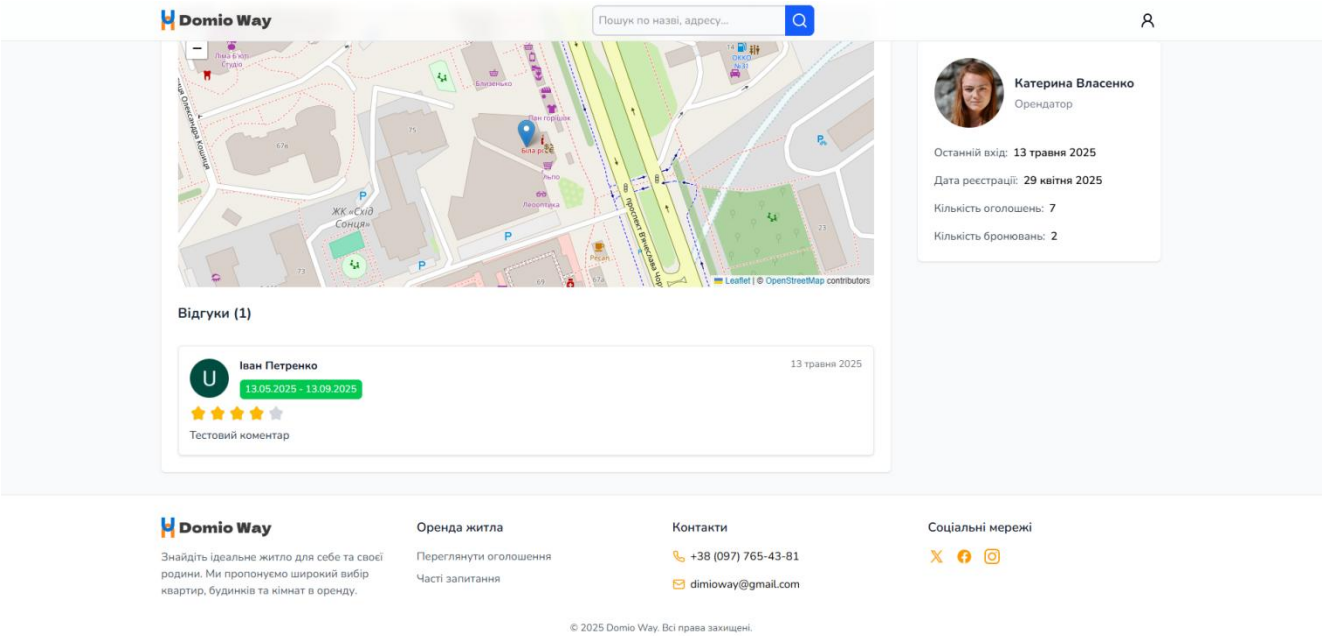


Рисунок 3.19 – Сторінка оголошення з доданим відгуком

Змінимо користувача, увійшовши під обліковим записом із роллю орендодавця. Орендодавець має можливість перейти до панелі управління через меню профілю або за допомогою кнопки додавання нерухомості, що розміщена в хедері. Після натискання на цю кнопку, у хлібних крихтах з’являється посилання

на “Панель орендодавця”, перехід за яким відкриває відповідний інтерфейс. На рис. 3.20 наведено сторінку панелі орендодавця.

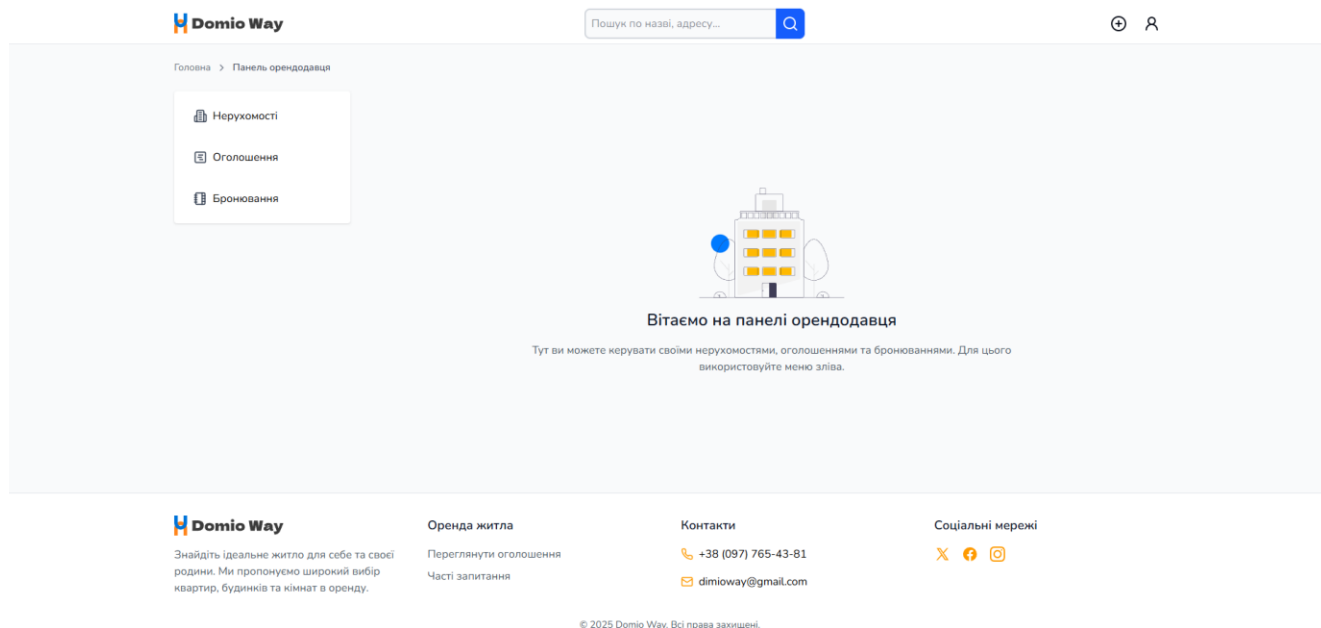


Рисунок 3.20 – Панель орендодавця

Зліва відображається меню навігації, за допомогою якого орендодавець може переміщуватися між різними розділами панелі. На сторінці нерухомостей орендодавцю надається можливість переглядати всі об'єкти, які ним були додані, здійснювати пошук, сортування, редагування або видалення відповідних нерухомостей. На рис. 3.21 наведено сторінку додавання нерухомості.

Сторінка оголошень має аналогічний функціонал до сторінки нерухомостей. Під час створення оголошення (рис. 3.22) користувач заповнює відповідну форму та натискає кнопку “Додати оголошення”. Після цього оголошення зберігається в системі, проте не відображається на головній сторінці. Воно стане доступним для інших користувачів лише після проходження модерації, під час якої адміністратор або модератор перевіряє відповідність оголошення правилам платформи.

На сторінці бронювань, орендодавець може переглянути всі здійснені бронювання орендаторами з можливістю підтвердження, скасування та оплати конкретного бронювання. На рис. 3.23 наведено останнє бронювання здійснено орендатором.

Рисунок 3.21 – Сторінка створення нерухомості

Рисунок 3.22 – Сторінка додавання оголошення

Змінимо користувача, увійшовши під обліковим записом із роллю модератора. Модератор може перейти до панелі керування через меню профілю. Його основне завдання — модерація оголошень, створених орендодавцями. Модератор перевіряє кожне оголошення на відповідність політиці платформи та приймає рішення про його схвалення або відхилення. На рис. 3.24 та рис. 3.25 наведено сторінки оголошень та огляду конкретного оголошення.

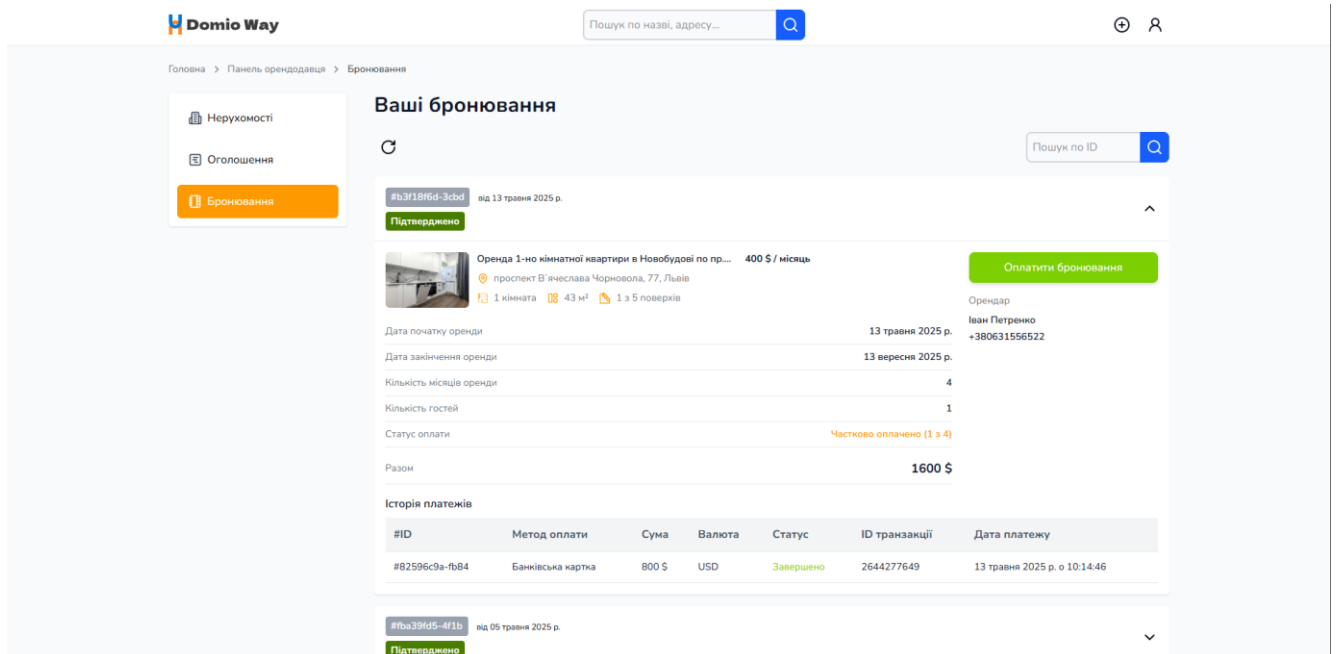


Рисунок 3.23 – Останнє бронювання здійснено орендатором

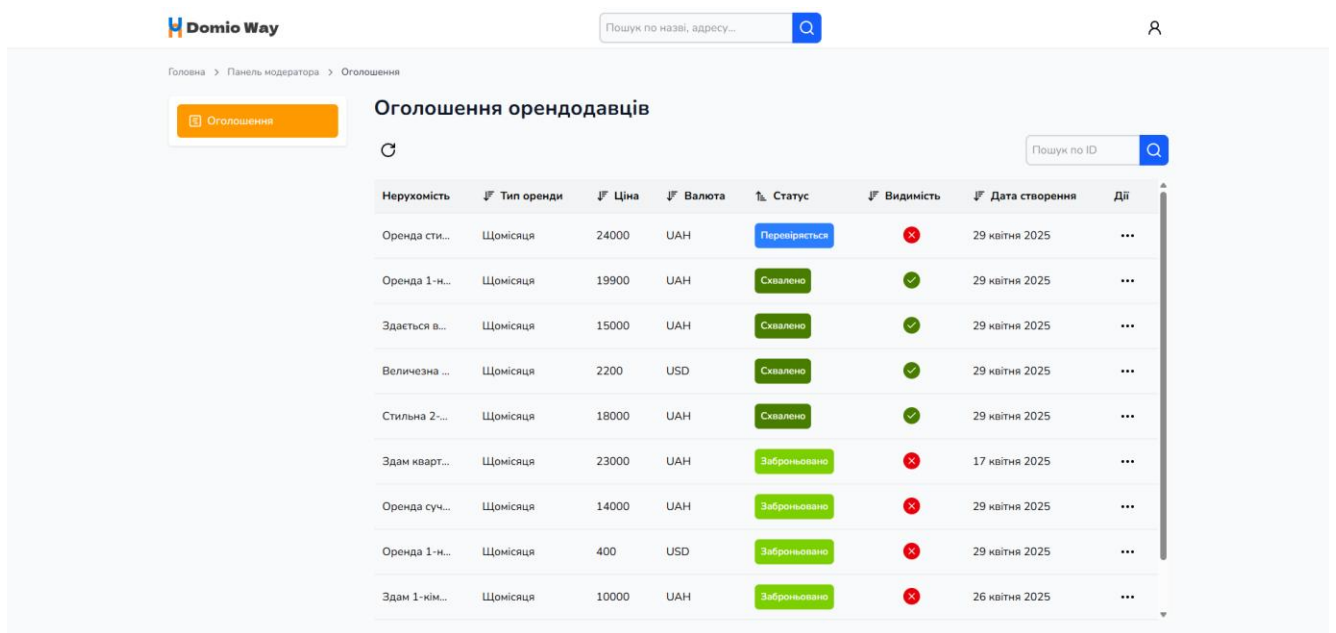


Рисунок 3.24 – Сторінка оголошень

Змінимо користувача, увійшовши під обліковим записом із роллю адміністратора. Адміністратор може перейти до панелі керування через меню профілю. В кожному розділі сторінок доступний пошук та сортування. Крім можливості модерації оголошень, адміністратор має розширений функціонал для керування користувачами: він може змінювати ролі облікових записів та блокувати

окремих користувачів у разі порушення правил платформи. На рис. 3.26 наведено сторінку користувачів на панелі адміністратора.

Domio Way Пошук по назві, адресі...

Головна > Панель модератора > Оголошення > 97246260 са34 475с 987d 04ба88e482d1 > Огляд

Огляд оголошення

Інформація про орендодавця

Прізвище: Власенко

Ім'я: Катерина

Телефон: +380976442033

Статус оголошення

Переглядається

Видимість оголошення: Приховано

Тип оренди: Щомісячно

Ціна оренди: 24000 UAH

Мінімальна кількість місяців оренди: 1

Максимальна кількість місяців оренди: 10

Підтвердити

Відхилити

Обрана нерухомість

Рисунок 3.25 – Сторінка огляду оголошення

Domio Way Пошук по назві, адресі...

Головна > Панель адміністратора > Користувачі

Користувачі

Оголошення Користувачі

ID Пошук...

ID	Прізвище	Ім'я	Електронна пошта	Роль	Блокування	Дата реєстрації	Дії
bdc99179-8559	Анна	Андрійчук	tenant2@gmail.com	Орендар	✓	05 травня 2025	...
565115f6-7535	Власенко	Катерина	landlord@gmail.com	Орендодавець	✓	29 квітня 2025	...
d798e8ae-4ae8	Петренко	Іван	tenant@gmail.com	Орендар	✓	22 квітня 2025	...
6a4c62d6-1c66	Герардіні	Ліза	admin@gmail.com	Адміністратор	✓	17 квітня 2025	...
fce2d85a-5b35	Domio	Cat	domiowaydev@gmail.com	Модератор	✓	17 квітня 2025	...
3ecdc3ab-024a	Шевчук	Олег	test@gmail.com	Орендодавець	✓	16 квітня 2025	...

© 2025 Domio Way. Всі права захищені.

Рисунок 3.26 – Сторінка користувачів на панелі адміністратора

Додатково реалізовано адаптивний дизайн для зручного користування веб-застосунком на мобільних пристроях. На рис. 3.27 та рис. 3.28 наведено адаптивний дизайн під мобільні пристрої.

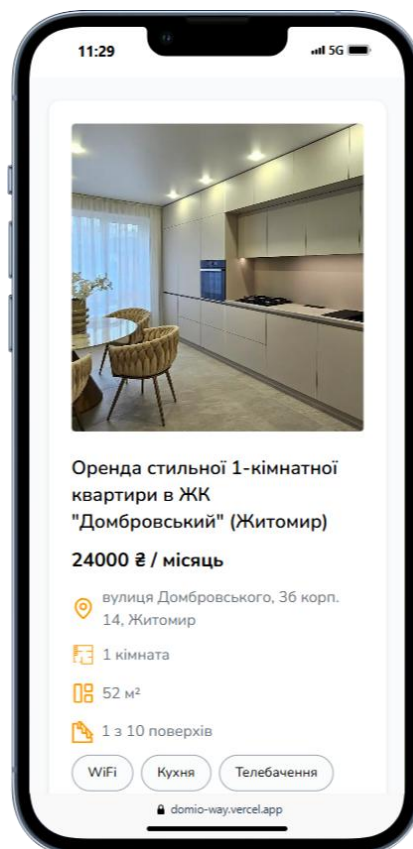


Рисунок 3.27 – Адаптивний дизайн головної сторінки

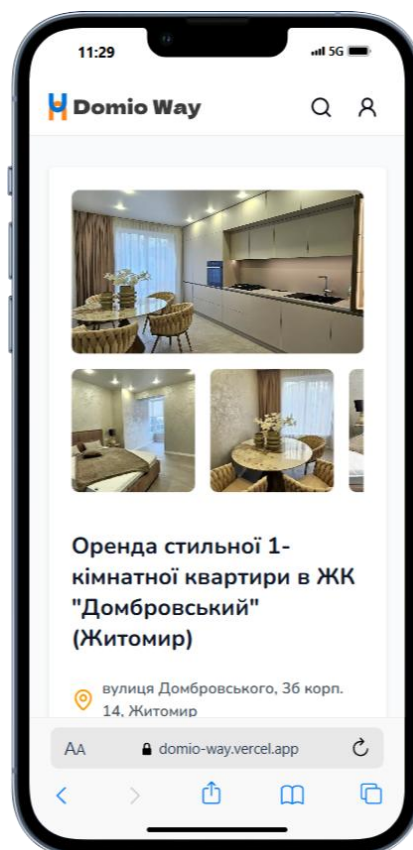


Рисунок 3.28 – Адаптивний дизайн сторінки оголошення

3.3 Тестування платформи

У процесі розробки веб-платформи особлива увага приділялася перевірці коректності її функціонування шляхом поетапного тестування з метою своєчасного виявлення логічних та технічних недоліків, що виникали під час інтеграції окремих компонентів системи.

Для виявлення потенційних збоїв у роботі веб-застосунку під час тестування застосовувались методики аналізу причинно-наслідкових зв'язків та передбачення помилок. Методика аналізу причинно-наслідкових зв'язків дозволяла встановити залежність між певними діями користувача та відповідною реакцією системи для точного визначення джерела виникнення помилки. Використання техніки передбачення помилок ідентифікує найбільш уразливі компоненти системи, в яких з високою ймовірністю могли виникнути помилки, що дозволило заздалегідь реалізовувати відповідні механізми обробки виняткових ситуацій.

У процесі виправлення виявлених дефектів системи проводилося повторне тестування, метою якого було підтвердження усунення помилки та перевірка стабільності відповідного функціоналу.

З метою збереження цілісності системи після інтеграції нових компонентів регулярно виконувалося регресійне тестування для виявлення можливих порушень в роботі раніше реалізованих модулів.

Після завершення етапу розробки було здійснено комплексне функціональне тестування для перевірки відповідності реалізованого функціоналу встановленим вимогам. Результати тестування продемонстрували коректну роботу основних сценаріїв взаємодії користувача з системою без критичних збоїв у виконанні її ключових операцій.

Додаткову увагу в процесі тестування було приділено графічному інтерфейсу користувача. Було здійснено перевірку коректності відображення елементів інтерфейсу, їхньої інтерактивності та відповідності встановленим вимогам. Поряд із цим проводилось юзабіліті-тестування для оцінки інтуїтивності, логічності та загальної зручності взаємодії користувача з системою.

Після завершення етапу ручного тестування було здійснено додаткову перевірку продуктивності веб-застосунку за допомогою онлайн-сервісу PageSpeed Insights [14], розробленого компанією Google. Даний інструмент призначений для комплексного аналізу веб-сторінок із метою оцінки їхньої продуктивності, доступності та відповідності сучасним веб-стандартам. Під час тестування PageSpeed Insights виконує моделювання завантаження сторінки на мобільних та десктопних пристроях, аналізує час відгуку, структуру DOM, ефективність використання ресурсів, а також дотримання рекомендацій щодо пошукової оптимізації. За результатами перевірки формується звіт, що містить загальні оцінки за кількома категоріями, виявлені проблеми та рекомендації щодо покращення продуктивності та взаємодії з користувачем. Для прикладу, було протестовано головну сторінку з відображенням оголошень (рис. 3.29), яка є однією з ключових для кінцевого користувача.

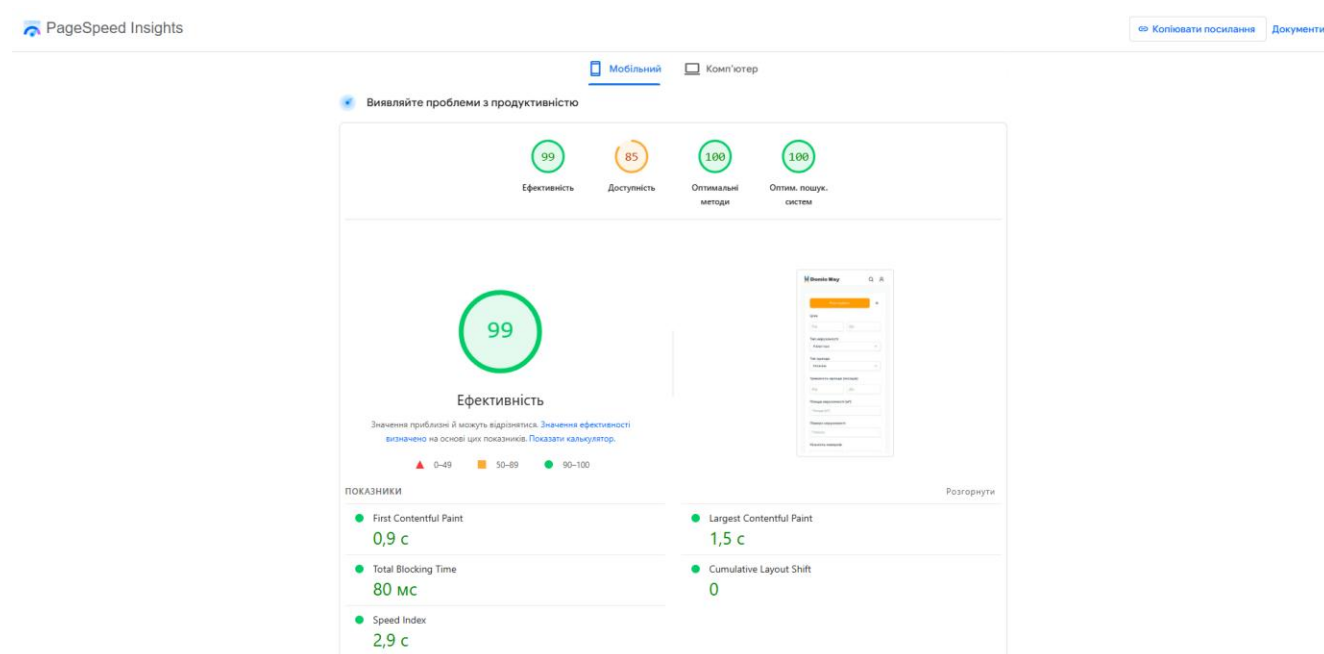


Рисунок 3.29 – Результат тестування сторінки для мобільного пристрою

Отримані результати показали високу ефективність роботи сторінки: показник ефективності становить 99 з 100, що вказує на швидке завантаження основного контенту та низького часу блокування. Максимальні бали у категоріях “Оптимальні методи” та “Оптимізація для пошукових систем” з дотриманням сучасних рекомендацій щодо розробки веб-застосунків. Показник доступності

складає 85, що вказує на наявність незначних аспектів, які потенційно можуть бути покращені для користувачів із особливими потребами.

Висновки до третього розділу

У процесі проектування та розгортання веб-застосунку було приділено особливу увагу вибору технологічної основи, яка забезпечує високу продуктивність, масштабованість та надійність функціонування системи. На основі порівняльного аналізу сучасних клієнтських і серверних платформ визначено оптимальні рішення для реалізації окремих компонентів додатку. Особлива увага зосереджена на сумісності обраних технологій, а також на забезпеченні безпеки даних та ефективної взаємодії між клієнтською і серверною частинами.

У роботі було детально описано структуру інтерфейсу та його елементи, які забезпечують ефективну взаємодію з системою. Описано функціональні можливості додатку залежно від ролі користувача. Інтерфейс було спроектовано з урахуванням принципів юзабіліті, доступності та адаптивності з забезпеченням зручності використання на різних пристроях та для користувачів з різноманітними потребами.

Було детально розглянуто процес тестування з метою виявлення потенційних недоліків у функціональності, продуктивності та доступності користувацького інтерфейсу. Основну увагу приділено практичному застосуванню сучасних інструментів автоматизованого тестування, зокрема сервісу Google PageSpeed Insights, який дає змогу об'єктивно оцінити ключові показники ефективності веб-сторінки.

ВИСНОВКИ

У межах даної кваліфікаційної роботи було спроектовано та реалізовано онлайн-платформу оренди житла, починаючи з комплексного дослідження предметної області та виявлення потреб різних категорій користувачів, а також зі здійснення порівняльного аналізу існуючих рішень на ринку. Отримані результати дозволили сформулювати набір функціональних і нефункціональних вимог, на підставі яких обґрунтовано обрану архітектуру, інструментарій розробки та технологічні стеки із урахуванням критеріїв продуктивності, масштабованості і безпеки.

У ході проектування платформи проведено системний аналіз сценаріїв взаємодії користувачів із продуктом та виконано детальне моделювання архітектури системи. Виявлено ключові компоненти й визначено їхні зв'язки з гарантуванням роботи всіх модулів. Було розроблено концептуальну та фізичну схеми бази даних для забезпечення ефективного обміну і зберігання інформації з дотриманням вимог цілісності й продуктивності.

У результаті було створено інтуїтивно зрозумілу та функціонально насичену онлайн-платформу оренди житла, яка забезпечує оперативний пошук і фільтрацію пропозицій, простий механізм бронювання та зручний інтерфейс управління оголошеннями й бронюваннями.

Інтерфейс платформи розроблено з урахуванням принципів простоти та логічної послідовності, що забезпечує швидку адаптацію нових користувачів і мінімізує час на виконання базових операцій. Використання клієнт-серверної архітектури та сучасних фреймворків гарантує стійкість системи до навантажень і високу швидкодію навіть при збільшенні кількості одночасних запитів. Інтеграція із зовнішніми сервісами платіжним шлюзом, картографічними й медіа-хостингом сприяють безперебійній роботі ключових функцій платформи. У сукупності ці рішення формують інноваційний продукт, що відповідає найсучаснішим вимогам безпеки, продуктивності та зручності, надаючи користувачам унікальну онлайн-платформу оренди житла.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. DimRia [Електронний ресурс] – Режим доступу до ресурсу: <https://dom.ria.com/uk/>.
2. Flatfy [Електронний ресурс] – Режим доступу до ресурсу: <https://www.typescriptlang.org/docs/>.
3. Booking.com [Електронний ресурс] – Режим доступу до ресурсу: <https://www.booking.com/>.
4. Node.js [Електронний ресурс] – Режим доступу до ресурсу: <https://nodejs.org/docs/latest/api/>.
5. TypeScript [Електронний ресурс] – Режим доступу до ресурсу: <https://www.typescriptlang.org/docs/>.
6. PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: <https://www.postgresql.org/>.
7. Prisma ORM [Електронний ресурс] – Режим доступу до ресурсу: <https://www.prisma.io/docs>.
8. Next.js [Електронний ресурс] – Режим доступу до ресурсу: <https://nextjs.org/docs>.
9. React.js [Електронний ресурс] – Режим доступу до ресурсу: <https://react.dev/learn>.
10. React Leaflet [Електронний ресурс] – Режим доступу до ресурсу: <https://react-leaflet.js.org/>.
11. LiqPay [Електронний ресурс] – Режим доступу до ресурсу: <https://www.liqpay.ua/doc>.
12. TailwindCSS [Електронний ресурс] – Режим доступу до ресурсу: <https://tailwindcss.com/>.
13. Vercel [Електронний ресурс] – Режим доступу до ресурсу: <https://vercel.com/docs>.
14. PageSpeedInsights [Електронний ресурс] – Режим доступу до ресурсу: <https://pagespeed.web.dev/>.

- 15.Casciaro M., Mammino L. Node.js Design Patterns: Design and implement production-grade Node.js applications using proven patterns and techniques. 3rd edition Packt Publishing, 2020. 660 p.
- 16.Obe R., Hsu L. PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database. 3rd edition вид. O'Reilly Media, 2017. 312 p.
- 17.Riva M. Real-World Next.js: Build scalable, high-performance, and modern web applications using Next.js, the React framework for production. Packt Publishing, 2022. 366 p.
- 18.Rescigno R. Tailwind CSS: a guide to using the popular utility-first CSS framework. Independently published, 2023. 42 p.
- 19.Vanderkam D. Effective TypeScript: 83 Specific Ways to Improve Your TypeScript. 2nd edition O'Reilly Media, 2024. 401 p.
- 20.Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. 2nd edition O'Reilly Media, 2020. 310 p.

ДОДАТКИ

					ІПЗ.КР.Б – 121 – 25 – ІПЗ	
						68

```

// api/cron/complete-finished-bookings/route.ts
import prisma from "@features/db";
import { getTotalDays, getTotalMonths } from "@features/utils";
import { NextRequest } from "next/server";

export async function POST(request: NextRequest) {
  const url = new URL(request.url);
  const secret = url.searchParams.get("secret");

  if (!secret || secret !== process.env.CRON_SECRET) {
    return new Response("Unauthorized", { status: 401 });
  }

  console.log("Running cron job to complete finished bookings...");

  const bookingsToComplete = await prisma.booking.findMany({
    where: {
      status: "confirmed",
    },
    include: {
      listing: true,
      payments: true,
    },
  });

  const currentDate = new Date();

  for (const booking of bookingsToComplete) {
    const startDate = new Date(booking.start_date);

    const totalPrice = booking.total_price;

    const totalPaid = booking.payments.reduce(
      (acc, payment) => acc + payment.amount,
      0
    );

    let endDate = new Date(booking.end_date);

    if (booking.listing.rental_duration_type === "monthly") {

```

```

    const maxMonths = getTotalMonths(booking.start_date,
booking.end_date);

    const paidMonths = (totalPaid * maxMonths) / totalPrice;

    if (paidMonths < maxMonths) {
        endDate = new Date(startDate.getTime());
        endDate.setMonth(endDate.getMonth() + Math.floor(paidMonths));
        endDate.setDate(endDate.getDate() + 4);
    }
} else if (booking.listing.rental_duration_type === "daily") {
    const maxDays = getTotalDays(booking.start_date, booking.end_date) ||
1;

    const paidDays = (totalPaid * maxDays) / totalPrice;

    if (paidDays < maxDays) {
        endDate = new Date(startDate.getTime());
        endDate.setDate(endDate.getDate() + Math.floor(paidDays) + 4);
    }
}

if (endDate.getTime() > currentDate.getTime()) continue;

await prisma.booking.update({
    where: { booking_id: booking.booking_id },
    data: { status: "completed" },
});

await prisma.listing.update({
    where: { listing_id: booking.listing_id },
    data: {
        status: "approved",
        is_published: true,
    },
});
}

return new Response("Cron job scheduled to complete finished bookings", {
    status: 200,
});
}

```

```

// api/cron/schedule-cancel-stale-bookings/route.ts
import prisma from "@features/db";
import { subDays } from "date-fns";
import { NextRequest } from "next/server";

export async function POST(request: NextRequest) {
  const url = new URL(request.url);
  const secret = url.searchParams.get("secret");

  if (!secret || secret !== process.env.CRON_SECRET) {
    return new Response("Unauthorized", { status: 401 });
  }

  console.log("Running cron job to cancel stale bookings...");

  const yesterday = subDays(new Date(), 1);

  const bookingsToCancel = await prisma.booking.findMany({
    where: {
      status: "confirmed",
      created_at: {
        lte: yesterday,
      },
      payments: {
        none: {},
      },
    },
    include: {
      listing: true,
    },
  });

  for (const booking of bookingsToCancel) {
    const createdHour = booking.created_at.getHours();
    const currentHour = new Date().getHours();

    if (createdHour > currentHour) continue;

    await prisma.booking.update({
      where: { booking_id: booking.booking_id },
      data: { status: "cancelled" },
    });
  }
}

```

```

    await prisma.listing.update({
      where: { listing_id: booking.listing_id },
      data: {
        status: "approved",
        is_published: true,
      },
    });

    console.log(`Cancelled booking with ID: ${booking.booking_id}`);
  }

  console.log("Cron job scheduled to cancel stale bookings");

  return new Response("Cron job scheduled to cancel stale bookings", {
    status: 200,
  });
}

```

```

// api/payments/liqpay/initiate/route.ts
import { NextResponse } from "next/server";
import { LiqPay } from "@lib/liqpay";
import prisma from "@features/db";
import { getTotalDays, getTotalMonths } from "@features/utils";

const liqPay = new LiqPay(
  process.env.LIQPAY_PUBLIC_KEY as string,
  process.env.LIQPAY_PRIVATE_KEY as string
);

export async function POST(request: Request) {
  const { bookingId, max_pay } = await request.json();

  const booking = await prisma.booking.findUnique({
    where: {
      booking_id: bookingId,
    },
    include: {
      listing: true,
      payments: true,
    },
  });

  if (!booking) {
    return NextResponse.json({ error: "Booking not found" }, { status:
404 });
  }

  const isDailyRental = booking.listing.rental_duration_type === "daily";

  const totalRentalPeriods = isDailyRental
    ? getTotalDays(booking.start_date, booking.end_date) || 1
    : getTotalMonths(
        booking?.start_date || new Date(),
        booking?.end_date || new Date()
      );

  const totalPrice = booking?.total_price || 1;
  const totalPaid =

```

```

    booking?.payments.reduce((acc, payment) => acc + payment.amount, 0) ||
0;

const rentalPeriods = (totalPaid * totalRentalPeriods) / totalPrice;

const isFullyPaid = rentalPeriods >= totalRentalPeriods;

if (isFullyPaid) {
    return NextResponse.json(
        { error: "All rental periods are already paid" },
        { status: 400 }
    );
}

const orderId = `${booking.booking_id}@${booking.payments.length}`;

const cnb = liqPay.cnbForm({
    action: "pay",
    amount: booking.listing.price * max_pay,
    currency: booking.listing.currency,
    description: `Оплата бронювання №${booking.booking_id.slice(0, 13)} |
    №${
        booking.payments.length + 1
    }`,
    order_id: orderId,
    version: 3,
    language: "uk",
    server_url: process.env.LIQPAY_CALLBACK_URL!,
    result_url: "http://localhost:3000/profile/bookings",
});

return new NextResponse(cnb, {
    headers: {
        "Content-Type": "text/html",
    },
});
}

```

```

// api/payments/liqpay/callback/route.ts
import { NextResponse } from "next/server";
import { LiqPay } from "@lib/liqpay";
import prisma from "@features/db";

const liqPay = new LiqPay(
  process.env.LIQPAY_PUBLIC_KEY as string,
  process.env.LIQPAY_PRIVATE_KEY as string
);

export async function POST(request: Request) {
  const formData = await request.formData();
  const data = formData.get("data")?.toString() || "";
  const signature = formData.get("signature")?.toString() || "";

  const isValidSignature = liqPay.verifySignature(data, signature);

  if (!isValidSignature) {
    return NextResponse.json({ error: "Invalid signature" }, { status:
403 });
  }

  const payload = JSON.parse(
    Buffer.from(data, "base64").toString("utf-8")
  ) as any;

  const { order_id, status, amount, currency, transaction_id } = payload;

  const bookingId = order_id.split("@")[0];

  const booking = await prisma.booking.findUnique({
    where: {
      booking_id: bookingId,
    },
    include: {
      listing: true,
    },
  });

  if (!booking) {
    return NextResponse.json({ error: "Booking not found" }, { status:
404 });
  }
}

```

```

}

await prisma.payment.create({
  data: {
    booking: {
      connect: { booking_id: bookingId },
    },
    payer: {
      connect: { user_id: booking.tenant_id },
    },
    amount: parseFloat(amount),
    currency: currency,
    transaction_id: String(transaction_id),
    status: status === "success" ? "completed" : "failed",
    paid_at: status === "success" ? new Date() : null,
  },
});

return NextResponse.json({ result: "ok" });
}

```