

**Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення**

**КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»**

**РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ ОРЕНДИ ТА БРОНЮВАННЯ
ЖИТЛА З ВИКОРИСТАННЯМ NEXT.JS ТА NODE.JS**

**DEVELOPMENT OF A WEB APPLICATION FOR RENTING AND
BOOKING ACCOMMODATION USING NEXT.JS AND NODE.JS**

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗ-41
Собіпан Арсен Володимирович

Керівник:
Ліщина Наталія Миколаївна

Кваліфікаційну роботу
допущено до захисту
«__» _____ 20__ р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

Луцьк – 2026 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних та інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Галузь знань: 12 «Інформаційні технології»

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітня програма: «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

«__» _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Собіпану Арсену Володимировичу

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи «Розробка вебзастосунку для оренди та бронювання житла з використанням Next.js та Node.js»

Керівник роботи: к.т.н., доцент Ліщина Наталія Миколаївна

затверджені наказом закладу вищої освіти від «31» грудня 2025 р. № 541/01-02

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра «29» травня 2026 р.

3. Вихідні дані до роботи: предметна область оренди житла та управління бронюваннями, технології розробки програмного забезпечення (TypeScript, Next.js, Node.js), засоби роботи з базами даних (PostgreSQL, Prisma ORM), хмарна платформа Supabase, засоби автентифікації (NextAuth.js), платіжні системи (Stripe), вимоги до інформаційних систем та забезпечення інформаційної безпеки..

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): аналіз сучасного стану ринку онлайн-оренди житла та існуючих рішень, постановка завдання, формування вимог до розроблюваної інформаційної системи, обґрунтування вибору технологій та засобів розробки, розробка структури бази даних, практична реалізація об'єкта проєктування, обґрунтування принципів захисту інформаційно-комп'ютерної системи, тестування та налагодження програмного забезпечення..

5. Перелік графічного матеріалу: 19 рисунків, 6 лістингів, 4 таблиці.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	Ліщина Н. М.		
Специфікація вимог до розробленої системи	Ліщина Н. М.		
Розробка об'єкта проектування	Ліщина Н. М.		
Нормоконтроль	Суринович О. М.		
Гарант ОП	Ліщина Н. М.		
Показник запозичень тексту	%		
Академічна доброчесність	Ліщина Н. М.		

7. Дата видачі завдання «3» лютого 2026 р.

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 14.02.2026 р.	
2	Аналіз проблеми розробки та впровадження об'єкту проектування	до 03.03.2026 р.	
3	Обґрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 14.03.2026 р.	
4	Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних	до 14.04.2026 р.	
5	Практична реалізація об'єкта проектування та розробка бази даних	до 20.04.2026 р.	
6	Тестування та налагодження об'єкта проектування	до 04.05.2026 р.	
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедрі	до 29.05.2026 р.	

Здобувач вищої освіти _____

Арсен СОБІПАН

Керівник кваліфікаційної роботи _____

Наталія ЛІЩИНА

АНОТАЦІЯ

Собіпан А. В. Розробка вебзастосунку для оренди та бронювання житла з використанням Next.js та Node.js. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2026.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел.

У першому розділі проведено дослідження предметної області онлайн-оренди житла, проаналізовано існуючі платформи та виявлено їхні сильні і слабкі сторони, а також обґрунтовано необхідність створення власної системи. У другому розділі сформувано вимоги до програмного продукту, здійснено проєктування системи із використанням UML-нотації та обґрунтовано вибір інструментальних засобів, методів і технологій розробки. У третьому розділі описано процес реалізації веб-додатку, розроблено структуру бази даних, розглянуто питання забезпечення захисту інформації, а також наведено результати тестування і налагодження програмного забезпечення. У висновках підсумовано отримані результати, підтверджено досягнення мети роботи та повне виконання поставлених завдань.

Ключові слова: веб-додаток, оренда житла, бронювання, автентифікація, рольовий контроль доступу, база даних, TypeScript, Next.js, Node.js, PostgreSQL, Prisma ORM, Supabase, Stripe.

ABSTRACT

Sobipan A. V. Development of a Web Application for Renting and Booking Accommodation Using Next.js and Node.js. Manuscript. Qualification work for bachelor's degree in Software Engineering, specialty "Software Engineering". Lutsk National Technical University. Lutsk, 2026.

The bachelor's qualification work consists of an introduction, three chapters, conclusions, a list of references.

In the first chapter examines the subject area of online housing rental, analyzes existing platforms and identifies their strengths and weaknesses, and justifies the need to create a proprietary system. The second chapter defines the requirements for the software product, designs the system using UML notation, and justifies the choice of development tools, methods, and technologies. The third chapter describes the process of implementing the web application, develops the database structure, addresses information security issues, and presents the results of software testing and debugging. The conclusions summarize the results obtained, confirm the achievement of the work's objectives, and confirm the full completion of the tasks set.

Keywords: web application, housing rental, booking, authentication, role-based access control, database, TypeScript, Next.js, Node.js, PostgreSQL, Prisma ORM, Supabase, Stripe.

ЗМІСТ

ВСТУП.....	7
РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ОНЛАЙН-ОРЕНДИ ТА БРОНЮВАННЯ ЖИТЛА І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ	9
1.1 Аналіз сучасного стану проблеми	9
1.2 Постановка завдання на кваліфікаційну роботу бакалавра	13
РОЗДІЛ 2 СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ...	15
2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проєктування програмного забезпечення	15
2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання ...	23
РОЗДІЛ 3 РОЗРОБКА ВЕБ-ДОДАТКУ ДЛЯ ОРЕНДИ ТА БРОНЮВАННЯ ЖИТЛА.....	29
3.1 Практична реалізація об'єкта проєктування.....	29
3.2 Розробка бази даних	36
3.3 Захист інформаційно-комп'ютерної системи.....	39
3.4 Тестування та налагодження програмного забезпечення	42
ВИСНОВКИ	45
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	47

ВСТУП

Актуальність теми. Ринок короткострокової оренди житла за останні десять років суттєво змінився. Airbnb, Booking.com та подібні платформи довели, що модель peer-to-peer оренди працює і масштабується. Водночас більшість таких платформ або занадто складні для невеликих господарів, або беруть значну комісію, або просто недоступні на певних ринках. Це створює простір для спеціалізованих рішень.

Процес оренди житла традиційно супроводжується низкою організаційних проблем. Господарі ведуть облік бронювань вручну, у таблицях або через месенджери. Гості не мають зручного способу порівняти варіанти, перевірити доступність та сплатити онлайн в одному місці. Відсутність централізованої системи призводить до накладок у бронюваннях, втрати комунікації та складнощів з оплатою.

Готові рішення або орієнтовані на великих гравців з тисячами об'єктів, або є надто дорогими для окремого господаря з кількома квартирами. Вбудовані механізми автентифікації, управління бронюваннями та онлайн-оплати в таких системах або відсутні, або реалізовані як окремі сервіси без інтеграції між собою.

Розробка власної платформи дозволяє вирішити ці проблеми комплексно. Веб-додаток StayHub реалізує повний цикл: від реєстрації господаря та публікації оголошення до бронювання гостем і проведення оплати через Stripe. Система підтримує рольову модель доступу (користувач та адміністратор), що дає змогу розмежувати функціональність для різних категорій користувачів.

Метою кваліфікаційної роботи є розробка веб-додатку для оренди та бронювання житла з використанням Next.js, Node.js і хмарної бази даних PostgreSQL через Supabase.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- проаналізувати існуючі платформи оренди житла та їхні технічні рішення;
- визначити вимоги до програмного продукту;

- обґрунтувати вибір технологій розробки;
- спроектувати структуру бази даних та архітектуру системи;
- реалізувати програмний продукт;
- забезпечити механізми захисту інформації та автентифікації;
- провести тестування та налагодження системи.

Об'єктом дослідження є процес оренди та бронювання житла між приватними особами через веб-платформу.

Предметом дослідження є методи та засоби розробки веб-додатків з використанням Next.js, TypeScript, Prisma ORM та хмарних сервісів.

Практичне значення роботи полягає у створенні працюючого веб-додатку, який може використовуватись господарями для публікації оголошень і прийому бронювань з онлайн-оплатою.

РОЗДІЛ 1

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ОНЛАЙН-ОРЕНДИ ТА БРОНЮВАННЯ ЖИТЛА І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Ринок короткострокової оренди житла за останні роки виріс суттєво. У 2024 році його обсяг становив 95,11 млрд дол. США, а до 2033 року очікується зростання до 201,51 млрд дол. при середньорічному темпі 8,7 % [1]. Онлайн-бронювання поки що поступається офлайн-каналу за часткою, але зростає швидше за все інше. Частки ринку короткострокової оренди житла у 2024 році наведено в рисунку 1.1.

Частки ринку короткострокової оренди житла (2024 р.)

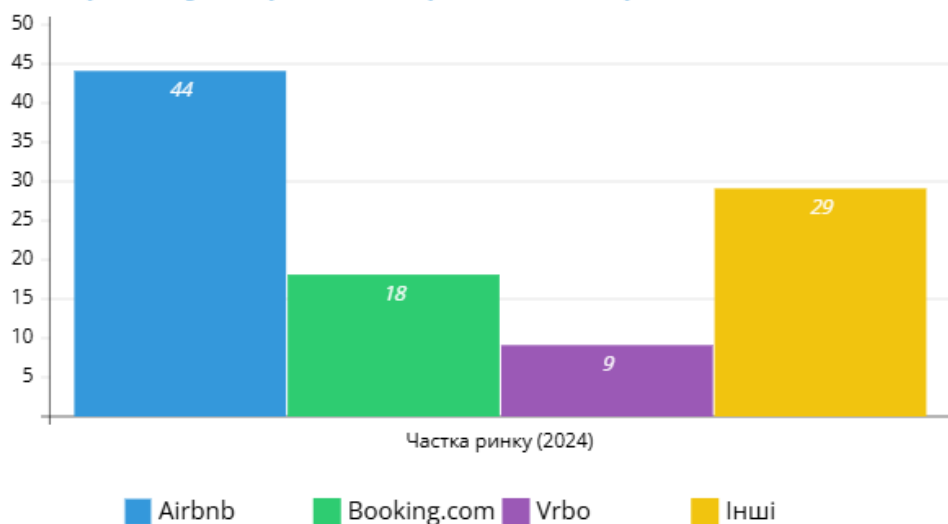


Рисунок 1.1 – Частки ринку короткострокової оренди житла у 2024 році

Мандрівники, особливо молодша аудиторія, дедалі частіше обирають орендоване житло замість готелів – через ціну і відчуття локальності. Пандемія 2020 року пришвидшила цей перехід: люди звикли бронювати дистанційно і платити онлайн. Господарі, своєю чергою, масово виходять на платформи, бо це простіше, ніж працювати через агентства нерухомості.

Науковий інтерес до P2P-платформ оренди також зростає. У статті «Attributes influencing peer-to-peer accommodation performance», опублікованій у журналі Tourism Management у 2025 році, автори систематизують фактори, що визначають ефективність таких платформ до і після пандемії [2]. Дослідження фіксує зростання вимог користувачів до безпеки транзакцій, прозорості комісій і якості верифікації – саме ті аспекти, де більшість існуючих рішень має слабкі місця.

Для українського ринку характерна своя специфіка. Через повномасштабне вторгнення відбулось суттєве перерозподілення попиту: схід втратив орендарів, тоді як Львів, Ужгород і Тернопіль отримали значний приплив мешканців з інших регіонів. Попит на зручні онлайн-інструменти бронювання при цьому лише зростає, оскільки орендарі шукають житло дистанційно.

Ринок фактично поділений між кількома великими гравцями. Airbnb у 2024 році контролював 44 % сегменту короткострокової оренди, мав понад 8 млн активних оголошень у 220 країнах і зафіксував 491 млн заброньованих ночей [3]. З технічного боку платформа побудована на мікросервісній архітектурі: Java і Kotlin на бекенді, React на фронтенді, PostgreSQL і Cassandra для зберігання даних, Apache Kafka для обробки подій. Інтерфейс платформи Airbnb наведено на рисунку 1.2.

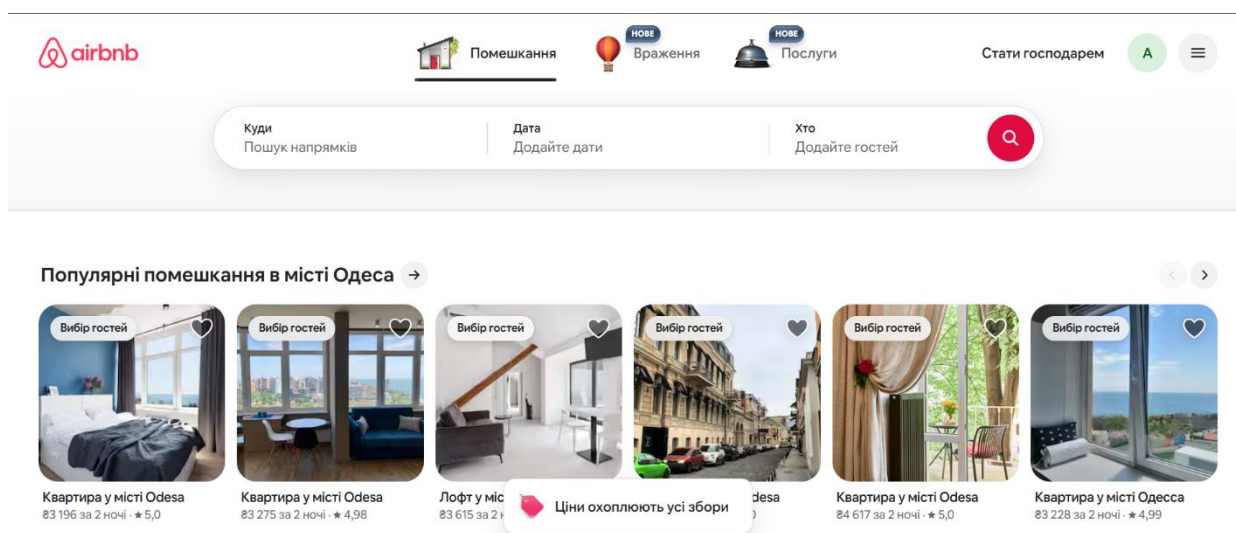


Рисунок 1.2 – Інтерфейс платформи Airbnb [4]

Airbnb це інфраструктура, що формувалась роками. Комісія для орендодавців – від 3 % при розподіленій моделі до 14-16 % при моделі «тільки для господаря».

Booking.com орієнтований переважно на готельний сегмент, але активно нарощує частку в оренді приватного житла – понад 7 млн приватних об'єктів з 31 млн загалом. Комісія коливається від 10 % до 25 % залежно від локації. Реєстрація об'єкта передбачає відповідність готельним стандартам і тривалу верифікацію, що для приватного власника однієї-двох квартир часто є непоборним бар'єром.

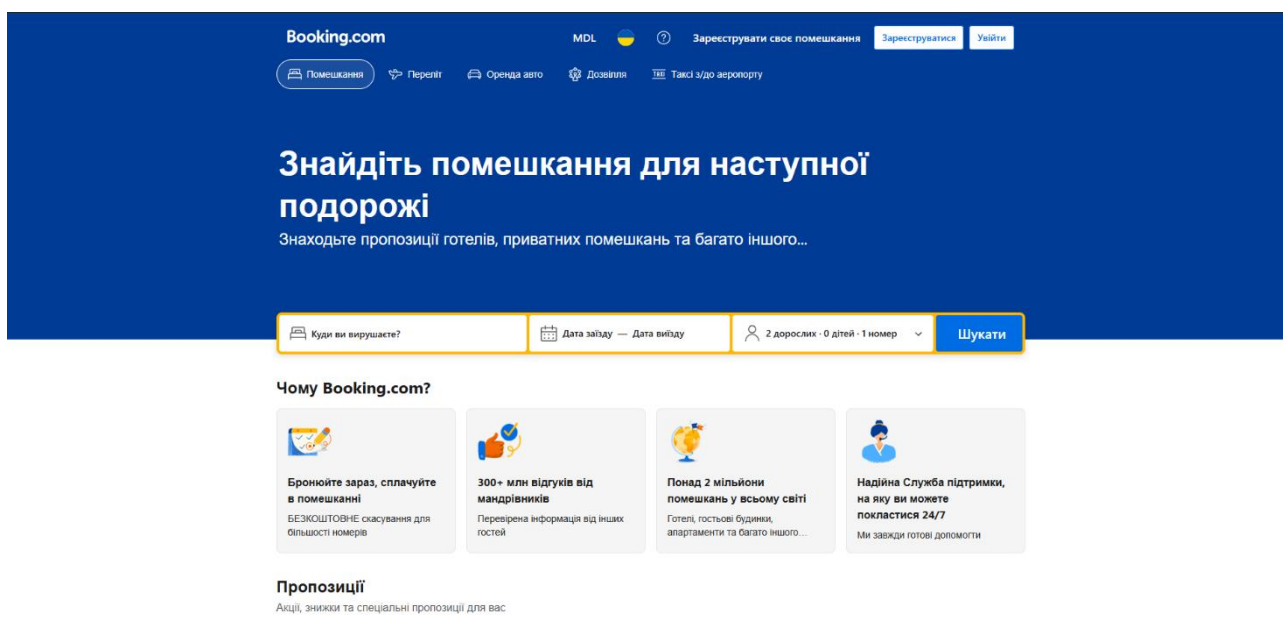


Рисунок 1.3 – Інтерфейс платформи Booking.com [5]

Vrbo орієнтований на оренду цілих будинків для сімейного відпочинку і не передбачає оренди окремих кімнат. Комісія – 5-8 %, одна з найнижчих серед великих платформ, але географічне охоплення обмежене переважно американським ринком.

OLX і Flatfy – основні представники вітчизняного ринку. Обидва надають базовий функціонал публікації оголошень, але без вбудованого бронювання і захищеної оплати. Угода укладається напрямку між сторонами поза межами

платформи – без гарантій, без механізму вирішення спорів, без автоматичного контролю дат. Розділ оренди житла на платформі OLX наведено на рисунку 1.4.

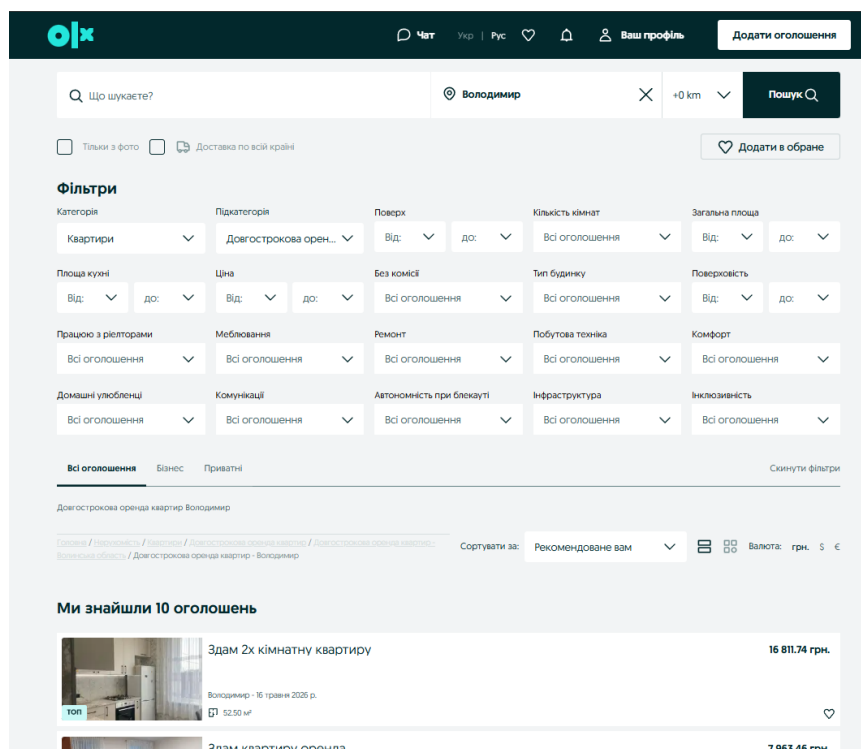


Рисунок 1.4 – Інтерфейс сервісу OLX (розділ оренда житла) [6]

Огляд інтерфейсів платформ показує, що підходи до організації пошуку у кожної з них різні. Airbnb робить ставку на фото і детальні профілі об’єктів, Booking.com – на швидкий пошук з великою кількістю фільтрів, OLX пропонує простий список оголошень без додаткової функціональності. Зведений порівняльний аналіз платформ за основними критеріями наведено в таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз існуючих платформ оренди житла

Платформа	Онлайн-бронювання	Захищена оплата	Комісія для господаря	Доступність для приватних орендодавців
Airbnb	Так	Так	3-16 %	Висока
Booking.com	Так	Так	10-25 %	Низька
Vrbo	Так	Так	5-8 %	Середня
OLX	Ні	Ні	0 %	Висока
Flatfy	Ні	Ні	0 %	Висока
StayHub	Так	Так	Прозора	Висока

Великі платформи або занадто дорогі для господаря, або закриті для нових учасників через складну верифікацію. Вітчизняні рішення не реалізують повного циклу орендної операції. Розроблювана система закриває цю нішу: господар отримує інструмент для публікації оголошень і управління бронюваннями, гість – зручний пошук з онлайн-оплатою через Stripe.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Метою кваліфікаційної роботи є проектування та реалізація веб-додатку для пошуку і бронювання житла з підтримкою онлайн-оплати, рольової моделі доступу та керування оголошеннями.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- проаналізувати існуючі платформи оренди житла, визначити їхні переваги і недоліки та обґрунтувати доцільність розробки власної системи;
- сформулювати вимоги до програмного продукту з урахуванням потреб двох категорій користувачів – господарів і гостей – та визначити функціональні межі системи;
- обґрунтувати вибір технологій розробки (Next.js і TypeScript для реалізації клієнтської та серверної частин, PostgreSQL і Prisma ORM для роботи з базою даних, Supabase для хмарного зберігання файлів, NextAuth.js для автентифікації та Stripe для обробки платежів);
- спроектувати структуру бази даних з урахуванням зв'язків між сутностями: користувачами, оголошеннями, бронюваннями та відгуками;
- реалізувати програмний продукт, що включає публічну частину з пошуком і фільтрацією оголошень, особистий кабінет господаря з управлінням оголошеннями і бронюваннями, систему онлайн-оплати через Stripe та адміністративну панель;
- забезпечити захист інформації шляхом реалізації рольового контролю доступу з користувача і адміністратора, захищеної автентифікації на основі JWT-токенів та валідації вхідних даних на рівні API;

– провести тестування програмного продукту, перевірити коректність роботи основних сценаріїв використання та усунути виявлені недоліки.

Результатом виконання кваліфікаційної роботи є повнофункціональний веб-додаток StayHub, розгорнутий у хмарному середовищі, з підтвердженою працездатністю всіх реалізованих модулів.

РОЗДІЛ 2

СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проєктування програмного забезпечення

Перед тим як переходити до технічних рішень, необхідно визначити, що саме система має робити і для кого. Вимоги до StayHub формуються трьома методами. Перший – аналіз існуючих платформ (Airbnb, Booking.com, OLX), проведений у підрозділі 1.1. Він дозволить виокремити обов’язковий набір функцій: пошук за датами і локацією, підтвердження бронювань, онлайн-оплата, відгуки після проживання. Другий – складання сценаріїв використання для кожного типу користувача. Це дає змогу визначити межі системи і виявити неочевидні ситуації – наприклад, що буде відбуватись при спробі забронювати вже зайнятий об’єкт або коли платіж не проходить. Третій – перевірка відповідності стандартному переліку функціональних характеристик інформаційних систем: введення даних, їх зміна, архівація, пошук, розмежування прав доступу, захист інформації, контроль введення, цілісність бази. Кожна характеристика оцінюється на практичну доцільність саме для сервісу оренди.

Для моделювання системи обрано об’єктно-орієнтований підхід із нотацією UML. StayHub складається з чітко відокремлених сутностей зі стійкими зв’язками між ними – користувач, оголошення, бронювання, платіж, відгук. UML природно відображає таку структуру. Структурний підхід на основі діаграм потоків даних більше підходить для систем із безперервною обробкою інформації, тоді як для транзакційного сервісу важливіші стани об’єктів і взаємодії між акторами. Для різних аспектів системи використані різні типи діаграм: use case – для функціональних вимог з позиції користувача, діаграма класів – для структури даних і зв’язків між сутностями, діаграма послідовності – для ключових сценаріїв взаємодії.

Головна мета системи – платформа, де власники нерухомості публікуватимуть оголошення, а орендарі знаходитимуть і бронюватимуть житло з оплатою онлайн. З цього випливає перелік конкретних задач:

- реєстрація і автентифікація користувачів двома способами (через обліковий запис Google і через email з паролем);
- розмежування прав доступу за двома ролями – користувач (USER), адміністратор (ADMIN);
- пошук оголошень за містом, типом нерухомості, датами, кількістю гостей і ціновим діапазоном з автоматичною перевіркою доступності;
- бронювання житла з розрахунком вартості і переходом до онлайн-оплати;
- збереження оголошень до списку бажань;
- управління власними оголошеннями;
- модерація нових оголошень адміністратором;
- підтвердження або відхилення бронювань власником оголошення;
- оцінювання проживання орендарем після його завершення;
- адміністрування – статистика, управління користувачами і контентом.

База даних наповнюється двома шляхами. Перший – реєстрація користувача, тобто введені дані пройдуть валідацію і збережуться як новий запис. Другий – публікація оголошення, іншими словами орендодавець заповнить форму з характеристиками об'єкта і завантажить фотографії, після чого всі поля перевіряється перед записом. Оновлення існуючих даних охоплює кілька сценаріїв. Орендодавець редагує або деактивує своє оголошення, адміністратор може змінювати роль користувача, статус бронювання оновлюється при підтвердженні, відхиленні, скасуванні або завершенні. Замість фізичного видалення оголошення переводяться у неактивний стан – це збереже пов'язану з ними історію бронювань і відгуків та дозволить при потребі відновити об'єкт без втрати даних.

Пошук передає параметри на сервер через HTTP GET. Сервер формує запит до бази з умовами фільтрації і повертає результати з пагінацією. Доступність дат визначається через перевірку перетину запитуваного діапазону з існуючими підтвердженими і очікуваними бронюваннями. Вся передача даних між клієнтом і сервером відбувається за протоколом HTTPS, що шифрує трафік і унеможливорює перехоплення даних у мережі. Паролі зберігаються виключно у хешованому вигляді, сесія підтримується через підписаний токен, а платіжна інформація не торкається сервера додатку – вона передає напряму до платіжного сервісу. Права доступу розмежовуватимуться за двома ролями:

- USER може шукати оголошення, зберігати їх у список бажань, бронювати, публікувати власні оголошення, керувати ними і підтверджувати бронювання;

- ADMIN має повний доступ, може модерувати оголошення і заблокувати будь-якого користувача.

Захист маршрутів діє на рівні серверного проксі – запити до захищених розділів без токена з потрібною роллю відхиляються до виконання будь-якої бізнес-логіки. Валідація введених даних відбувається у два етапи: на стороні клієнта для негайного зворотного зв'язку і на стороні сервера як остаточна незалежна перевірка. Зв'язки між таблицями бази визначаються через зовнішні ключі, а транзакційні операції виконуються в рамках єдиної транзакції, що унеможливить часткове збереження при помилці. Результати пошуку відображаються у вигляді сітки карток із фото, назвою, містом, ціною за ніч і середнім рейтингом, детальна сторінка надаватиме повний опис і список відгуків; власні бронювання та оголошення будуть доступні в особистому кабінеті.

Вимоги до системи зведені у таблицю 2.1 у форматі «подія-відгук», де для кожного сценарію вказано джерело події і реакцію системи.

Таблиця 2.1 – Специфікація вимог у форматі «подія-відгук»

№	Подія	Актор	Відгук системи
1	Реєстрація через email/пароль	Неzareєстрований відвідувач	Валідація даних, хешування пароля, створення облікового запису, відкриття сесії
2	Вхід через Google OAuth	Будь-який користувач	Автентифікація через зовнішній провайдер, створення або оновлення зв'язку з обліковим записом, відкриття сесії
3	Пошук оголошень	USER	Фільтрація за параметрами, перевірка перетину дат із бронюваннями, повернення пагінованого результату
4	Перегляд оголошення	Будь-який користувач	Повернення даних об'єкта разом зі списком відгуків і середнім рейтингом
5	Спроба бронювання зайнятого об'єкта	USER	Повідомлення про недоступність на обрані дати, пропозиція змінити параметри
6	Бронювання доступного житла	USER	Перевірка доступності, розрахунок вартості, збереження заявки, перехід до оплати
7	Успішна оплата	Платіжний сервіс	Верифікація підпису запиту, підтвердження платежу, зміна статусу бронювання на «підтверджено»
8	Неуспішна оплата	Платіжний сервіс	Верифікація підпису, зміна статусу на «оплата не пройшла», повідомлення орендаря
9	Публікація оголошення	USER	Валідація полів форми, збереження даних об'єкта, публікація в каталозі
10	Редагування або деактивація оголошення	USER	Перевірка права власності на об'єкт, оновлення або зміна статусу isPublished
11	Підтвердження або відхилення бронювання	USER	Перевірка права власності на об'єкт, зміна статусу бронювання
12	Написання відгуку	USER після завершення бронювання	Перевірка статусу бронювання, збереження відгуку, перерахунок середнього рейтингу оголошення
13	Зміна ролі користувача або блокування	ADMIN	Оновлення ролі або прапорця isBlocked, зміна прав при наступній сесії
14	Збереження оголошення у список	USER	Створення або видалення запису SavedListing, оновлення стану кнопки

Перелічені задачі зручно відобразити графічно – діаграма прецедентів одразу показує, хто що зможе робити і де між діями є залежності. Ролі є ієрархічні, тобто USER успадковує можливості Visitor, ADMIN – все що має USER. Деякі прецеденти нерозривно пов'язані, тобто бронювання без перевірки

доступності не має сенсу, тому перевірка є його обов’язковою складовою. Відгук інший – він не є частиною бронювання, а стане доступним лише після його завершення. Діаграму прецедентів наведено на рисунку 2.1.

Система розраховується на три типи користувачів. Незареєстрований відвідувач може переглядати каталог і детальні сторінки, зареєструватись або ввійти. USER отримає доступ до решти: пошук з фільтрами, список бажань, бронювання з оплатою, перегляд і скасування власних бронювань, відгуки, публікація й редагування оголошень, керування статусами бронювань. ADMIN має все те саме і ще модерацію нових оголошень, управління користувачами та доступ до статистики. Прецедент «Бронювання» включає «Перевірку доступності» і «Перехід до оплати» як обов’язкові складові. «Написання відгуку» розширює «Перегляд бронювань» і доступне лише для завершених бронювань.

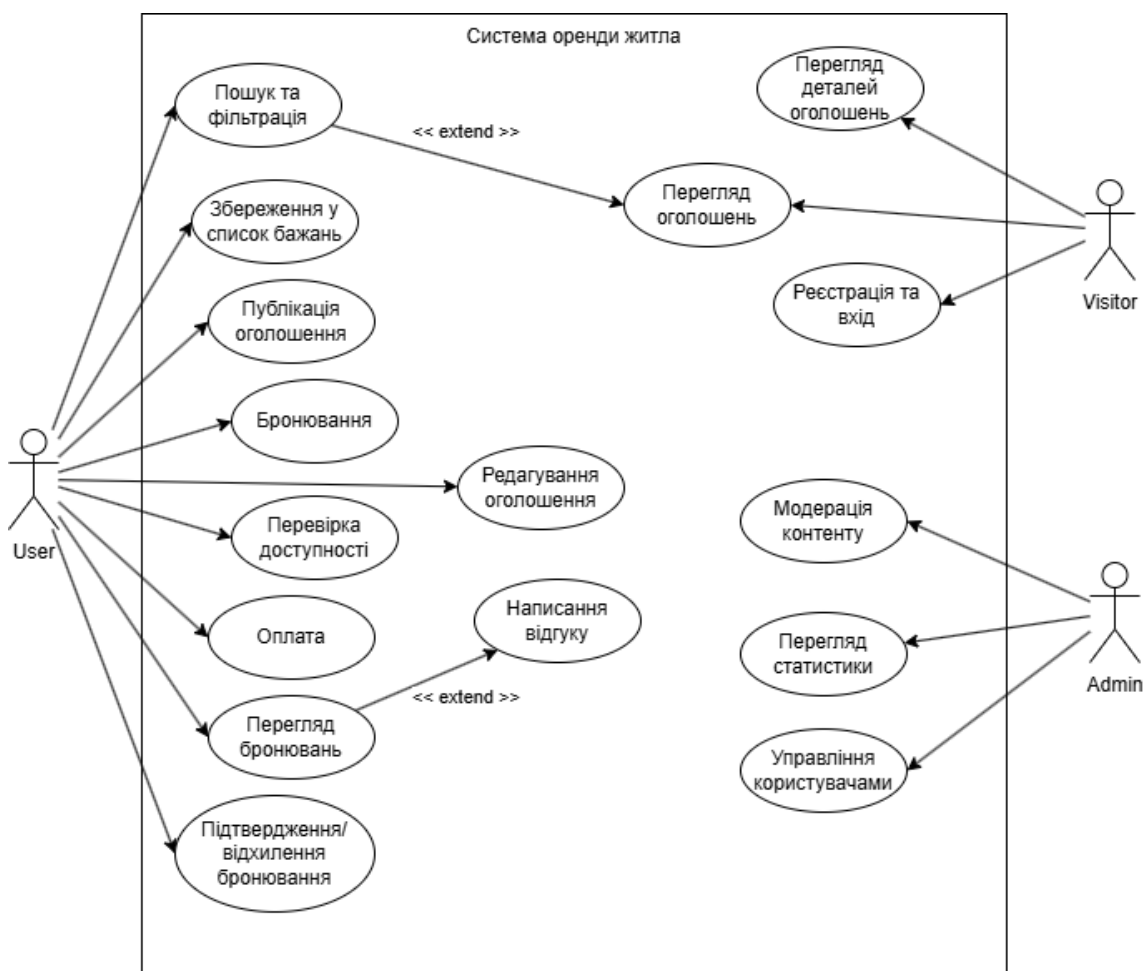


Рисунок 2.1 – Діаграма прецедентів системи StayHub

Взаємодії між користувачами описують що система робитиме, але не як організовані її дані. Кожен прецедент з діаграми спирається на конкретні сутності – користувача, оголошення, бронювання, платіж, відгук. Між ними будуть чіткі зв'язки, одне оголошення матиме багато бронювань, кожне бронювання породжує рівно один платіж, а відгук прив'язується до конкретного проживання, а не просто до оголошення.

Центральна сутність – клас Listing. Один орендодавець має кілька оголошень – відношення «один до багатьох» через поле hostId. Клас Booking з'єднує оголошення з орендарем через два зовнішні ключі: listingId і guestId. З кожним бронюванням пов'язується рівно один клас Payment – відношення «один до одного». Клас Review прив'язується до оголошення, користувача і конкретного бронювання, що унеможливорює повторний відгук на те саме проживання. Структуру цих сутностей зображено графічно на рисунку 2.2.

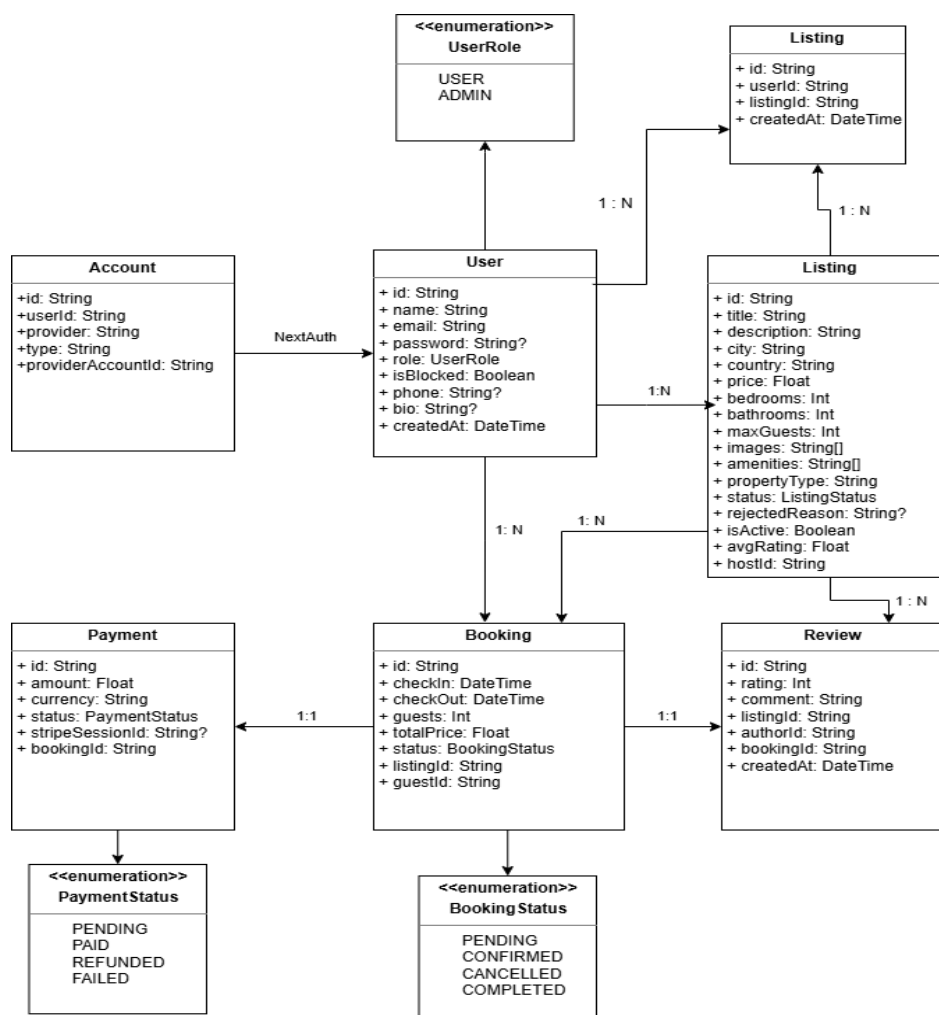


Рисунок 2.2 – Діаграма класів системи StayHub

Діаграма класів показує структуру даних, але не розкриває як система поводить себе в конкретній ситуації. Для цього використовують діаграму послідовності – вона відображає обмін повідомленнями між учасниками в хронологічному порядку. Як приклад обрано сценарій бронювання житла: він залучає браузер, сервер, базу даних і зовнішній платіжний сервіс, тому добре показує як рівні системи взаємодіють між собою. Діаграму послідовності наведено на рисунку 2.3.

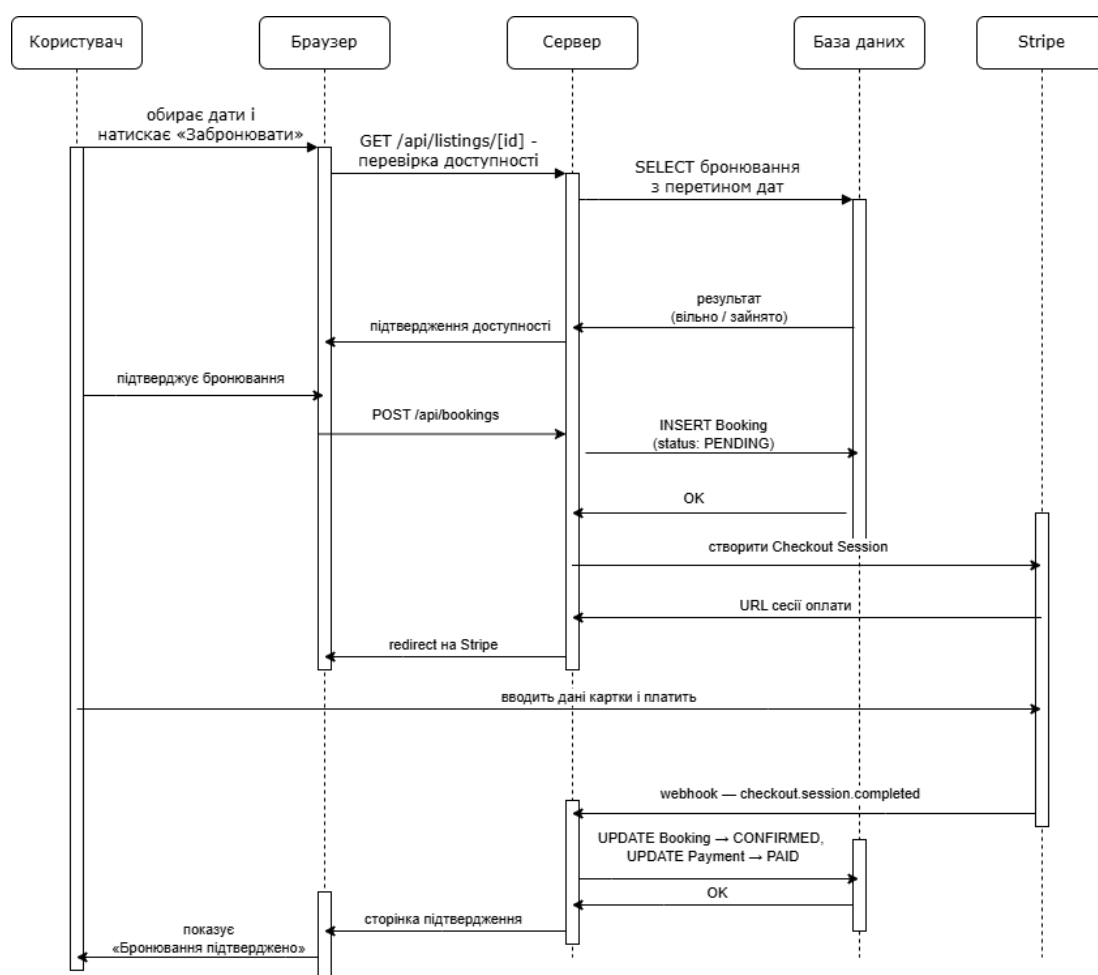


Рисунок 2.3 – Діаграма послідовності системи StayHub

На цій діаграмі бачимо, як система обмінюється даними між трьома рівнями: браузером, сервером веб-додатку і сховищами. Між браузером і сервером усі запити передаються через HTTPS: запити на читання використовують метод GET із параметрами у рядку адреси, запити на запис – POST або PUT із даними у тілі запиту у форматі JSON. Між сервером і базою

даних запити формуються через ORM-прошарок, що перетворює об'єктні операції на SQL; з'єднання підтримується через пул, що дозволяє обробляти кілька одночасних запитів без накладних витрат на встановлення нового з'єднання кожного разу. Фотографії оголошень завантажуються безпосередньо до хмарного сховища – сервер повертає клієнту тільки публічне посилання, а браузер завантажує зображення зі сховища напряму. Після обробки платежу зовнішній сервіс ініціює зворотний HTTP-запит на адресу сервера – цей потік є асинхронним і не залежить від дій користувача в браузері.

UX/UI-проектування починається із визначення того, які дані кому потрібні і в який момент. Visitor отримує каталог і детальні сторінки без реєстрації – це мінімум для оцінки пропозиції. Авторизований USER працює з двома потоками одночасно, а саме як той, хто шукає житло, і як той, хто його публікує. Обидві потреби обслуговує один кабінет без розділення на ролі, з єдиним входом. ADMIN бачитиме додатковий шар – черга модерації, список користувачів, статистика. Схеми інтерфейсів проектується так, щоб скоротити кількість кроків між наміром і дією, тобто форма бронювання розміщується прямо на сторінці оголошення, фільтри каталогу поруч із результатами, а не в окремому модальному вікні. На мобільних пристроях фільтри ховаються у висувну панель, сітка оголошень перебудовується в одну колонку, а форма бронювання займає всю ширину екрана.

Кольорова схема – кремовий фон, темно-зелений текст, помаранчевий колір на кнопках і СТА. Для заголовків обрано Instrument Serif, для тіла – Geist. Ця схема дасть відчуття ближче до журнального видання. Головна сторінка має пошуковий блок із вибором міста, дат і гостей, добірку популярних міст і оголошення за категоріями. Каталог – сітка карток із бічною панеллю фільтрів. Детальна сторінка – галерея, опис, зручності, календар доступності і форма бронювання. На мобільних фільтри ховається у панель, сітка перебудовується в одну колонку.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Вибір технологічного стеку здійснювався з урахуванням кількох практичних вимог одночасно. Продуктивність при серверному рендерингу, підтримка реляційних даних, безпека платіжних операцій і можливість розгортання без складної інфраструктури – всі ці чинники розглядалися разом, оскільки окремо взяті технології добрі лише тоді, коли добре взаємодіють між собою.

Як основну мову реалізації обрано TypeScript. Порівняно з чистим JavaScript він додає статичну типізацію, тобто помилки типів виявляються на етапі компіляції, а не в момент, коли користувач вже натиснув кнопку оплати. У системі з кількома взаємопов'язаними сутностями, зокрема оголошеннями, бронюваннями, платежами і відгуками, ця особливість суттєво зменшує ризик помилок при передачі даних між шарами застосунку. Середовищем виконання є Node.js, що дозволяє використовувати TypeScript як для клієнтської, так і для серверної частини в одному проєкті.

Питання вибору фреймворку виявилось менш однозначним. Серед розглянутих варіантів були React як бібліотека без вбудованих серверних можливостей, Nuxt.js, Remix і Next.js. React без додаткового налаштування не надає серверного рендерингу. Nuxt.js є зрілим варіантом, але орієнтований на Vue-екосистему, а отже менш природним при наявному досвіді з React. Remix пропонує цікаву модель завантаження даних, проте має вужчу спільноту і скромнішу документацію.

Зрештою обрано Next.js 16 з App Router. Серверні компоненти дозволяють звертатися до бази даних прямо у компонентах, де це доцільно, без введення окремого API-шару. Файлова маршрутизація спрощує підтримку великої кількості сторінок, а вбудовані API Routes дозволяють розмістити серверну логіку в тому самому репозиторії без окремого Node.js-сервера. Порівняльний аналіз фреймворків наведено у таблиці 2.2.

Таблиця 2.2 – Порівняльний аналіз фреймворків

Критерій	React	Remix	Nuxt.js	Next.js
Серверний рендеринг	Потребує налаштування	Так	Так	Так
Серверні компоненти	Ні	Частково	Ні	Так
Файлова маршрутизація	Ні	Так	Так	Так
Розмір спільноти	Дуже великий	Середній	Великий	Дуже великий
API Routes	Ні	Так	Так	Так

Для зберігання даних розглядалися MongoDB, MySQL і PostgreSQL. MongoDB з гнучкою схемою зручний для прототипів з нестабільною структурою. Але система бронювання вимагає чітких реляційних зв'язків між сутностями: одне бронювання прив'язане до конкретного оголошення і платежу, і будь-яке порушення цих зв'язків призводить до фінансових розбіжностей. Реляційна модель тут обґрунтованіша.

З реляційних варіантів перевагу надано PostgreSQL перед MySQL. Він підтримує складніші типи даних, зокрема JSON-поля, масиви і enum-типи, має кращу підтримку повнотекстового пошуку і нативно інтегрується з платформою Supabase, яку обрано для хостингу. Для взаємодії з PostgreSQL із TypeScript-коду обрано Prisma ORM. Альтернативою розглядався TypeORM, проте Prisma пропонує декларативну схему у власному форматі, автоматичну генерацію типізованого клієнта і вбудований Prisma Studio, що пришвидшує роботу під час налагодження.

Вибір PostgreSQL зумовив і вибір хмарної платформи. Firebase з Firestore є документоорієнтованою СУБД, що не відповідає обраній моделі. AWS Amplify надає широкі можливості, але налаштування займає значно більше часу, ніж виправдовує масштаб проєкту. Supabase побудовано поверх PostgreSQL і надає SQL-доступ, REST API і окремий сервіс для файлового зберігання. Завдяки цьому одна платформа охоплює і базу даних, і сховище фотографій оголошень.

Окремо розглядалася система автентифікації, оскільки додаток має підтримувати вхід через Google OAuth і через email з паролем. Auth0 і Clerk є рішеннями з готовими компонентами входу, але обидва вводять залежність від

стороннього сервісу і мають платні тарифи за певного обсягу користувачів. NextAuth є відкритим рішенням, інтегрованим з Next.js. Він підтримує JWT-сесії, провайдери OAuth і Credentials та зберігає дані у вже наявній базі даних.

Останньою з ключових складових стала платіжна система. Серед розглянутих варіантів PayPal має складнішу API для вбудованих форм, LiqPay орієнтований на українських провайдерів і менш зручний для тестування. Stripe обрано через детальну документацію, sandbox-середовище з тестовими картками і підтримку webhook-подій для підтвердження оплати. Готова Checkout Session переносить відповідальність за PCI-сумісність на сторону Stripe, що спрощує архітектуру серверної частини. Архітектурний зв'язок між усіма компонентами системи відображено на рисунку 2.4.

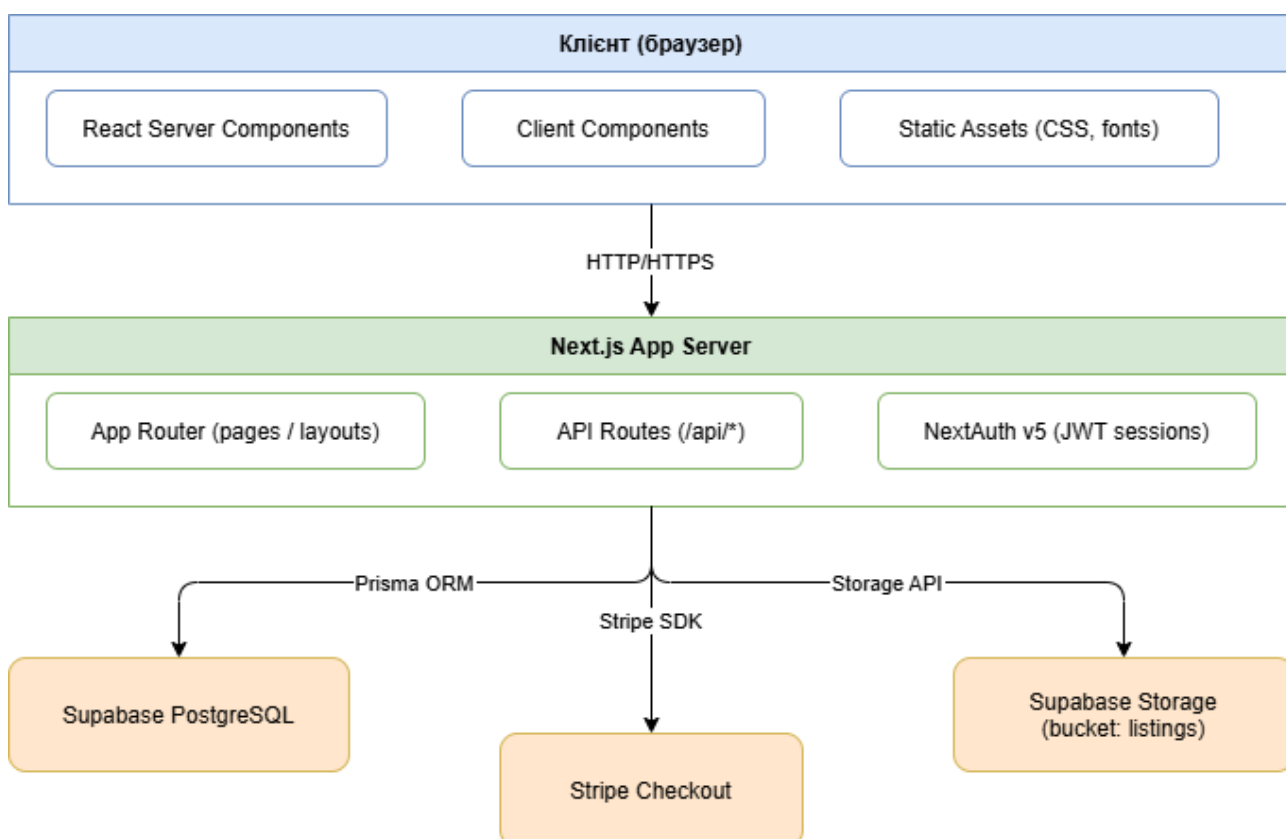


Рисунок 2.4 – Діаграма компонентів системи StayHub

Діаграма показує статичну структуру системи, але розуміти її повністю можна лише через те, як компоненти взаємодіють під час реальних операцій. Запити від клієнта проходять через App Router, де серверні компоненти

звертаються до бази даних через Prisma ORM, а захищені дії перевіряються через NextAuth. Зовнішні сервіси підключаються лише на конкретних етапах: Stripe при оформленні бронювання, Supabase Storage при завантаженні фотографій. Щоб зрозуміти ці взаємодії детальніше, розглянемо основні алгоритми, що реалізовуватимуться в системі.

Також я б хотів розглянути побудову каталогу оголошень, а саме алгоритм фільтрації і пагінації, який формує запит до бази даних на основі параметрів, переданих через URL. У цьому випадку користувач може задати місто або назву оголошення, діапазон цін, кількість спалень і наявність конкретних зручностей. Параметри зчитуються з рядка запиту і перетворюватимуться на умови фільтрації перед зверненням до бази даних.

Текстовий пошук виконується одночасно по полях міста, назви і країни через OR-запит, тому користувачу не потрібно обирати окрему категорію. Числові фільтри за ціною і кількістю спалень використовуватимуть порівняльні оператори, фільтр зручностей перевірятиме наявність обраного значення у масиві поля amenities. Якщо жоден параметр не задано, система поверне всі доступні оголошення.

Пагінація реалізовується методом offset/limit. Номер сторінки передаються через URL-параметр, на його основі система розраховує зміщення у вибірці. Паралельно виконують окремий запит для підрахунку загальної кількості записів, що відповідають фільтрам, щоб сформувати навігацію між сторінками. Результати сортуються за датою створення у спадному порядку, якщо користувач не задав інший критерій. Алгоритм наведено на рисунку 2.5.

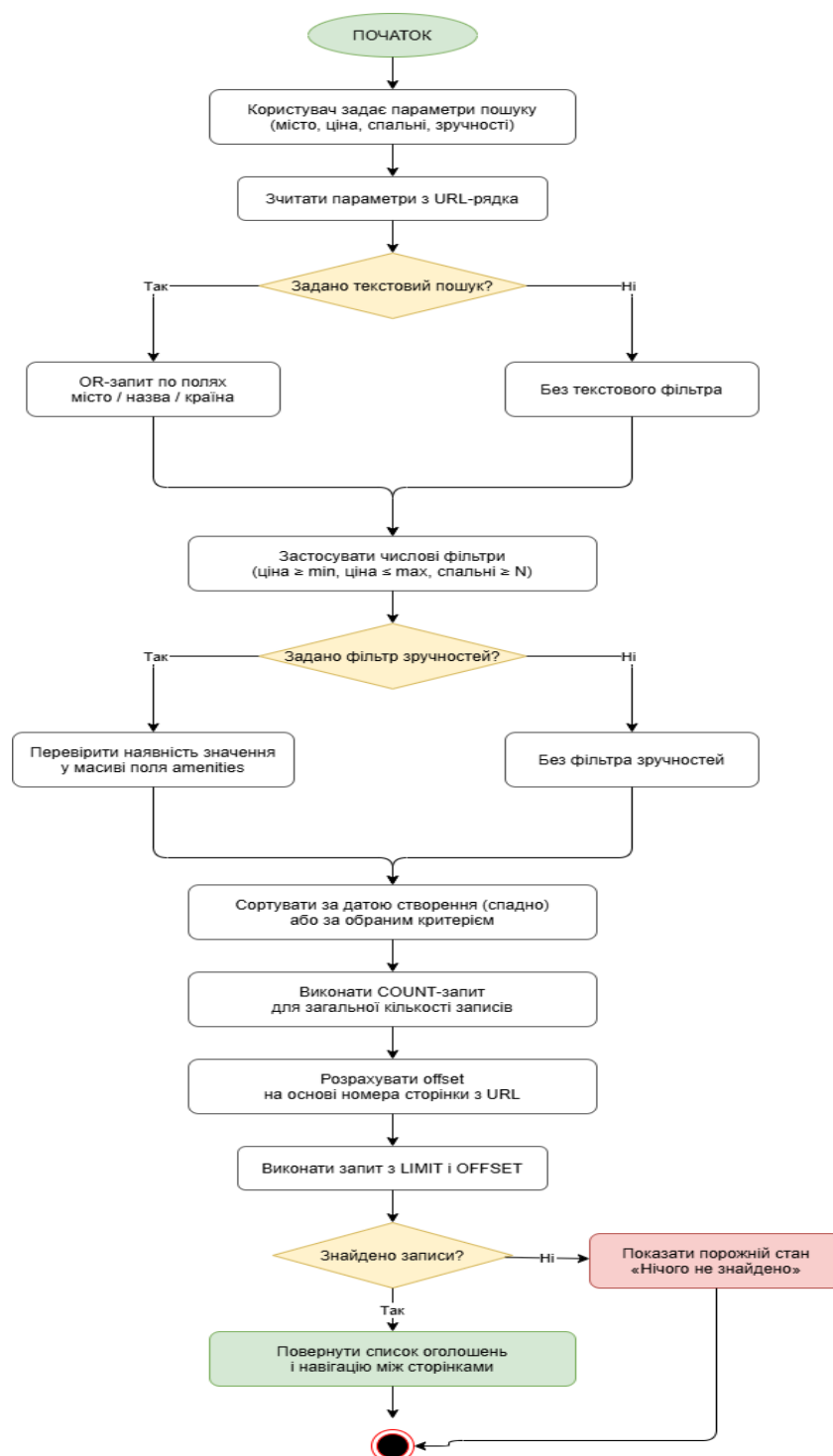


Рисунок 2.5 – Блок-схема алгоритму фільтрації і пагінації каталогу оголошень

Окремо розглянемо алгоритм автентифікації та авторизації, оскільки більшість функцій системи доступна лише після підтвердження особи користувача. Система підтримуватиме два способи входу: через обліковий запис Google за протоколом OAuth 2.0 і через email з паролем. В обох випадках після успішної перевірки облікових даних сервер формує JWT-токен з

ідентифікатором користувача і його роллю, який зберігаються на стороні клієнта і передаються з кожним наступним запитом.

При вході через Google користувача буде перенаправлено на сторінку авторизації, після чого сервер отримає код підтвердження і обміняє його на дані профілю. Якщо обліковий запис з таким email ще не існує, система створить його автоматично. При вході через email система перевірить наявність запису в базі і порівняє хеш введеного пароля зі збереженим.

При кожному запиті до захищених маршрутів proxy.ts перевірятиме наявність і валідність токена. Сторінки адмін-панелі додатково потребують ролі ADMIN у токени, користувачам вони недоступні. Відсутній або прострочений токен перенаправляє на сторінку входу. Алгоритм наведено на рисунку 2.6.

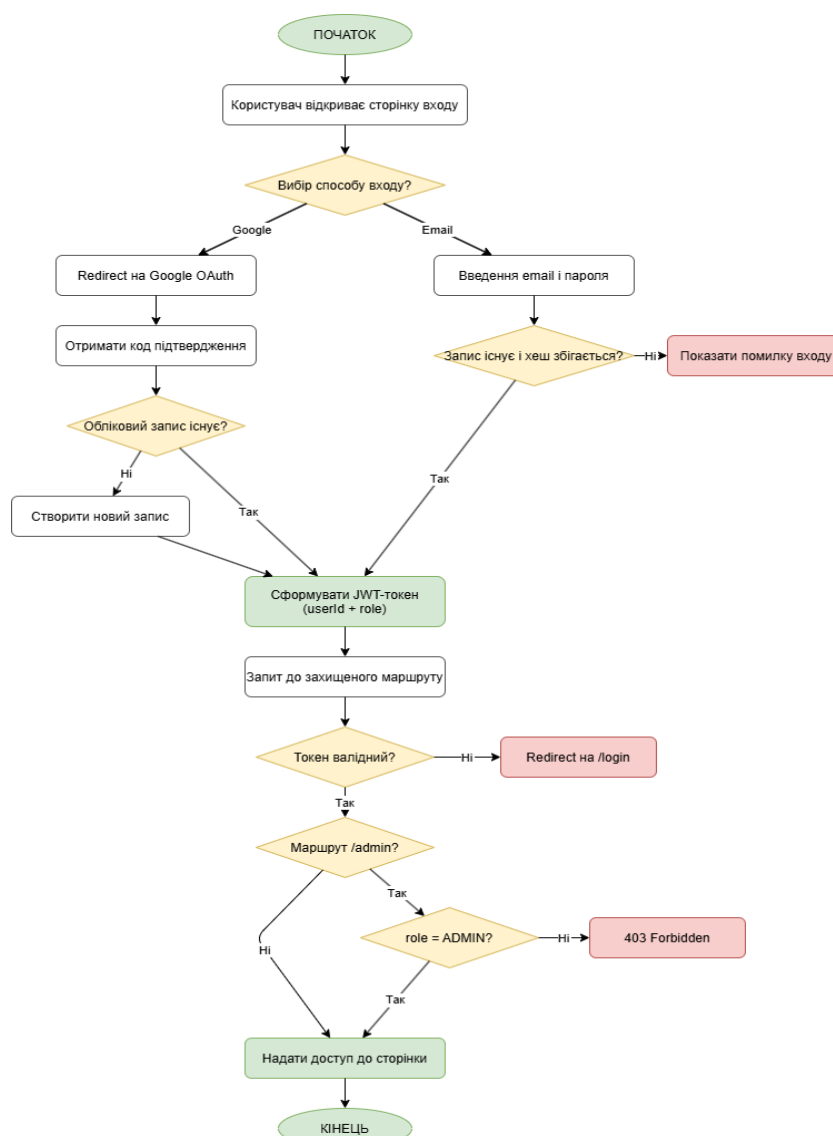


Рисунок 2.6 – Блок-схема алгоритму автентифікації та авторизації

РОЗДІЛ 3

РОЗРОБКА ВЕБ-ДОДАТКУ ДЛЯ ОРЕНДИ І БРОНЮВАННЯ ЖИТЛА

3.1 Практична реалізація об'єкта проєктування

Веб-додаток StayHub реалізовано на Next.js 16 з App Router. Структура проєкту стандартна, а саме це серверні компоненти і сторінки у `src/app`, бізнес-логіка у `src/lib`, API-маршрути у `src/app/api`, компоненти інтерфейсу у `src/components`. Середовищем розробки слугував Visual Studio Code із розширеннями ESLint, Prettier та Prisma; версійний контроль – Git. Запуск dev-сервера – `npm run dev`, перевірка збірки – `npm run build` [7]. Головну сторінку додатку наведено на рисунку 3.1.

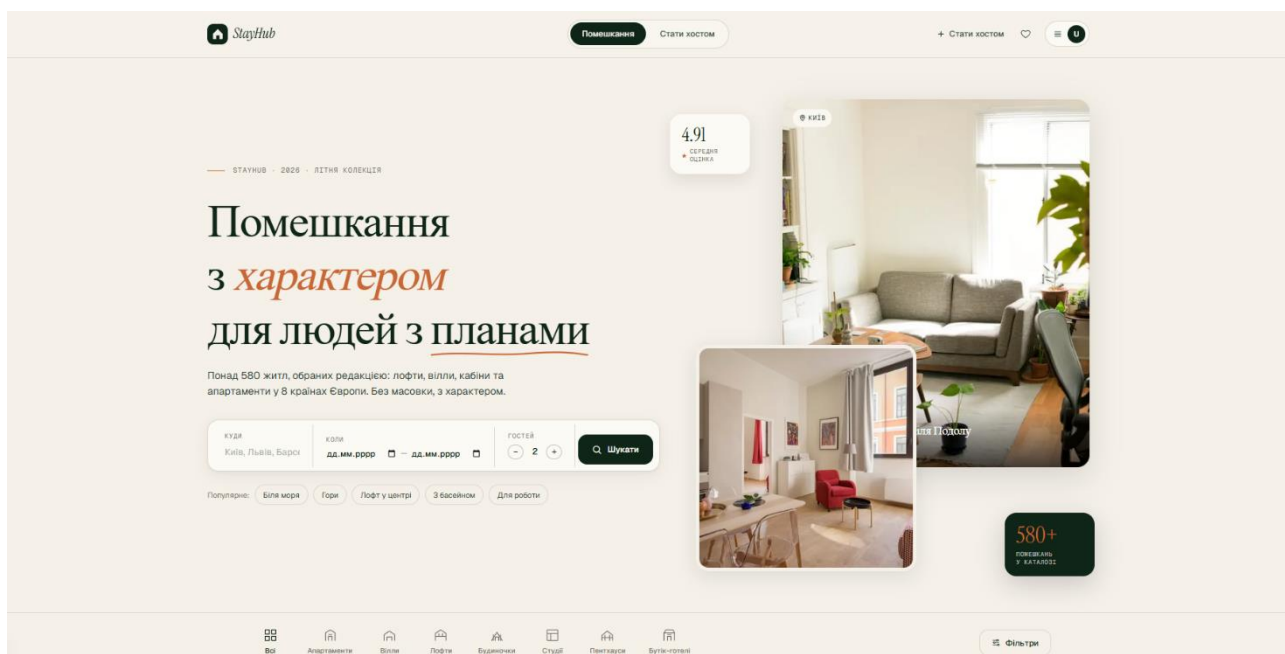


Рисунок 3.1 – Головна сторінка додатку StayHub

Маршрутизацію розподілено на дві зони. Публічна частина – головна сторінка, каталог і картки оголошень – відкрита для всіх відвідувачів. Захищені маршрути `/bookings`, `/dashboard` і `/admin` перевіряються у файлі `proxy.ts`, що запускається на Edge Runtime до початку рендерингу сторінки. Більшість сторінок написано як React Server Components, тобто запити до бази даних виконуються на сервері. Використання React Server Components дозволяє

виконувати рендеринг на сервері без передачі зайвого JavaScript на клієнт, що скорочує час першого завантаження сторінки [8]. Директива "use client" з'являється лише де справді потрібна інтерактивність, тобто фільтри каталогу, форми, booking-віджет і вкладки бронювань.

При реалізації каталогу основну складність становив алгоритм фільтрації із виключенням помешкань, зайнятих на вибрані дати. Функція `getListings()` у `src/lib/listings.ts` приймає параметри з URL і будує Prisma-запит динамічно. Якщо передано дати заїзду і виїзду, спочатку виконується запит до таблиці бронювань, система шукає записи з конкретною умовою (`checkIn < checkOutDate AND checkOut > checkInDate`), тобто будь-яке перекриття з вибраним діапазоном. Знайдені ідентифікатори передаються в `notIn` основного запиту – такі оголошення зникають з повернення. Текстовий пошук перевіряє три поля через OR: місто, назву і країну; PostgreSQL порівнює без урахування регістру через режим `insensitive` [9]. Реалізацію динамічного фільтра наведено у лістингу 3.1.

Лістинг 3.1 – Реалізація динамічного фільтра в `getListings()`

```
const where = {
  isActive: true,
  ...(query.city && {
    OR: [
      { city: { contains: query.city, mode: "insensitive" } },
      { title: { contains: query.city, mode: "insensitive" } },
      { country: { contains: query.city, mode: "insensitive" } },
    ],
  }),
  ...(bedroomsMin && { bedrooms: { gte: bedroomsMin } }),
  ...(query.superhost === "1" && { host: { isVerified: true } }),
  ...(amenityList.length > 0 && { amenities: { hasEvery: amenityList } }),
  ...(excludedListingIds.length > 0 && { id: { notIn: excludedListingIds } }),
};

const [listings, total] = await Promise.all([
  db.listing.findMany({ where, orderBy, skip, take: LIMIT }),
  db.listing.count({ where }),
]);

return { listings, total, pages: Math.ceil(total / LIMIT), page };
```

кінець лістингу 3.1

Розмір сторінки фіксований – 12 оголошень, зсув рахується як (page – 1) помножити на 12. Вибірка і підрахунок загальної кількості записів запускаються паралельно через Promise.all. Сортювання підтримує чотири варіанти, а саме це ціна за зростанням, ціна за спаданням, рейтинг і за замовчуванням, а рейтинг разом із датою публікації. Зовнішній вигляд каталогу з відкритою панеллю фільтрів наведено на рисунку 3.2.

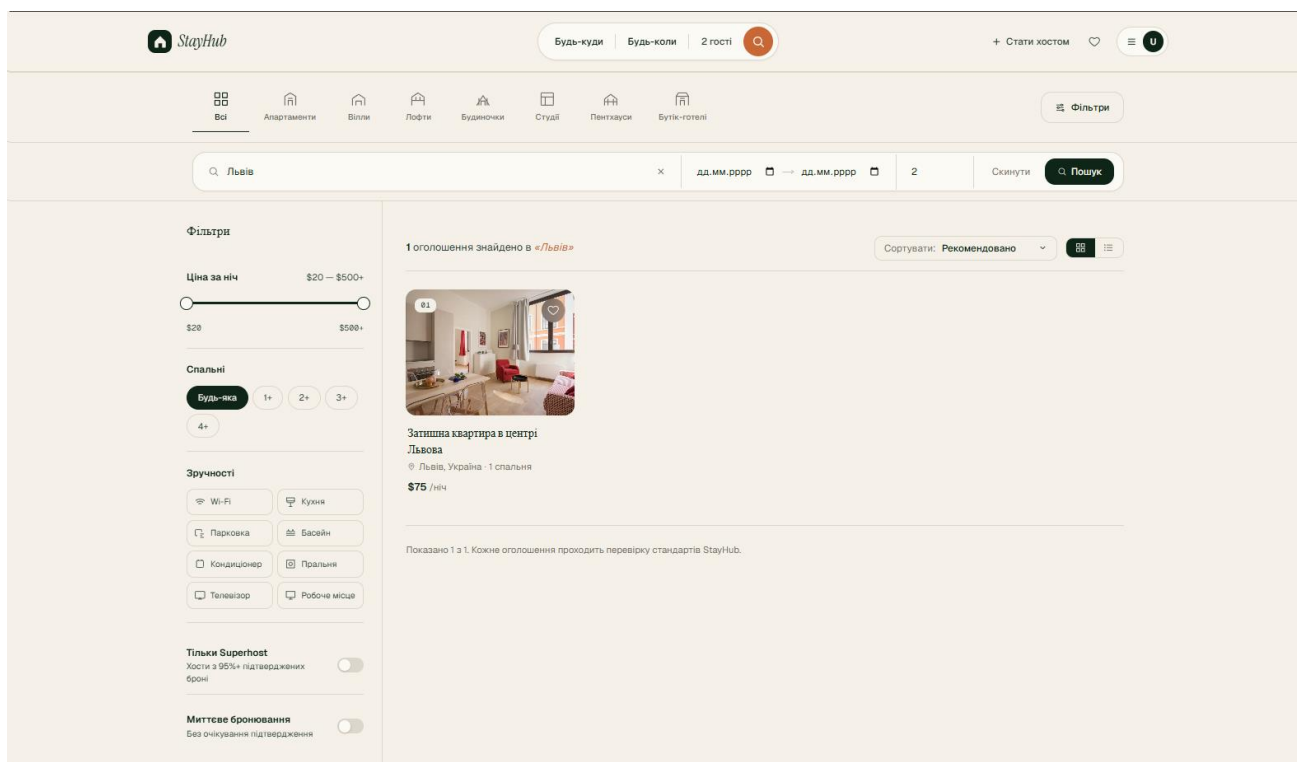


Рисунок 3.2 – Каталог оголошень із відкритою панеллю фільтрів

Маршрут POST /api/bookings перевіряє по черзі наявність активної сесії, існування і активний статус оголошення, відповідність кількості гостей обмеженню maxGuests, заборону власнику бронювати власне помешкання і відсутність перетину дат із бронюваннями у статусах PENDING або CONFIRMED. Вартість розраховується лише після того, як усі перевірки пройдено [9]. Фрагмент коду перевірки конфлікту дат і розрахунку вартості наведено у лістингу 3.2.

Лістинг 3.2 – Перевірка конфлікту дат і розрахунок вартості бронювання

```

const conflict = await db.booking.findFirst({
  where: {
    listingId,
    status: { in: ["PENDING", "CONFIRMED"] },
    checkIn: { lt: checkOutDate },
    checkOut: { gt: checkInDate },
  },
});
if (conflict) {
  return NextResponse.json(
    { error: "Ці дати вже заброньовані" }, { status: 409 }
  );
}

const nights = Math.round(
  (checkOutDate.getTime() - checkInDate.getTime()) / (1000 * 60 * 60 *
24)
);
const subtotal = nights * listing.price;
const serviceFee = Math.round(subtotal * 0.12);
  where: {
    listingId,
    status: { in: ["PENDING", "CONFIRMED"] },
    checkIn: { lt: checkOutDate },
    checkOut: { gt: checkInDate },
  },
});
if (conflict) {
  return NextResponse.json(
    { error: "Ці дати вже заброньовані" }, { status: 409 }
  );
}

const nights = Math.round(
  (checkOutDate.getTime() - checkInDate.getTime()) / (1000 * 60 * 60 *
24)
);
const subtotal = nights * listing.price;
const serviceFee = Math.round(subtotal * 0.12);
const totalPrice = subtotal + serviceFee;

```

кінець лістингу 3.2

Сервісний збір 12 % від вартості проживання, додається автоматично і разом із базовою сумою записується у поле `totalPrice`. Після цього бронювання отримує статус `PENDING` і користувач переходить до сторінки оплати. Вигляд

сторінки окремого оголошення з галереєю і booking-віджетом наведено на рисунку 3.3.

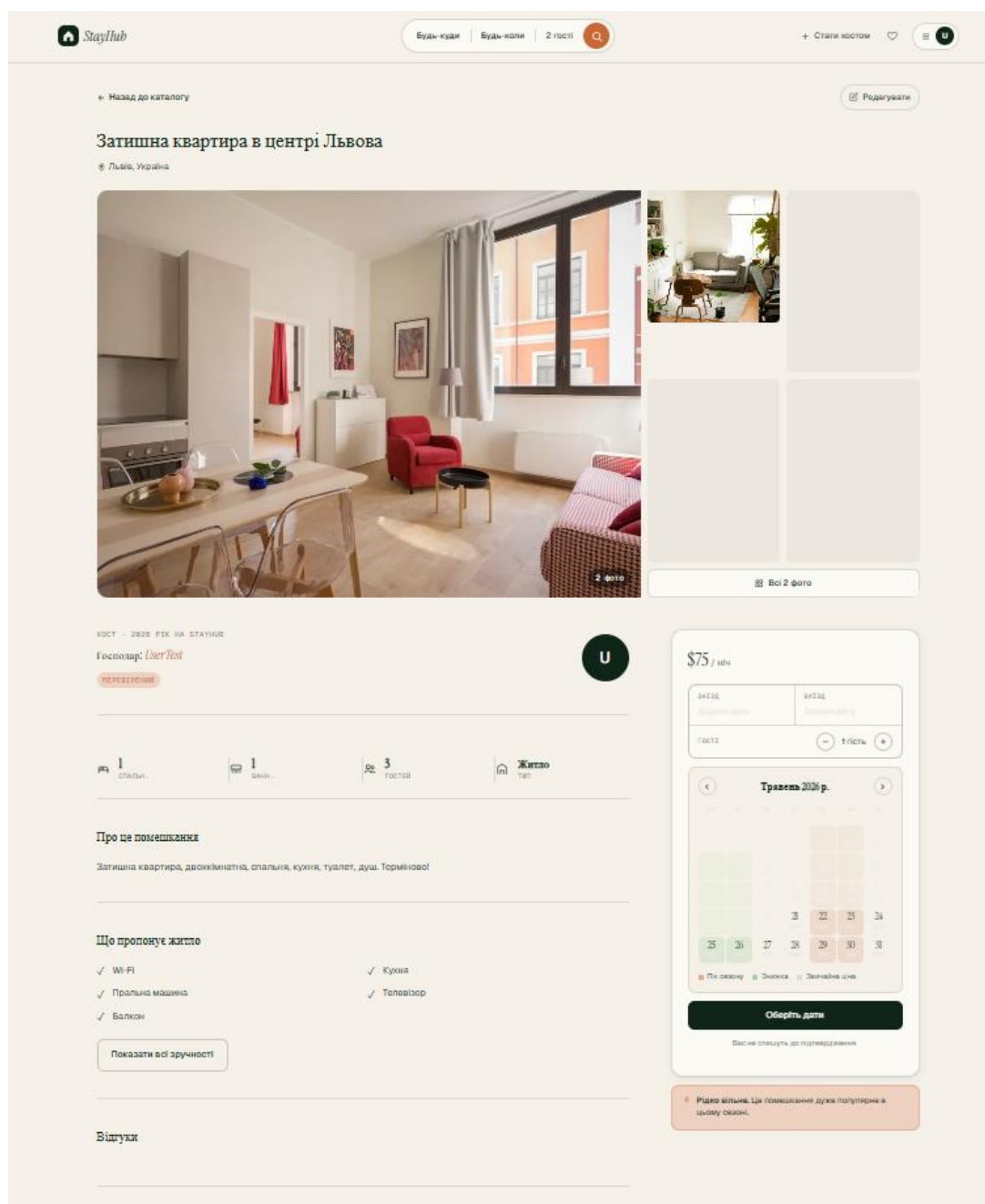


Рисунок 3.3 – Сторінка оголошення з галереєю і booking-віджетом

Оплату інтегровано через Stripe Checkout. Маршрут `POST /api/stripe/checkout` завантажує дані бронювання, формує Checkout Session із назвою оголошення і сумою у центях; у метадані передаються `bookingId` і `userId`. Одночасно у таблицю `Payment` записується рядок зі статусом `PENDING` і

stripeSessionId [10]. Реалізацію формування Checkout Session наведено у лістингу 3.3.

Лістинг 3.3 – Формування Stripe Checkout Session

```
const checkoutSession = await stripe.checkout.sessions.create({
  mode: "payment",
  success_url:
`${baseUrl}/bookings/success?session_id={CHECKOUT_SESSION_ID}`,
  cancel_url:
`${baseUrl}/bookings/${bookingId}/payment?cancelled=true`,
  line_items: [{
    price_data: {
      currency: "usd",
      product_data: { name: `StayHub: ${booking.listing.title}` },
      unit_amount: Math.round(booking.totalPrice * 100),
    },
    quantity: 1,
  }],
  metadata: { bookingId: booking.id, userId: session.user.id },
});
```

кінець лістингу 3.3

Після успішної оплати Stripe надсилає подію checkout.session.completed на /api/webhooks/stripe. Обробник перевіряє HMAC-підпис і лише після цього змінює статус бронювання на CONFIRMED, а платежу – на PAID. Якщо користувач закрав вікно без підтвердження – приходить подія checkout.session.expired і платіж переходить у FAILED. Така схема тримає стан бронювань у базі синхронізованим із Stripe навіть при мережевих затримках. Список бронювань користувача наведено на рисунку 3.4.

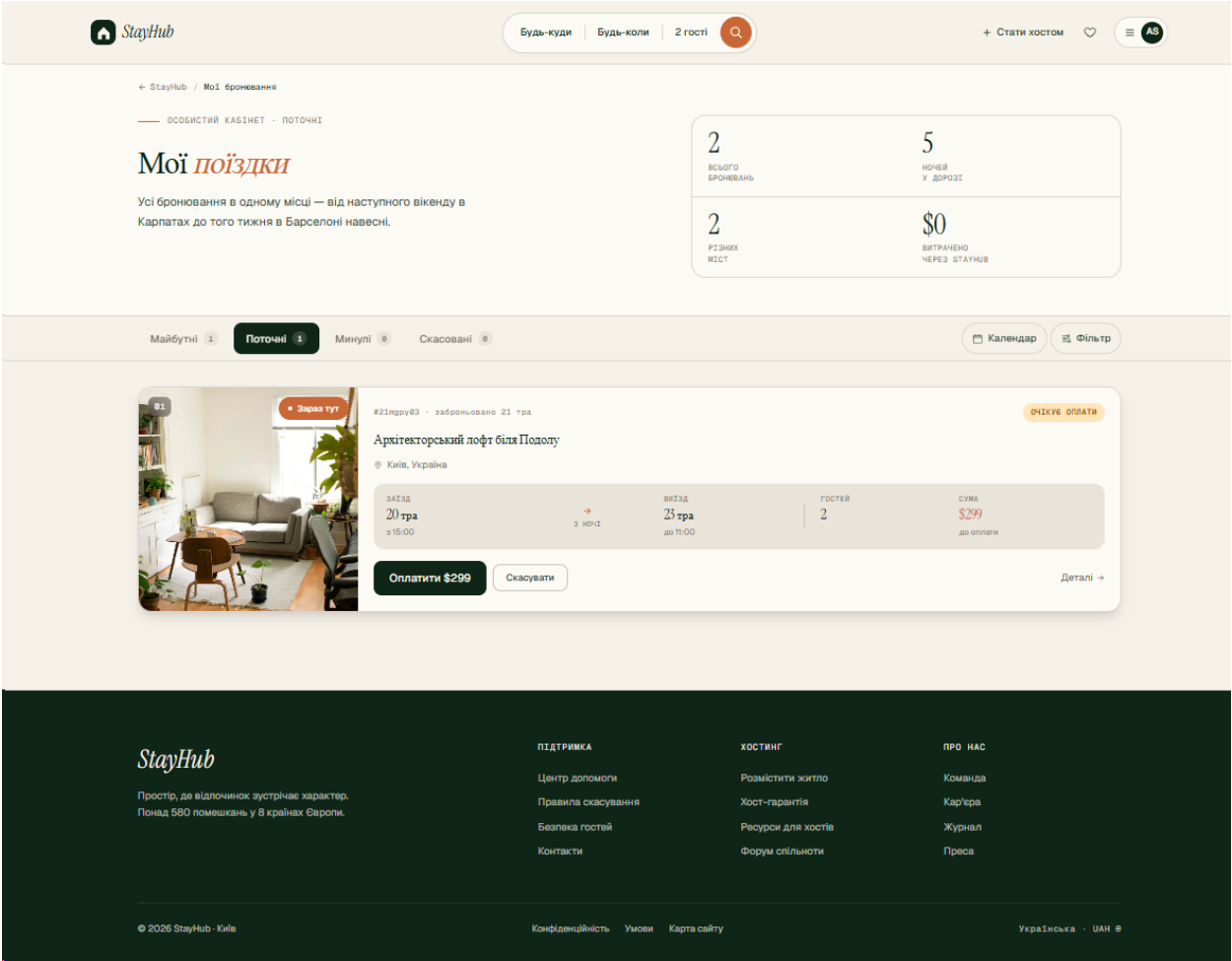
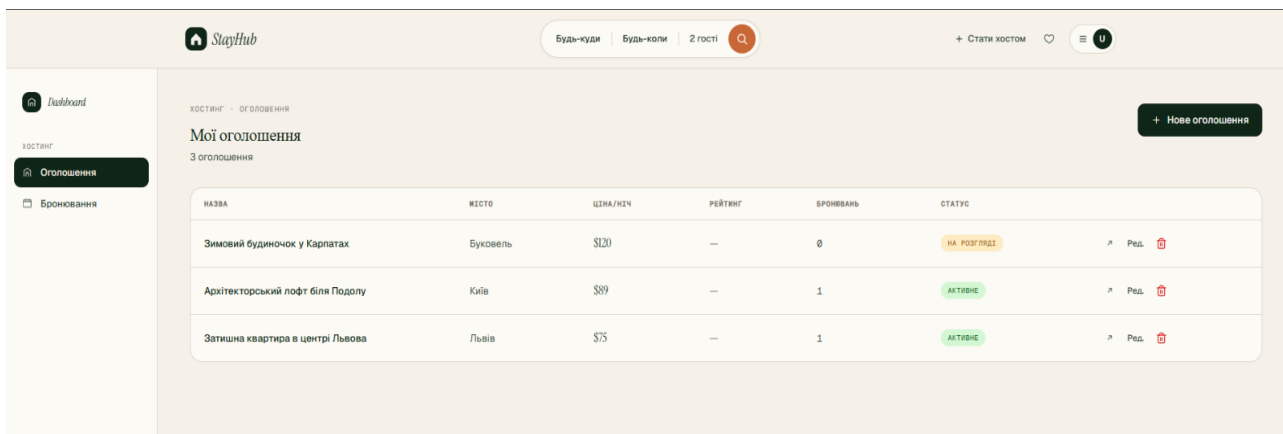


Рисунок 3.4 – Список бронювань із вкладками в особистому кабінеті користувача

Окремо реалізовано кабінет для управління оголошеннями. Публікація відбувається через багатокрокову форму `/dashboard/listings/new` – хост заповнює назву, опис, адресу, характеристики помешкання, завантажує фотографії до Supabase Storage і вказує ціну за ніч. Після збереження оголошення отримує статус `PENDING` і потрапляє в чергу модерації адміністратора. Адміністратор у панелі `/admin/listings` переглядає чергу і переводить оголошення у `ACTIVE` або `REJECTED`. Якщо відхилено – причина зберігається у полі `rejectedReason` і відображається власнику прямо в таблиці його кабінету, щоб він розумів що виправити перед повторною подачею. Окрім модерації, кабінет дозволяє редагувати наявні оголошення та вмикати або вимикати їх видимість у каталозі без видалення. Вигляд таблиці оголошень у кабінеті хоста наведено на рисунку 3.5.



The screenshot shows the 'Мії оголошення' (My Listings) section of the StayHub dashboard. It features a table with 6 columns: Назва (Name), Місто (City), Ціна/ніч (Price/night), Рейтинг (Rating), Бронювань (Bookings), and Статус (Status). There are 3 listings listed.

Назва	Місто	Ціна/ніч	Рейтинг	Бронювань	Статус
Зимовий будиночок у Карпатах	Буковель	\$120	—	0	НА РОЗГЛЯДІ
Архітекторський лофт біля Подолу	Київ	\$89	—	1	АКТИВНЕ
Затишна квартира в центрі Львова	Львів	\$75	—	1	АКТИВНЕ

Рисунок 3.5 – Таблиця оголошень зі статусами в dashboard користувача

Панель /admin показує статистику за поточний тиждень із порівнянням із попереднім: кількість користувачів, активних оголошень, бронювань і загальний дохід. Вигляд адміністративної панелі наведено на рисунку 3.6.

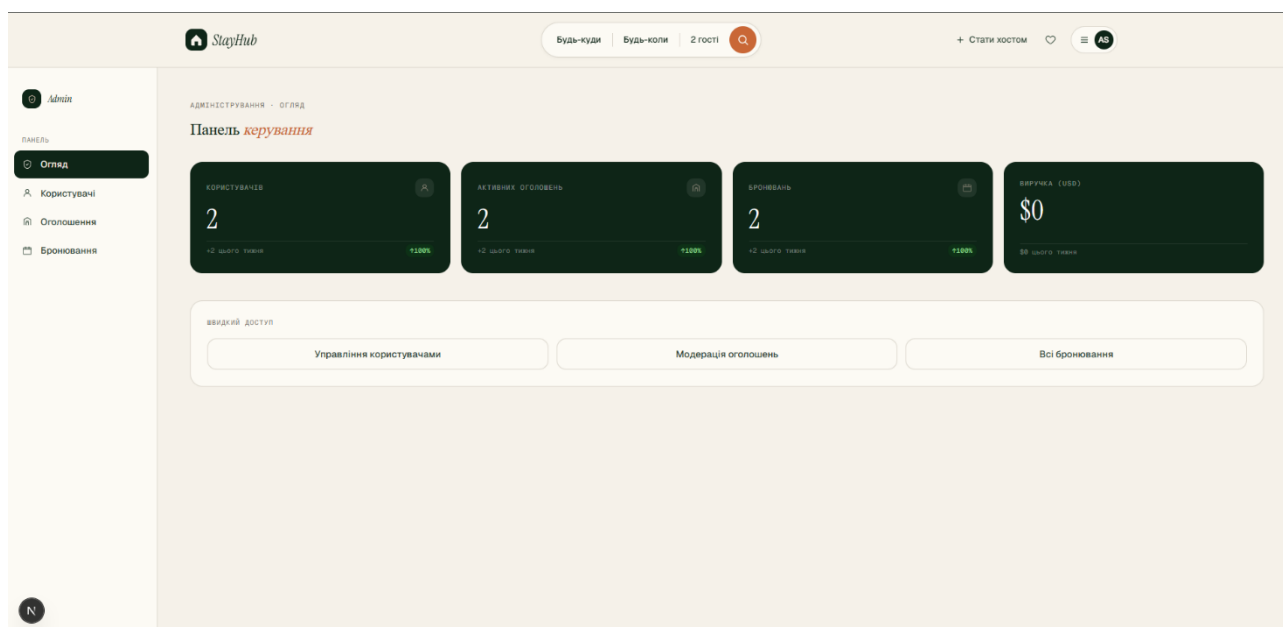


Рисунок 3.6 – Статистика платформи в адмін-панелі

3.2 Розробка бази даних

Базу даних побудовано на PostgreSQL, розгорнутому через Supabase. Для роботи з нею використовується Prisma ORM, а сама схема описана у файлі `prisma/schema.prisma`, міграції застосовуються командою `prx prisma migrate dev`.

Зміни структури фіксуються у файлах міграцій і зберігаються в репозиторії разом із кодом, що дозволяє відтворити точний стан бази на будь-якому середовищі [9].

База містить дев'ять моделей: User, Account, Session, VerificationToken, Listing, Booking, Payment, Review і SavedListing. Моделі Account, Session і VerificationToken обслуговують NextAuth і бізнес-логікою не зачіпаються. Центральна сутність – Listing, до якої прив'язані бронювання, відгуки і збережені оголошення.

Модель User зберігає профіль, це ім'я, унікальний email, захешований пароль, фото, телефон, біо і роль (USER або ADMIN). Два окремих прапори – isVerified для підтверджених хостів і isBlocked для заблокованих адміністратором. Первинний ключ – рядок у форматі CUID, що генерується на рівні застосунку. Опис моделі наведено у лістингу 3.4.

Лістинг 3.4 – Модель User у схемі Prisma

```
model User {
  id          String    @id @default(cuid())
  name        String?
  email       String    @unique
  image       String?
  phone       String?
  bio         String?
  isVerified  Boolean    @default(false)
  isBlocked   Boolean    @default(false)
  role        UserRole   @default(USER)
  password    String?
  createdAt   DateTime   @default(now())
  updatedAt   DateTime   @updatedAt

  listings    Listing[]
  bookings    Booking[]
  reviews     Review[]
  savedListings SavedListing[]
}
```

кінець лістингу 3.4

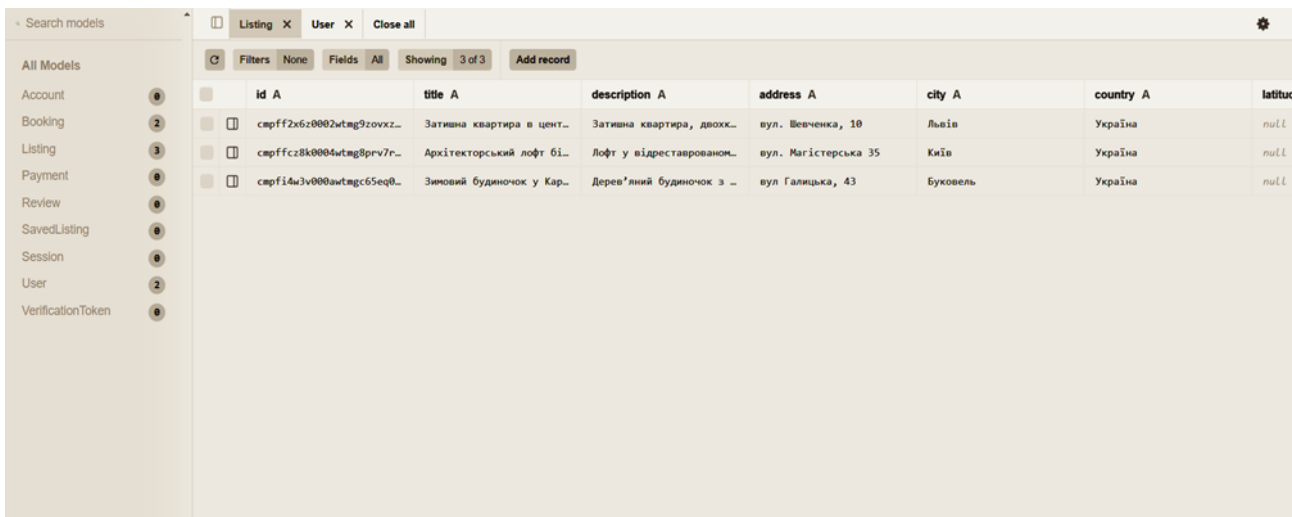
Модель Listing – найбільша у схемі. Вона зберігає назву, опис, адресу, місто, країну, координати (latitude, longitude), ціну за ніч, масив URL фотографій (images String[]), кількість спалень і ванних кімнат, максимальну кількість гостей

і масив зручностей (`amenities String[]`). Поля `avgRating` і `reviewCount` оновлюються після кожного відгуку і зберігаються денормалізовано – так каталог не потребує агрегатних запитів при кожному завантаженні. Статус оголошення контролюється через enum `ListingStatus` зі значеннями `DRAFT`, `PENDING`, `ACTIVE` і `REJECTED`; поле `rejectedReason` зберігає причину відхилення. Зовнішній ключ `hostId` зв'язує оголошення з власником через відношення «багато до одного» [11].

Модель `Booking` фіксує бронювання: дати заїзду і виїзду, кількість гостей, загальну вартість і статус (`PENDING`, `CONFIRMED`, `CANCELLED`, `COMPLETED`). Вона пов'язана з `Listing` і `User` через зовнішні ключі `listingId` і `guestId`. З моделлю `Payment` встановлено відношення один до одного через `@unique` на полі `bookingId` – одне бронювання має не більше одного платежу. Те саме обмеження діє для `Review` через `@unique` на `bookingId`: один відгук на одне бронювання, повторна оцінка того ж перебування неможлива на рівні бази.

Модель `SavedListing` зберігає список обраних помешкань користувача. Містить два зовнішніх ключі – `userId` і `listingId` – і складений унікальний індекс `@@unique([userId, listingId])`, який не дозволяє зберегти одне оголошення двічі. По суті це таблиця зв'язку «багато до багатьох» між `User` і `Listing`.

Усі зовнішні ключі налаштовані з `onDelete: Cascade`. При видаленні користувача автоматично видаляються його оголошення, бронювання і відгуки; при видаленні оголошення – всі пов'язані записи. Це прибирає потребу в ручній очистці зв'язаних даних на рівні застосунку. Окрім каскадного видалення, цілісність даних підтримується через унікальні індекси: наприклад, пара `provider` + `providerAccountId` у таблиці `Account` не дозволяє продублювати OAuth-акаунт одного провайдера для того самого користувача. Для перегляду і ручного редагування даних під час розробки використовувався `Prisma Studio` – GUI що запускається командою `prisma studio` і відкриває доступ до всіх таблиць через браузер на порті 5555. Схему бази даних наведено на рисунку 3.7.



The screenshot shows the Prisma Studio interface. On the left is a sidebar with 'All Models' and a list of models: Account, Booking, Listing, Payment, Review, SavedListing, Session, User, and VerificationToken. The 'Listing' model is selected. The main area displays a table with columns: id, title, description, address, city, country, and latitude. There are 3 records shown. The interface includes tabs for 'Listing', 'User', and 'Close all', and buttons for 'Filters', 'None', 'Fields', 'All', 'Showing 3 of 3', and 'Add record'.

id	title	description	address	city	country	latitude
cmff2x6z0002wtmg9zovxz...	Затишна квартира в цент...	Затишна квартира, двохк...	вул. Шевченка, 10	Львів	Україна	null
cmffcz8k0004wtmg8prv7r...	Архітекторський лофт бі...	Лофт у відреставрованом...	вул. Магістерська 35	Київ	Україна	null
cmff14u3v000awtmgc6Seq0...	Зимовий будиночок у Кар...	Дерев'яний будиночок з ...	вул. Галицька, 43	Буковель	Україна	null

Рисунок 3.7 – Перегляд таблиці Listing у Prisma Studio

На рисунку 3.7 видно три тестові оголошення, які використовувались під час розробки і тестування системи – помешкання у Львові, Києві та Буковелі з різними цінами за ніч. Кожен запис містить усі обов’язкові поля схеми: назву, опис, адресу, місто, країну і ціну. Поля latitude і longitude у тестових даних залишені порожніми, оскільки геолокаційний функціонал у поточній версії системи не реалізовувався і передбачений як розширення у майбутніх версіях.

3.3 Захист інформаційно-комп’ютерної системи

Захист системи розділяється на кілька незалежних рівнів, а саме автентифікацію користувачів, авторизацію за роллю, хешування паролів, перевірку API-маршрутів і верифікацію webhook-запитів від Stripe.

Автентифікацію побудовано на NextAuth v5 із двома провайдерами Google OAuth 2.0 і Credentials (email та пароль). При вході через Google користувач перенаправляється на сторінку авторизації Google, після чого повертається токен, з якого NextAuth зчитує профіль. При вході через email і пароль функція authorize завантажує запис з бази і порівнює пароль через bcrypt.compare. Якщо пароль не збігається або запис не знайдено – функція повертає null і вхід не відбувається [12].

Паролі зберігаються у базі у вигляді bcrypt-хешу. Алгоритм bcrypt із фактором складності за замовчуванням робить перебір хешів обчислювально дорогим – це стандартна практика для захисту облікових даних. У відкритому вигляді пароль ніде не зберігається і не передається після реєстрації. Реалізацію перевірки пароля при вході наведено у лістингу 3.5.

Лістинг 3.5 – Перевірка пароля при вході через Credentials

```
const user = await db.user.findUnique({
  where: { email: credentials.email as string },
});

if (!user || !user.password) return null;

const isValid = await bcrypt.compare(
  credentials.password as string,
  user.password
);

if (!isValid) return null;
```

кінець лістингу 3.5

Сесію реалізовано через JWT-стратегію (strategy: "jwt"). Після успішного входу NextAuth формує підписаний JWT-токен із id користувача і його роллю. Токен зберігається у httpOnly cookie – клієнтський JavaScript до нього доступу не має. При кожному запиті до захищеного маршруту токен перевіряється автоматично. Якщо роль користувача змінилась у базі, вона оновлюється при наступному оновленні сесії через колбек jwt з тригером update [12].

Авторизацію за маршрутами реалізовано у proху.ts, що виконується на Edge Runtime до того, як запит потрапить до будь-якого серверного компонента. Незавторизований користувач при спробі відкрити /bookings або /dashboard отримує перенаправлення на /login. Авторизований користувач без ролі ADMIN при зверненні до будь-якого маршруту /admin/* – також перенаправляється. Реалізацію захисту маршрутів наведено у лістингу 3.6.

ЛІСТИНГ 3.6 – Захист маршрутів у proxy.ts

```
export default auth((req: NextAuthRequest) => {
  const { pathname } = req.nextUrl;
  const session = req.auth;

  const isLoggedIn = !!session?.user;
  const role = session?.user?.role;

  if (pathname === "/login" && isLoggedIn) {
    return NextResponse.redirect(new URL("/", req.url));
  }

  if (pathname.startsWith("/admin")) {
    if (!isLoggedIn || role !== "ADMIN") {
      return NextResponse.redirect(new URL("/login", req.url));
    }
  }

  if (pathname.startsWith("/dashboard")) {
    if (!isLoggedIn) {
      return NextResponse.redirect(new URL("/login", req.url));
    }
  }

  if (pathname.startsWith("/bookings")) {
    if (!isLoggedIn) {
      return NextResponse.redirect(new URL("/login", req.url));
    }
  }

  return NextResponse.next();
});
```

кінець лістингу 3.6

Кожен API-маршрут додатково перевіряє сесію через виклик `auth()` на початку обробника. Навіть якщо запит якимось чином минув `proxy.ts`, без валідної сесії API поверне 401 Unauthorized. Захист не покладається на єдину точку перевірки.

Webhook-запити від Stripe верифікуються через HMAC-підпис. Stripe підписує тіло кожного запиту секретним ключем `STRIPE_WEBHOOK_SECRET`, що зберігається у змінних середовища. Обробник викликає `stripe.webhooks.constructEvent(body, sig, secret)` – якщо підпис не збігається,

повертається 400 Bad Request і база не оновлюється. Це унеможливило підробку webhook-подій від сторонніх джерел [10].

Передача даних між клієнтом і сервером відбувається через HTTPS – Supabase і Vercel забезпечують TLS-шифрування на рівні інфраструктури. Чутливі змінні середовища (ключі Stripe, секрети NextAuth, рядок підключення до бази) зберігаються у .env.local і не потрапляють до репозиторію. Сторінку входу до системи наведено на рисунку 3.8.

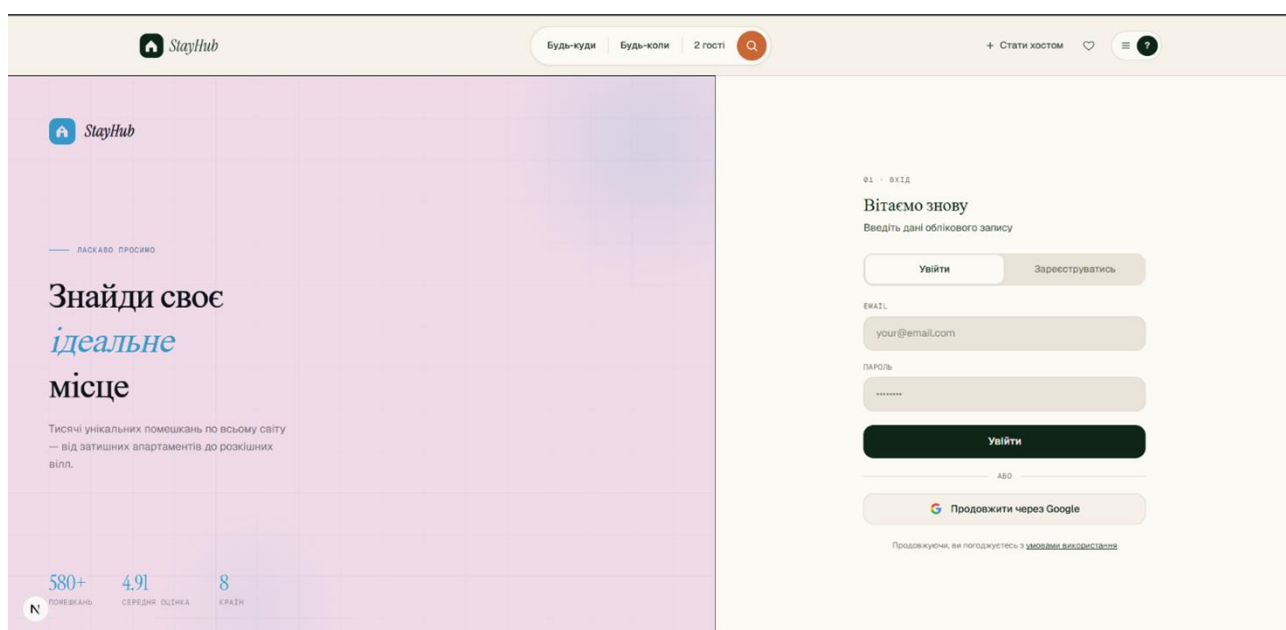


Рисунок 3.8 – Вкладки Google OAuth і email/пароль на сторінці автентифікації

На рисунку 3.8 показано сторінку входу з двома способами автентифікації через обліковий запис Google і через email з паролем. Перемикання між ними реалізовано вкладками без перезавантаження сторінки, що спрощує процес входу для різних категорій користувачів.

3.4 Тестування та налагодження інформаційно-комп'ютерної системи

У проєкті застосовувалось мануальне тестування разом із перевіркою статичної типізації TypeScript і аналізом збірки через `npm run build`. Окремого

фреймворку для автоматизованих тестів не підключалось – для навчального проєкту такого обсягу мануальна перевірка всіх сценаріїв виявилась достатньою.

Статична типізація TypeScript відловлює помилки ще до запуску. Компілятор фіксує невідповідності типів, звернення до неіснуючих полів і помилки у сигнатурах функцій на етапі написання коду. Prisma автоматично генерує TypeScript-типи для всіх моделей, тому звернення до неіснуючого поля бази або передача рядка замість числа одразу позначається як помилка компіляції [7].

Перевірку збірки виконано командою `npm run build`. Next.js під час збірки компілює весь TypeScript, перевіряє серверні компоненти на відсутність клієнтських API, аналізує маршрути і генерує статичні сторінки де можливо. Збірка завершилась без помилок і попереджень. Результат наведено на рисунку 3.9.

```
> app@0.1.0 build
> next build

▲ Next.js 16.2.6 (Turbopack)
- Environments: .env.local, .env

Creating an optimized production build ...
✓ Compiled successfully in 15.4s
✓ Finished TypeScript in 65s
✓ Collecting page data using 7 workers in 1956ms
✓ Generating static pages using 7 workers (30/30) in 2.2s
✓ Finalizing page optimization in 48ms

Route (app)
├─ /
├─ /_not-found
├─ /admin
├─ /admin/bookings
├─ /admin/listings
├─ /admin/users
├─ /api/admin/listings/[id]
├─ /api/admin/stats
├─ /api/admin/users/[id]
├─ /api/auth/[...nextauth]
├─ /api/bookings
├─ /api/bookings/[id]
├─ /api/listings
├─ /api/listings/[id]
├─ /api/profile
├─ /api/reviews
├─ /api/saved
├─ /api/stripe/checkout
├─ /api/upload
├─ /api/webhooks/stripe
├─ /bookings
├─ /bookings/[id]/payment
├─ /bookings/cancel
├─ /bookings/success
├─ /dashboard/bookings
├─ /dashboard/listings
├─ /dashboard/listings/[id]/edit
├─ /dashboard/listings/new
├─ /host-landing
├─ /listings
├─ /listings/[id]
├─ /login
├─ /profile
├─ /saved
├─ o /sitemap.xml
└─ /superhost
```

Рисунок 3.9 – Результат виконання `npm run build`

Функціональне тестування проводилось вручну для кожного реалізованого сценарію. Перевірявся повний цикл бронювання: реєстрація користувача, пошук помешкання з фільтрами, вибір дат у booking-віджеті, перехід на Stripe Checkout, успішна оплата і зміна статусу бронювання на CONFIRMED. Окремо перевірявся сценарій відхиленої оплати – статус залишається PENDING, а webhook checkout.session.expired переводить платіж у FAILED.

Тестування захисту маршрутів проводилось прямим зверненням до /dashboard, /bookings і /admin без авторизації – у кожному випадку відбувалось перенаправлення на сторінку входу. Спроба відкрити /admin під акаунтом із роллю USER давала той самий результат, що підтвердило коректну роботу proxy.ts.

Під час розробки виявлено і виправлено кілька помилок. Текстовий пошук у каталозі спочатку працював лише по полю city – після виправлення запит охопив також title і country через оператор OR. Фільтр зручностей не вимикався через передачу undefined замість порожнього рядка у параметрах URL – замінено на явне "". Слайдер ціни мав слабо видимий текст через CSS-змінну --muted із низьким контрастом – замінено на --ink-2. Усі помилки виправлено до фінальної збірки.

Мінімальні системні вимоги для роботи з додатком наведено у таблиці 3.1.

Таблиця 3.1 – Мінімальні системні вимоги

Параметр	Мінімальні вимоги	Рекомендовані
Браузер	Chrome 110+, Firefox 110+, Safari 16+	Chrome 120+
Оперативна пам'ять	2 ГБ	4 ГБ
Інтернет-з'єднання	1 Мбіт/с	10 Мбіт/с
Роздільна здатність	1024x768	1920x1080
JavaScript	Увімкнений	Увімкнений

ВИСНОВКИ

У кваліфікаційній роботі досягнуто поставлену мету – розроблено веб-додаток StayHub для оренди та бронювання житла між приватними особами з використанням Next.js, Node.js і хмарних сервісів.

Проведено аналіз сучасного стану ринку короткострокової оренди житла. Світовий ринок vacation rental у 2024 році оцінено у 95,11 млрд доларів із прогнозом зростання до 201,51 млрд до 2033 року. Здійснено огляд існуючих платформ – Airbnb, Booking.com, Vrbo, OLX і Flatfy – визначено їх переваги та обмеження. На основі проведеного аналізу обґрунтовано доцільність розробки власної системи, орієнтованої на ринок P2P-оренди.

Визначено функціональні та нефункціональні вимоги до програмного забезпечення. Сформовано перелік основних функцій системи: публікація і пошук оголошень, бронювання з перевіркою дат, онлайн-оплата, управління профілем і адміністрування платформи. Побудовано UML-діаграми прецедентів, класів і послідовності, що дозволило формалізувати структуру системи і визначити взаємозв'язки між її компонентами.

Обґрунтовано вибір технологій і архітектури системи. Обрано Next.js 16 як основний фреймворк, PostgreSQL через Supabase як базу даних, Prisma ORM для взаємодії з нею, NextAuth v5 для автентифікації і Stripe для обробки платежів. Описано алгоритми фільтрації каталогу, автентифікації та авторизації користувачів.

Розроблено структуру бази даних із дев'яти моделей із каскадними зв'язками і унікальними індексами. Реалізовано зв'язки між таблицями через первинні і зовнішні ключі, що забезпечує цілісність і узгодженість даних. Для перегляду і керування даними під час розробки використовувався Prisma Studio.

Реалізовано програмний продукт із використанням TypeScript і Next.js. Розроблено інтерфейс користувача з адаптивною версткою, реалізовано основні функції системи: пошук і фільтрацію оголошень, бронювання з розрахунком вартості, інтеграцію з платіжною системою Stripe, кабінет хоста і

адміністративну панель із модерацією. У роботі наведено фрагменти програмного коду, що підтверджують практичну реалізацію алгоритмів.

Забезпечено захист інформаційної системи. Реалізовано автентифікацію через Google OAuth 2.0 і email/пароль, авторизацію за роллю на рівні Edge Runtime, хешування паролів алгоритмом bcrypt і верифікацію webhook-запитів від Stripe через HMAC-підпис. Передача даних між клієнтом і сервером захищена протоколом HTTPS.

Проведено тестування та налагодження системи. Перевірено коректність роботи основних сценаріїв, виявлено та усунуено помилки у роботі текстового пошуку, фільтрів каталогу і відображення інтерфейсу. Продакшн-збірка завершилась без помилок і підтвердила працездатність системи.

Усі поставлені завдання виконано у повному обсязі. Розроблений веб-додаток StayHub може використовуватись як платформа для P2P-оренди житла з повним циклом від публікації оголошення до підтвердження оплати.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Vacation Rental Market Size to Hit Around USD 138.74 Bn by 2035. Precedence Research – Advisory, Research & Reports. URL: <https://www.precedenceresearch.com/vacation-rental-market> (дата звернення: 02.05.2026).
2. Attributes influencing peer-to-peer accommodation performance: Classification and the future pathways. ScienceDirect. URL: <https://www.sciencedirect.com/science/article/pii/S0278431925000167> (дата звернення: 03.05.2026).
3. Short-Term Rentals: Airbnb’s Dominance and Booking’s Gains in 1 Chart. Skift. URL: <https://skift.com/2025/03/14/short-term-rentals-airbnbs-dominance-and-bookings-gains-in-1-chart> (дата звернення: 04.05.2026).
4. Airbnb. Vacation rentals, cabins, beach houses, & more. Airbnb. URL: <https://www.airbnb.com> (дата звернення: 12.05.2026).
5. Booking.com – офіційний сайт. Booking Holdings. 2025. URL: <https://www.booking.com> (дата звернення: 13.05.2025).
6. Оголошення OLX.ua: сервіс оголошень України – купівля/продаж бу та нових товарів, різноманітні послуги на сайті OLX.ua. Оголошення OLX.ua: сервіс оголошень України – купівля/продаж бу та нових товарів, різноманітні послуги на сайті OLX.ua. URL: <https://www.olx.ua> (дата звернення: 14.05.2026).
7. Vercel. Next.js Docs. Next.js. Next.js by Vercel - The React Framework. URL: <https://nextjs.org/docs> (дата звернення: 05.05.2026).
8. Server Components – React. React. URL: <https://react.dev/reference/rsc/server-components> (дата звернення: 11.05.2026).
9. Get started with Prisma. Prisma. Serverless Postgres, Type-Safe ORM, and Database Tools. URL: <https://www.prisma.io/docs> (дата звернення: 06.05.2026).
10. Stripe Checkout Documentation. Stripe Documentation. URL: <https://docs.stripe.com/payments/checkout> (дата звернення: 07.05.2026).

11. Supabase Docs. Supabase. The Postgres Development Platform. URL: <https://supabase.com/docs> (дата звернення: 08.05.2026).

12. Auth.js. Getting Started. Auth.js. Authentication for the Web. URL: <https://authjs.dev/getting-started> (дата звернення: 09.05.2026).