ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕКОНОМІКИ, УПРАВЛІННЯ, ПРАВА ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

Пояснювальна записка

до кваліфікаційної роботи на здобуття ступеня вищої освіти Бакалавр

на тему: «Розроблення кросплатформенної версії вебсайту туристичного агентства»

Виконав: здобувач вищої освіти
за освітньо-професійною програмою
Інформаційні управляючі системи
спеціальності 126 Інформаційні
системи та технології
ступеня вищої освіти Бакалавр
групи 126ІСТбд41
Міров Д.В.
Керівник: Копішинська О.П.
Рецензент: Брикун О.М.

Полтава – 2024 року

ВСТУП

В умовах сучасного світу, де цифрові технології проникають у всі сфери життя, створення ефективних онлайн-платформ стає важливим елементом розвитку будь-якої галузі. Особливо це стосується ринку оренди житла, де попит на короткострокову та довгострокову оренду постійно зростає. Зручність та швидкість доступу до інформації про житло, можливість швидко та безпечно укладати угоди оренди стають ключовими чинниками успішності платформ.

Актуальність теми кваліфікаційної роботи обумовлена необхідністю створення зручних та ефективних онлайн-платформ для оренди житла. Сучасні користувачі шукають швидкі та безпечні способи оренди, а орендодавці потребують інструментів для ефективного управління своїми оголошеннями. Створення кросплатформного вебсайту для оренди житла з використанням сучасних JavaScript фреймворків дозволить забезпечити користувачів зручним інтерфейсом та надійними функціональними можливостями. Це вимагає комплексного підходу та поєднання сучасних технологій для створення функціонального і комерційно ефективного вебсайту.

Метою даного проекту є здійснення обґрунтованого вибору технологій розробки, створення якісного дизайну та впровадження функціонального вебсайту для оренди житла, який забезпечуватиме зручність користувачів та сприятиме популяризації сервісу.

Завданнями кваліфікаційної роботи є:

– дослідження сучасних тенденцій дизайну та засобів досягнення функціональності та властивостей вебсайтів для оренди житла;

– вибір середовища розробки для створення вебсайту;

– описати бізнес-модель, яка буде використовуватися для комерційної діяльності сайту;

– розробити макет вебсайту та продумати розміщення необхідного контенту;

– проаналізувати і використати інструменти та методи просування вебсайту;

– виконати технічну реалізацію вебсайту з використанням сучасних технологій та заходів оптимізації для покращення продуктивності і швидкості завантаження.

Об'єктом дослідження є процес розробки адаптивного вебсайту для оренди житла з використанням сучасних вебтехнологій.

Предметом дослідження є технології дизайну та розробки сучасного вебсайту на прикладі платформи для оренди житла, а також засоби оптимізації сайту для пошукових систем.

Методи дослідження: включають аналітико-синтетичний, інформаційно-пошуковий, роботу з репозиторіями програмних кодів, технічний аудит вебсайтів, графічний та інші.

Практична значущість полягає у розробці вебсайту для оренди житла з сучасним дизайном, що висвітлює переваги та функціональні можливості платформи, має форму зворотного зв'язку для реалізації всіх запитів користувачів щодо послуг компанії та залишення заявки у разі зацікавленості послугами. Дизайн сайту має адаптивні властивості. Його можна використовувати для просування будь-яких інших послуг, удосконалювати та масштабувати.

Результати роботи апробовані в рамках щорічної студентської наукової конференції кафедри інформаційних систем та технологій ПДАУ в 2024 р.

Структура та обсяг кваліфікаційної роботи. Робота складається зі вступу, трьох розділів, висновків та списку літератури. Обсяг основної частини роботи становить 61 сторінок. Робота містить 3 таблиць, 48 рисунків, 3 додатки. Список використаних джерел налічує 36 найменувань.

РОЗДІЛ 1
АНАЛІЗ ОСОБЛИВОСТЕЙ ФУНКЦІЮВАННЯ КОМЕРЦІЙНИХ ВЕБСАЙТІВ ДЛЯ БРОНЮВАННЯ ЖИТЛА

1.1 Виникнення та розвиток онлайн-платформ для оренди житла

Ідея онлайн-платформ для оренди житла зародилася на початку 1990-х рр., коли інтернет став більш доступним і популярним. Перші вебсайти в цій галузі пропонували власникам помешкань можливість здавати вільні кімнати чи квартири для короткострокової оренди.

Одним з перших таких сайтів був Craigslist, який запустили в 1991 р. Craigslist надав можливість користувачам розміщувати оголошення про здачу в оренду житла, проте цей процес був досить простим і неструктурованим, без спеціалізованих функцій для оренди житла [1] (рис. 1.1).

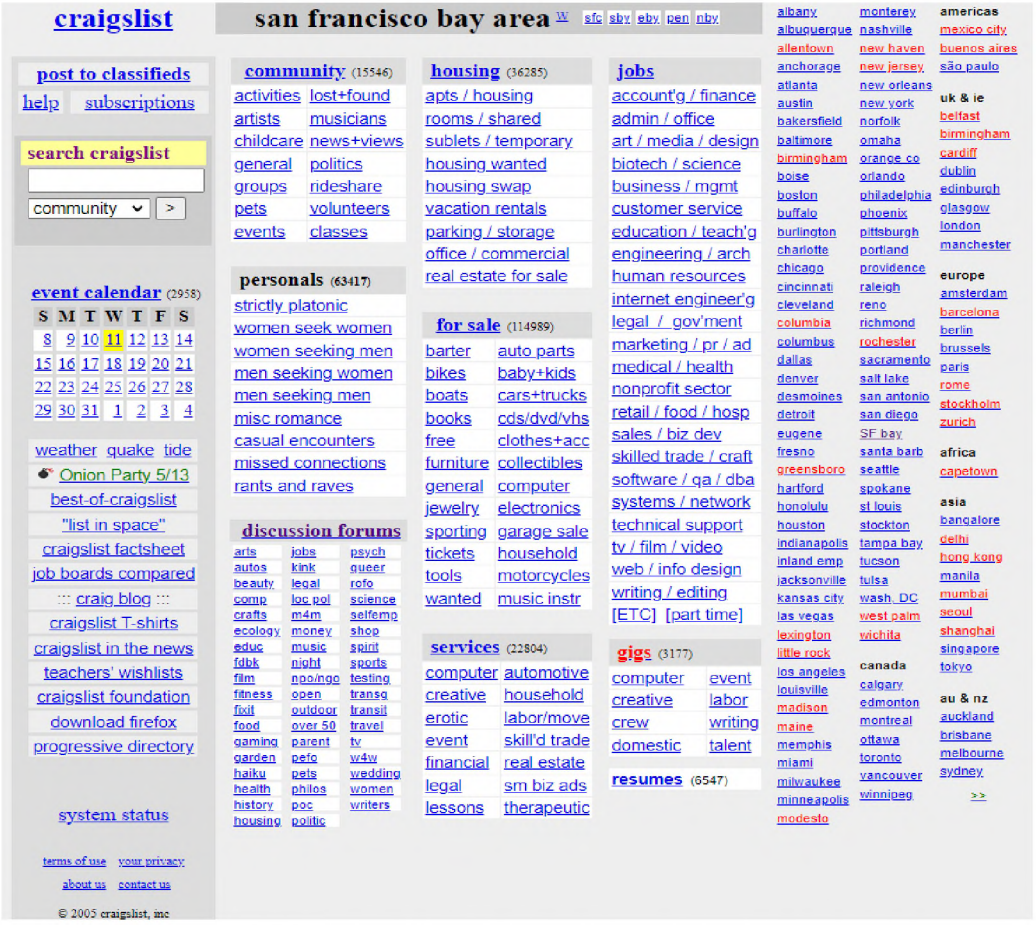

Рисунок 1.1 – Дизайн сайту Craigslist на 2005 р. [2]

Іншим важливим кроком в розвитку ринку оренди житла став сайт VRBO (Vacation Rentals By Owner), заснований в 1995 р. VRBO спеціалізувався на оренді житла для відпустки, дозволяючи власникам напряму пропонувати свої будинки та квартири для короткострокової оренди (рис. 1.2). Цей сайт надав більш структуровану платформу з можливістю перегляду фото та детального опису помешкань [3].

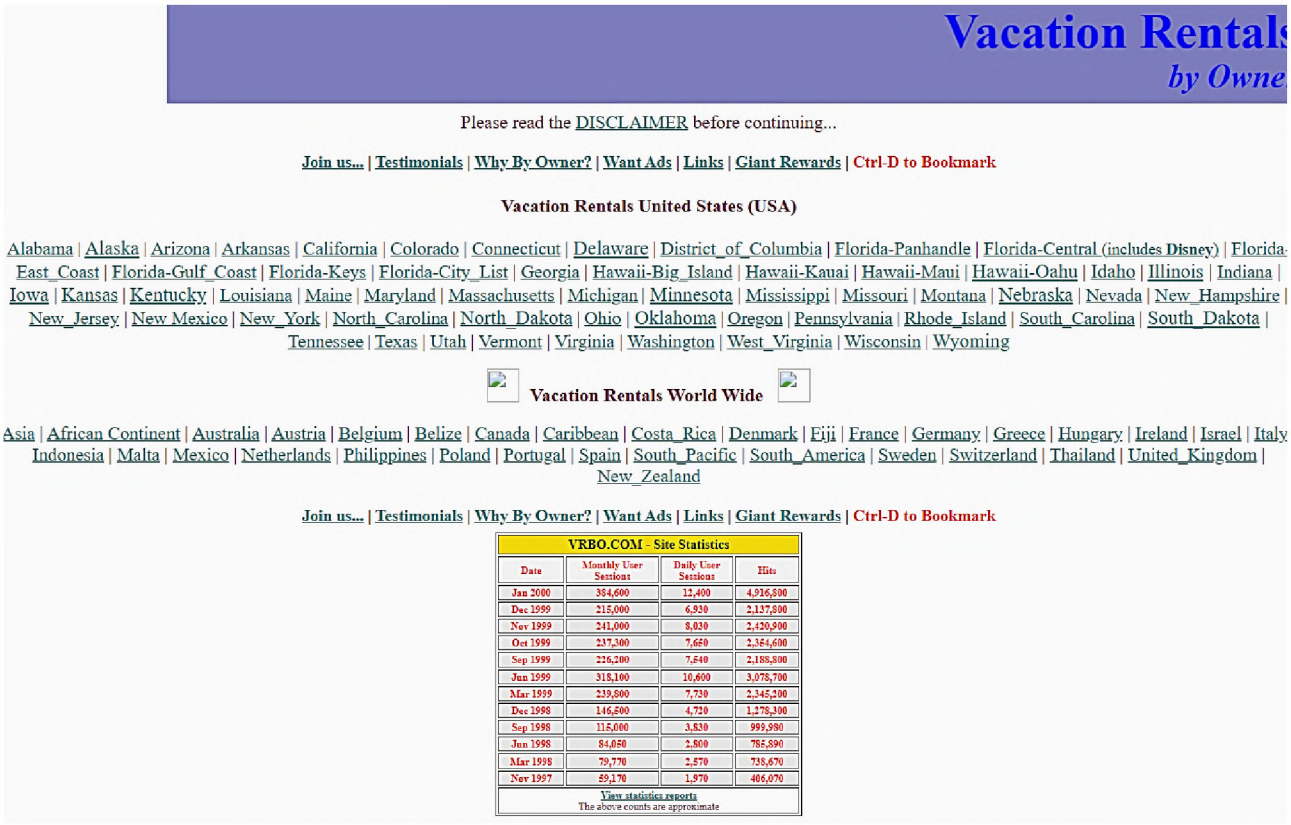

Рисунок 1.2 – Дизайн сайту VRBO на 2000 р. [4]

З часом з'явилося більше спеціалізованих платформ, таких як HomeAway, які пропонували удосконалені функції пошуку та бронювання житла. HomeAway зробила вагомий внесок у зміну способу організації подорожей, запропонувавши альтернативу традиційним готелям (рис. 1.3). Однак, у 2015 р. компанію HomeAway викупила більша компанія VRBO, що спеціалізується на орендуванні приватного житла для відпочинку. Після об'єднання, вони намагалися конкурувати з Airbnb, проте все ще відставали у питаннях безпеки угод,

ефективної взаємодії між орендодавцями та орендарями, а також вибору по-справжньому унікальних помешкань у різних куточках світу [5].

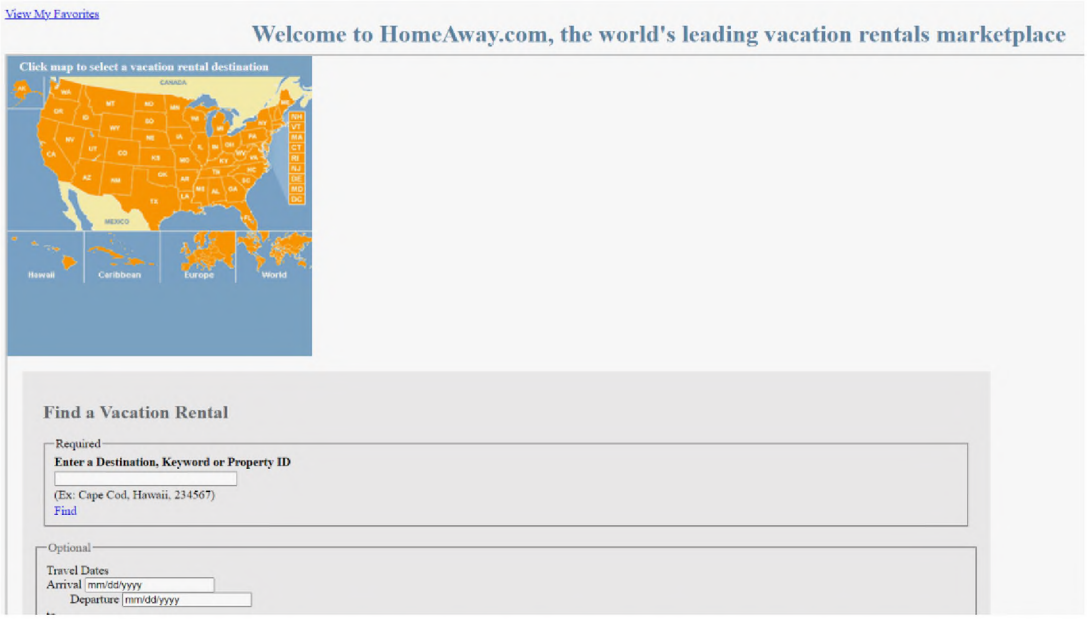

Рисунок 1.3 – Дизайн сайту HomeAway на 2000 р. [6]

Проривом у цій галузі стало створення у 2008 р. Airbnb - онлайн-платформи для бронювання унікальних помешкань у різних куточках світу. Ідея виникла в Браяна Чеські та Джо Геббіа, які почали здавати вільну кімнату в своїй квартирі в Сан-Франциско через вебсайт із метою залучення додаткового доходу.

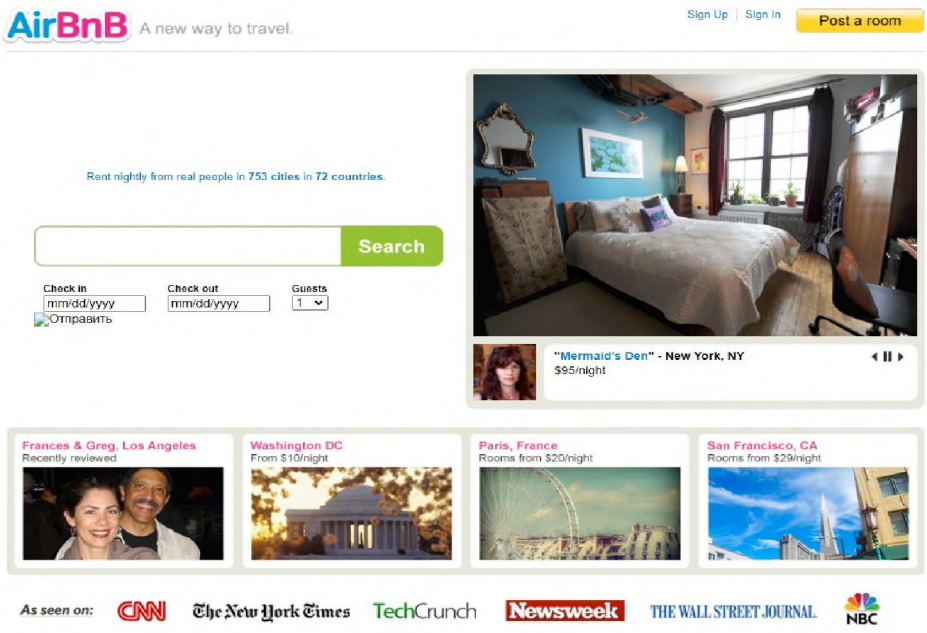

Рисунок 1.4 – Дизайн сайту Airbnb на 2006 р. [7]

На відміну від попередників, Airbnb запропонував зручний вебінтерфейс для пошуку, бронювання та оплати житла, а також механізми забезпечення безпеки та надійності процесу оренди. Це допомогло платформі швидко набути популярності серед мандрівників та туристів по всьому світу (рис. 1.4).

Сьогодні Airbnb є глобальним лідером галузі, що налічує понад 4 млн. помешкань для оренди в більш як 100 тис. міст світу [8]. Основна концепція вебсайту Airbnb полягає у наданні зручної онлайн-платформи для пошуку, бронювання та оренди унікального житла під час подорожей чи відпусток. Цільовою аудиторією є мандрівники та туристи, які шукають альтернативу традиційним готелям, а також місцеві жителі, що виїжджають на короткий термін і бажають здати своє житло в оренду.

Згідно з дослідженням компанії Statista, ринок онлайн-платформ для оренди зріс з 2015 по 2020 рр. на 30% щорічно [9]. На 2021 р., ринок оренди житла через онлайн-платформи оцінюється в 100 млрд. доларів США (рис. 1.5).

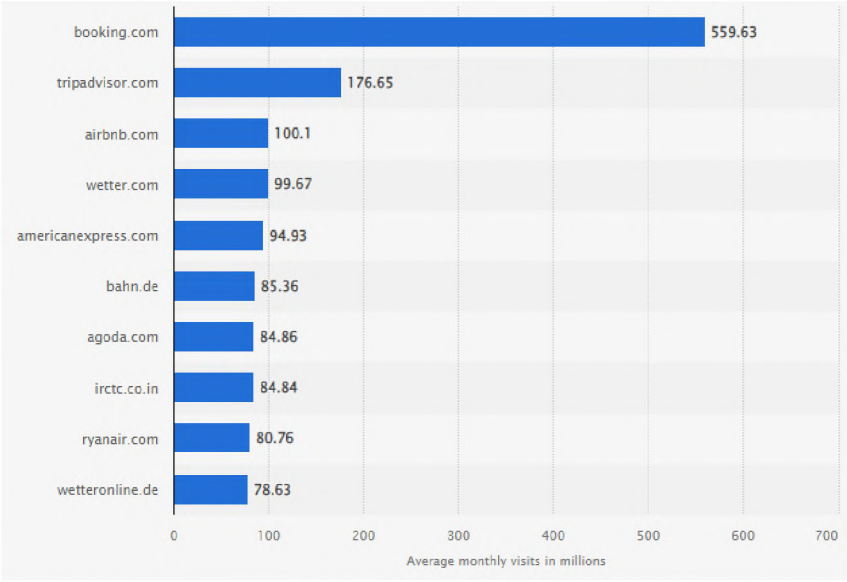

Рисунок 1.5 – Статистика відвідувань сайтів від компанії Statista

Крім того, дослідження показують, що платформи для оренди житла стають все більш популярними серед мандрівників. Згідно з дослідженням компанії Airbnb, 40% мандрівників вважають, що оренда житла через онлайн-платформи є більш економічно вигідною, ніж зупинка в готелі [10].

На відміну від систем бронювання готелів, Airbnb пропонує більш автентичний і місцевий досвід проживання у квартирах, котеджах, хостелах, будиночках на деревах тощо. Вебсайт надає зручні інструменти пошуку доступного житла за різними критеріями: місцерозташування, ціна, кількість спалень, зручності та послуги. Користувачі можуть переглядати детальну інформацію про об'єкти з фото, відгуками та оцінками інших мандрівників.

Водночас, Airbnb відкриває нові можливості для власників житла отримувати додатковий пасивний дохід. Орендодавці можуть створювати привабливі оголошення про здачу своїх помешкань в оренду ,бронювання та сплату через безпечну платформу.

Airbnb активно працює над удосконаленням своєї платформи, додаючи нові функції для зручності користувачів та безпеки угод. Наприклад, платформа запровадила систему відгуків і рейтингів, що дозволяє мандрівникам і орендодавцям залишати чесні відгуки один про одного, створюючи таким чином більш прозоре середовище.

1.2 Аналіз популярного вебсайту для оренди житла

Для аналізу вебсайту було обрано сервіс Airbnb, одну з найпопулярніших онлайн-платформ для оренди житла у світі. Airbnb пропонує широкий вибір помешкань для оренди, включаючи квартири, будинки, вілли та інші типи житла, розташовані в різних куточках світу.

Airbnb – це глобальна онлайн-платформа для оренди житла, яка надає можливість знаходження та бронювання помешкань у різних куточках світу [11]. Сайт має зручний інтерфейс та широкі можливості пошуку, включаючи фільтри за місцезнаходженням, типом помешкання, кількістю спалень, бюджетом та іншими параметрами (рис. 1.6). Користувачі можуть переглядати детальну інформацію про кожне помешкання, включаючи фото, опис, відгуки та оцінки від

попередніх гостей. Airbnb також пропонує можливість безпечної оплати через платформу та гарантує безпеку угоди між орендодавцем та орендарем.

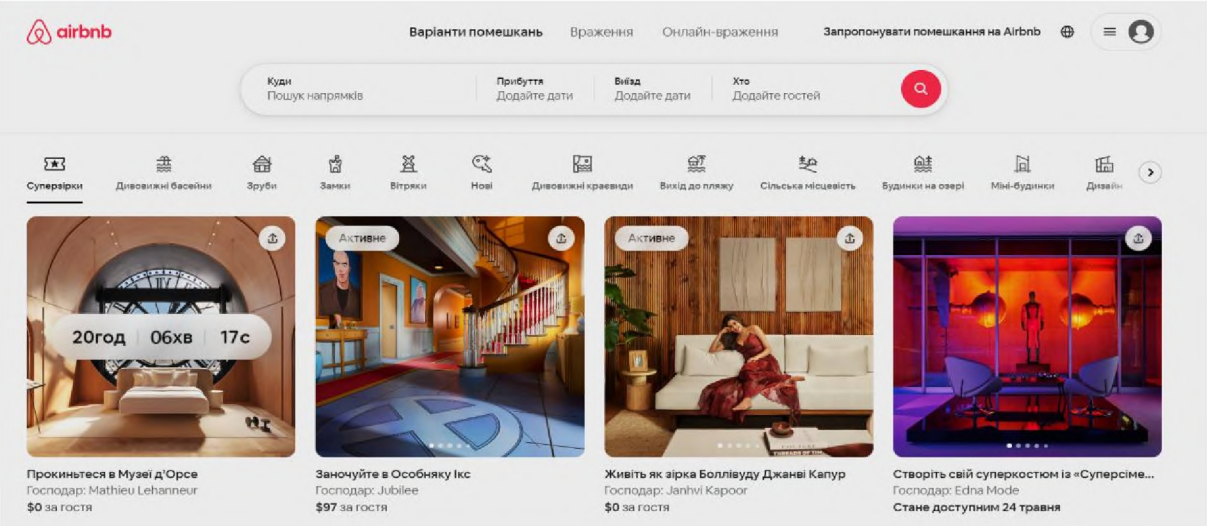

Рисунок 1.6 – Головна сторінка платформи

Airbnb розробив розширену систему фільтрації, що дозволяє користувачам вибирати помешкання згідно з їхніми уподобаннями та потребами. Фільтри включають такі параметри, як тип помешкання, розташування, доступність транспорту, наявність певних зручностей, кількість спалень, а також бюджет (рис. 1.7). Користувачі можуть також вказати дати перебування та кількість гостей, що дозволяє отримати найбільш релевантні результати пошуку.

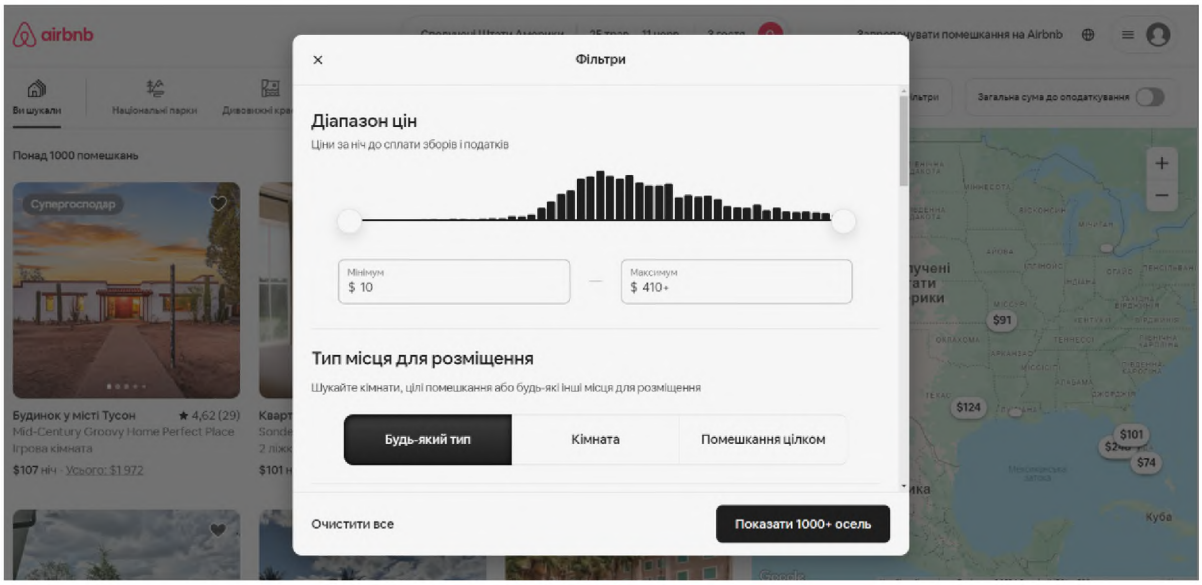

Рисунок 1.7 – Вигляд модального вікна "Фільтрація"

Функціональність сайту Airbnb забезпечує зручний та ефективний процес пошуку та бронювання помешкань. Завдяки розширеним фільтрам користувачі можуть швидко знайти ідеальне житло для своїх потреб. Платформа також пропонує можливість забронювати помешкання безпосередньо через сайт. Після вибору помешкання, користувач може зробити бронювання, вказавши дати перебування, кількість гостей та спосіб оплати (рис. 1.8). Airbnb забезпечує безпечну платформу для оплати, де користувач може вибрати зручний для нього спосіб (банківська картка, PayPal тощо). Після здійснення бронювання, Airbnb надсилає підтвердження як орендодавцю, так і орендареві.

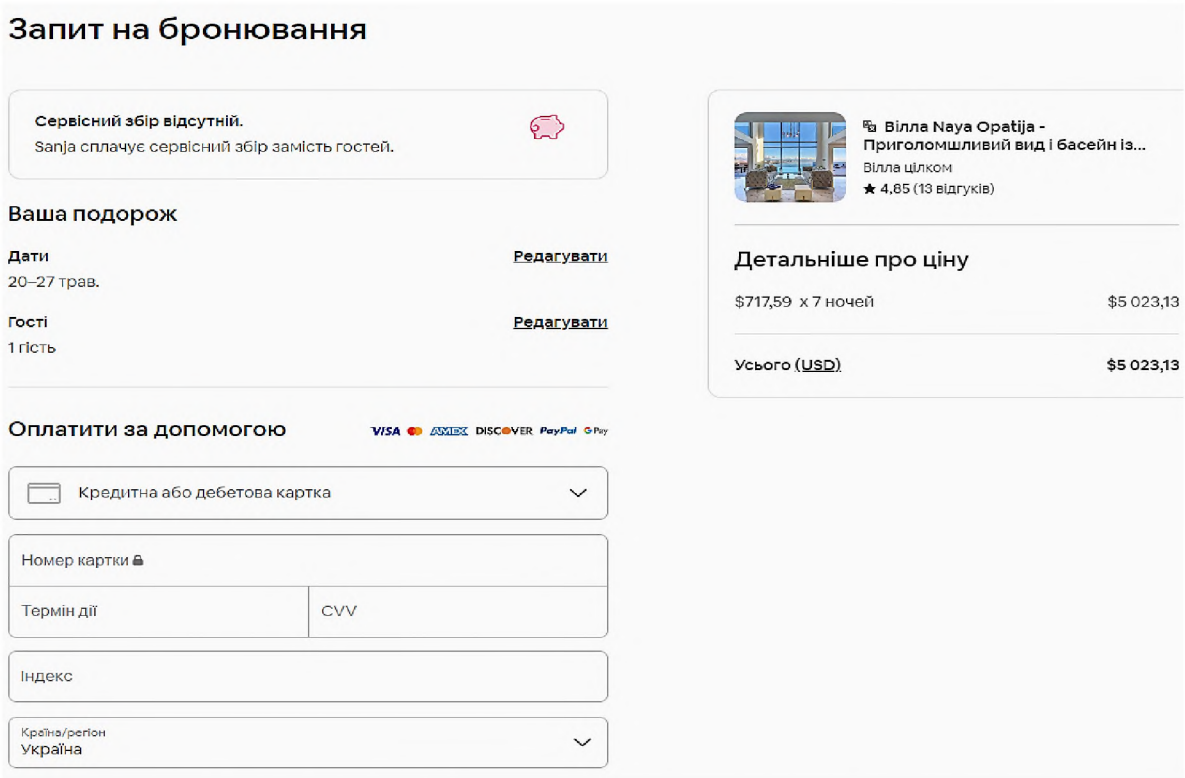

Рисунок 1.8 – Блок оплати та бронювання на Airbnb

Крім того, Airbnb пропонує безпечну систему оплати та гарантує безпеку угоди між орендодавцем та орендарем. Ця комплексна система допомагає забезпечити комфорт і спокій користувачів під час планування їхніх подорожей. Компанія постійно вдосконалює свою платформу, враховуючи зворотний зв'язок від користувачів, що дозволяє їм залишатися лідером на ринку оренди житла. Це робить процес бронювання ще більш зручним і надійним для всіх учасників.

Однією з важливих функцій Airbnb є інтеграція мапи на сторінці пошуку помешкань. Ця функція дозволяє користувачам переглядати розташування доступних варіантів на інтерактивній мапі, що значно полегшує процес вибору. Завдяки цьому користувачі можуть не тільки бачити самі помешкання, але й оцінити їх розташування відносно певних районів, транспортних вузлів, пам'яток чи інших важливих об'єктів.

Користувачі можуть відфільтрувати результати пошуку за місцезнаходженням безпосередньо на мапі (рис. 1.9). Наприклад, вони можуть обрати певний район міста або навіть конкретну вулицю, щоб знайти житло, яке знаходиться у зручному для них місці. Це особливо корисно для тих, хто хоче бути поруч з визначними місцями, конференц-центрами, парками або іншими важливими для них об'єктами.

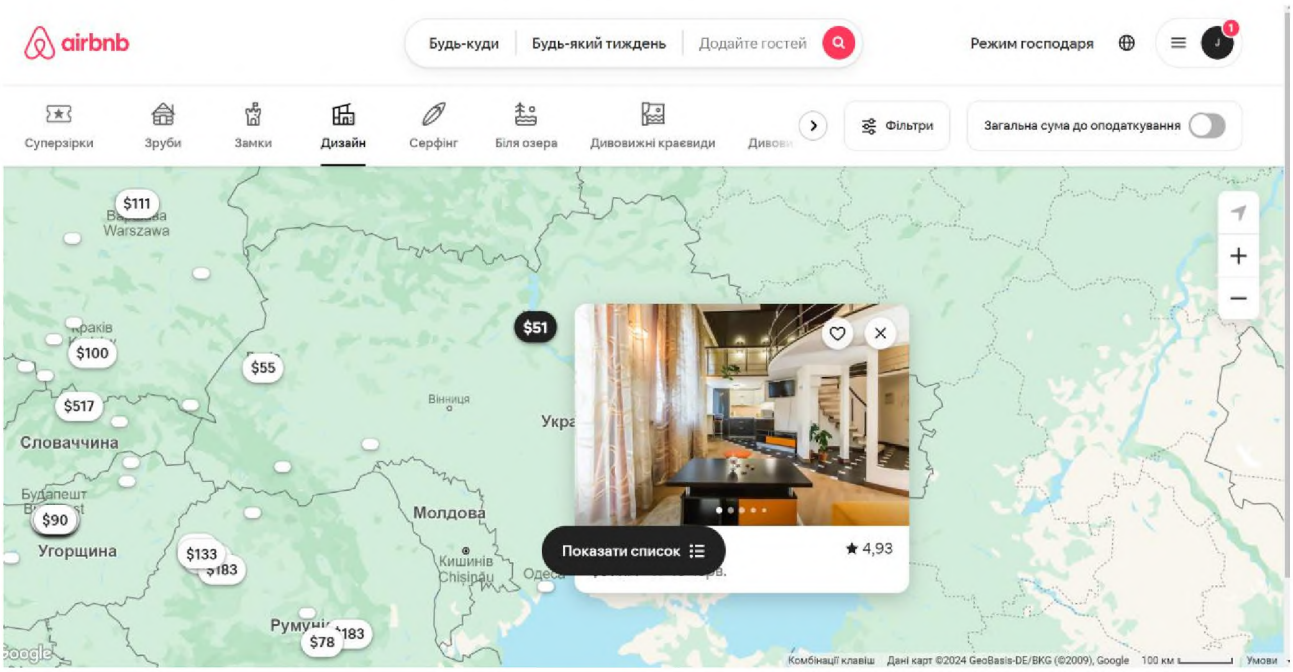

Рисунок 1.9 – Мапа об’єктів на платформі

Функція мапи також включає можливість налаштування радіусу пошуку, що дозволяє користувачам знаходити варіанти помешкань в певному радіусі від вказаного ними місця. Це зручно для тих, хто планує бути в певному районі міста та хоче мати легкий доступ до обраних ними місць, таких як офіси, навчальні заклади або туристичні атракції.

Airbnb є однією з провідних платформ для оренди житла по всьому світу, пропонуючи користувачам широкий вибір помешкань і зручний процес бронювання. Проте, як і будь-який сервіс, він має свої сильні та слабкі сторони. У табл. 1.1 основні переваги та недоліки використання Airbnb.

Таблиця 1.1 – Сильні та слабкі сторони функціональності вебсайту Airbnb

Сильні сторони	Слабкі сторони
Розширена система фільтрації	Висока конкуренція
Інтеграція мапи для зручного пошуку	Можливі проблеми з безпекою
Можливість вибору зручного способу оплати	Варіативність якості помешкань
Безпечна платформа для оплати	Залежність від відгуків користувачів
Велика кількість варіантів помешкань	Потенційні труднощі з поверненням коштів
Можливість пошуку унікальних пропозицій	Обмеження в доступності деяких регіонів
Постійне вдосконалення платформи	Високі комісії за послуги
Орієнтація на зворотний зв'язок користувачів	Проблеми з обслуговуванням клієнтів

Таким чином, Airbnb надає безліч можливостей для зручного та безпечного бронювання житла, проте користувачам варто бути уважними та ретельно перевіряти інформацію перед бронюванням, щоб уникнути можливих проблем (див. табл. 1.1). Завдяки розширеній системі фільтрації та інтерактивній мапі, користувачі можуть швидко знайти житло, яке відповідає їхнім вимогам щодо місця розташування та умов. Однак змінна якість житла та потенційні комісії можуть впливати на загальний досвід. Незважаючи на це, Airbnb залишається популярним вибором для багатьох мандрівників завдяки своїй різноманітності пропозицій і надійній підтримці клієнтів.

1.3 Оцінка найкращих технологій для інтерфейсу та серверної логіки

Під час розробки вебдодатків, вибір правильного фреймворку або бібліотеки для клієнтської та серверної частин грає вирішальну роль у досягненні успіху проєкту. В рішенні цього завдання розробники зіштовхуються з широким

спектром альтернатив, серед яких переважають кілька популярних інструментів. Перед вибором конкретного інструментарію, необхідно провести детальний аналіз їхніх можливостей, переваг та обмежень.

Під час розробки вебдодатків, розробники клієнтської частини мають на вибір безліч фреймворків та бібліотек, які допомагають полегшити процес створення користувацького інтерфейсу та роблять його більш динамічним та інтерактивним. Ось деякі з найпопулярніших та впливових інструменти для створення клієнтської частини на JavaScript [12]:

1) React.js відомий своєю простотою та ефективністю у розробці вебдодатків. Однією з головних переваг React є його використання віртуального DOM, що дозволяє реагувати на зміни у стані додатку та автоматично оновлювати відповідні елементи DOM без перемалювання всього дерева [13]. Це робить React надзвичайно швидким та продуктивним для розробки великих та складних інтерфейсів. Крім того, широке співтовариство та велика кількість доступних бібліотек та компонентів роблять React одним з найпопулярніших виборів для клієнтської частини.

2) Vue.js відрізняється своєю легкістю вивчення та використанням, що робить його ідеальним вибором для початківців та досвідчених розробників. Одним з основних переваг Vue.js є його простота та зрозумілість синтаксису [14]. Реактивність Vue.js дозволяє автоматично оновлювати інтерфейс при зміні даних, а вбудований механізм компонентів – створювати багаторівневі та повторювані елементи інтерфейсу з легкістю. Крім того, Vue.js має велику кількість офіційних та сторонніх бібліотек, що полегшує розробку різноманітних додатків.

3) Angular відомий своєю потужністю та повнотою функціональності. Однією з ключових переваг Angular є його вбудована підтримка маршрутизації, форм, HTTP-запитів та інших важливих аспектів веброзробки [15]. Angular також має вбудовану систему модулів та завантаження, що дозволяє ефективно організовувати та масштабувати великі додатки. Крім того, Angular пропонує широкий набір інструментів для тестування, що робить його ідеальним вибором для розробки великих та надійних додатків.

4) Svelte – це новаторський фреймворк, який відрізняється своєю ефективністю та продуктивністю. Однією з основних переваг Svelte є те, що він компілює код під час розробки, а не під час виконання. Це означає, що весь важкий JavaScript код перетворюється на оптимізований код, що працює значно швидше [16]. Крім того, Svelte надає простий та елегантний синтаксис, а також широкий набір вбудованих функцій, таких як реактивність та компоненти, що робить його дуже привабливим вибором для розробки вебдодатків.

При виборі фреймворка або бібліотеки для клієнтської частини на JavaScript, розробники зіштовхуються з різноманіттям інструментів та технологій. Кожен з них має свої унікальні особливості, переваги та недоліки, що можуть вплинути на рішення про вибір. Деякі особливості та переваги описану у табл. 1.2.

Таблиця 1.2 – Розгляд та співставлення провідних фреймворків для розробки призначених для користувача інтерфейсів

Фреймворк	Особливості	Переваги	Недоліки
React.js	Віртуальний DOM Компонентна архітектура	Швидкість та продуктивність Велике співтовариство	Потреба в додаткових бібліотеках для повноцінного функціонування
Vue.js	Легкість вивчення та використання	Простота та елегантний синтаксис. Велика кількість бібліотек	Менша популярність порівняно з React та Angular
Angular	Повна функціональність та широкі можливості	Вбудована підтримка маршрутизації та HTTP-запитів	Складність вивчення та використання
Svelte	Компіляція під час розробки	Ефективність та продуктивність	Менша популярність та менша екосистема

Після уважного порівняння особливостей та характеристик фреймворків та бібліотек для клієнтської частини, вирішив обрати React.js для повноцінного проєкту. Переваги швидкості та продуктивності, а також велике співтовариство та розширюваність зробили React оптимальним вибором для наших потреб. Його віртуальний DOM та компонентна архітектура дозволять ефективно створювати складні та динамічні користувацькі інтерфейси, забезпечуючи при цьому швидку та надійну роботу додатку.

При розробці серверної частини вебдодатків на Node.js, важливим аспектом є вибір типу API для взаємодії між клієнтом та сервером. Існує кілька основних типів API, які широко використовуються в сучасних вебдодатках: REST, GraphQL, WebSocket та інші.

REST є найпопулярнішим та найпоширенішим типом API. Він базується на HTTP протоколі і використовує стандартні методи, такі як GET, POST, PUT, DELETE для виконання операцій з ресурсами. REST API забезпечує високу масштабованість та простоту використання [17].

GraphQL, розроблений Facebook, дозволяє клієнту запитувати тільки ті дані, які йому потрібні. Це робить запити більш ефективними і зменшує кількість переданих даних. GraphQL також дозволяє об'єднувати кілька ресурсів в одному запиті, що робить його дуже гнучким та потужним [18].

WebSocket забезпечують двосторонній зв'язок між клієнтом та сервером в режимі реального часу. Це особливо корисно для додатків, які вимагають миттєвого оновлення даних, таких як чати, ігри та системи оповіщення [19]. WebSocket дозволяють серверу ініціювати повідомлення до клієнта без необхідності в запитах від клієнта. Порівняння переваг описано у табл. 1.3.

Таблиця 1.3 – Порівняння REST, GraphQL та WebSockets для серверної частини на Node.js

Параметр	REST	GraphQL	WebSocket	Параметр
Простота використання	Висока	Середня	Середня	Простота використання
Гнучкість	Низька	Висока	Висока	Гнучкість
Ефективність запитів	Середня	Висока	Висока (в реальному часі)	Ефективність запитів
Реалізація CRUD операцій	Дуже добра	Дуже добра	Посередня	Реалізація CRUD операцій
Масштабованість	Висока	Висока	Середня	Масштабованість
Навчання та налаштування	Легке	Складніше, ніж REST	Середнє	Навчання та налаштування

З огляду на переваги використання REST API, наступним кроком є вибір фреймворка, який найкраще підходить для реалізації цього підходу на платформі Node.js. Існує кілька потужних фреймворків, що допомагають забезпечити

ефективну роботу та швидкий розвиток проєкту. Ось деякі з найпопулярніших фреймворків для серверної розробки на Node.js:

1) Express.js є одним із найпопулярніших і найвідоміших фреймворків для Node.js. Його популярність обумовлена простотою використання, гнучкістю та великою кількістю доступних плагінів [20]. Express.js дозволяє швидко створювати надійні та масштабовані вебдодатки та RESTful API.

2) Koa.js створений командою, яка розробила Express.js, і має на меті бути менш громіздким та надавати більш виразні та надійні засоби для написання серверного коду. Koa використовує async/await, що робить код чистішим і зрозумілішим [21].

3) Hapi.js – це потужний фреймворк для створення додатків і служб, який забезпечує багатий набір функцій для роботи з запитами, валідації, кешування, аутентифікації та інших завдань [22]. Hapi.js особливо добре підходить для великих корпоративних проєктів.

4) NestJS – це прогресивний фреймворк для створення ефективних, масштабованих серверних додатків. Він використовує TypeScript і натхненний Angular, що робить його особливо привабливим для розробників, знайомих з цією фронтенд-фреймворком [23].

Після аналізу різних фреймворків для розробки серверної частини на Node.js, ми дійшли висновку, що для нашого сайту з оренди нерухомості найкраще підходить Express.js. Цей фреймворк пропонує простоту використання, високу масштабованість та гнучкість, що є ключовими для швидкого розвитку та ефективної підтримки нашого проєкту. Express.js ідеально підходить для реалізації RESTful API, що забезпечує надійну взаємодію між клієнтською та серверною частинами додатку.

РОЗДІЛ 2
ПЛАНУВАННЯ РОЗРОБКИ КРОСПЛАТФОРМНОГО ВЕБСАЙТУ ОРЕНДИ ПРИМІЩЕНЬ RENTIFYUA

2.1 Проєктування макетів та прототипів інтерфейсу

Проєктування інтерфейсу користувача (UI) є ключовим етапом у розробці кросплатформного вебсайту оренди приміщень RENTIFYUA. У цьому підрозділі розглядаються основні аспекти розробки прототипів та макетів інтерфейсу, використання принципів UX/UI дизайну, а також забезпечення доступності та зручності користування [35]. За основу беремо дизайн вебсайту Airbnb, який був детально проаналізований у першому розділі, з метою створення сучасного, зручного та привабливого інтерфейсу.

Проєктування інтерфейсу користувача включає кілька етапів, серед яких розробка прототипів та макетів інтерфейсу займає центральне місце. Створення wireframes є важливим кроком, що забезпечує візуалізацію основних елементів та структури сторінок сайту.

Wireframes є базовими ескізами, що показують розташування основних елементів на сторінках, таких як меню, кнопки, поля для введення та інші інтерфейсні компоненти. Використовуючи досвід Airbnb, плануємо розташувати ключові елементи так, щоб забезпечити легку навігацію. Wireframes також дозволяють визначити, як інформація буде організована та подана користувачам, забезпечуючи простоту та логічність у розміщенні елементів.

На першому етапі створення wireframes розробляємо початкові ескізи, що зображають список доступної нерухомості для оренди. На цих ескізах показано:

- верхнє меню для навігації по сайту;
- поле пошуку для введення запитів;
- фільтри для сортування результатів за різними критеріями;
- мапа об’єктів нерухомості;

– список об'єктів нерухомості з короткою інформацією, фото та ціною.

На даному прикладі wireframe зображено початковий прототип сторінки списку нерухомості (рис. 2.1). Цей ескіз допомагає зрозуміти, як будуть розташовані основні елементи та як користувачі будуть взаємодіяти з сайтом.

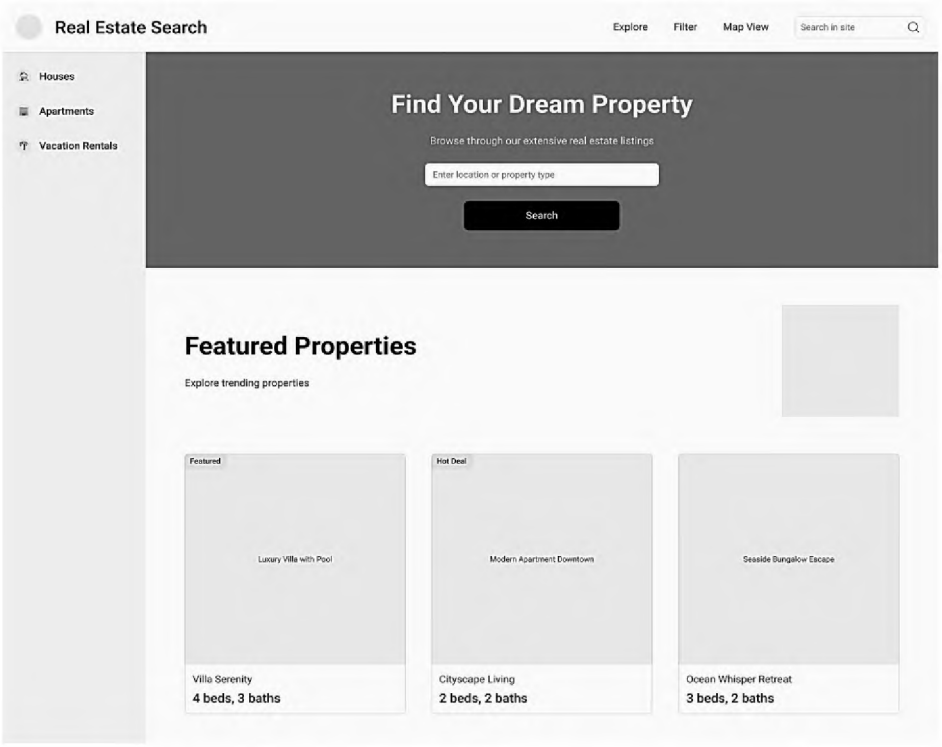

Рисунок 2.1 – Перший варіант каркасу дизайну

Після створення wireframes переходимо до розробки високодеталізованих макетів, що відображають остаточний вигляд інтерфейсу. Цей етап включає вибір кольорової схеми, шрифтів, іконок та інших графічних елементів. Дизайн макетів повинен бути привабливим та відповідати сучасним стандартам, а також враховувати потреби нашої цільової аудиторії.

Для цього будемо використовувати принципи, що вже довели свою ефективність на платформі Airbnb. Зокрема, наші макети включатимуть в себе:

– головна сторінка: яскравий та привабливий банер з полем для пошуку, що дозволяє швидко знайти житло за заданими параметрами. Також на головній сторінці будуть розміщені популярні напрямки та спеціальні пропозиції;

– сторінка пошуку: фільтри для зручного сортування результатів за ціною, розташуванням, типом житла та іншими критеріями. Список результатів буде

відображатися у вигляді карток з фотографіями, коротким описом та ціною;

– сторінка об'єкта нерухомості: детальна інформація про житло, включаючи фотографії, опис, відгуки гостей, розташування на карті та умови бронювання. Також буде можливість зв'язатися з орендодавцем для уточнення деталей;

– сторінка бронювання: інтуїтивно зрозумілі форми для введення даних, вибору дат та підтвердження бронювання. Особлива увага буде приділена безпеці та конфіденційності даних користувачів;

– кабінет користувача: особистий кабінет з можливістю перегляду історії бронювань, управління оголошеннями та налаштуваннями профілю.

Перший прототип: список нерухомості.

Як показано на рис. 2.2 представлений прототип сторінки зі списком нерухомості. Цей макет демонструє, як користувачі можуть переглядати доступні варіанти оренди житла з детальною інформацією про кожен об'єкт.

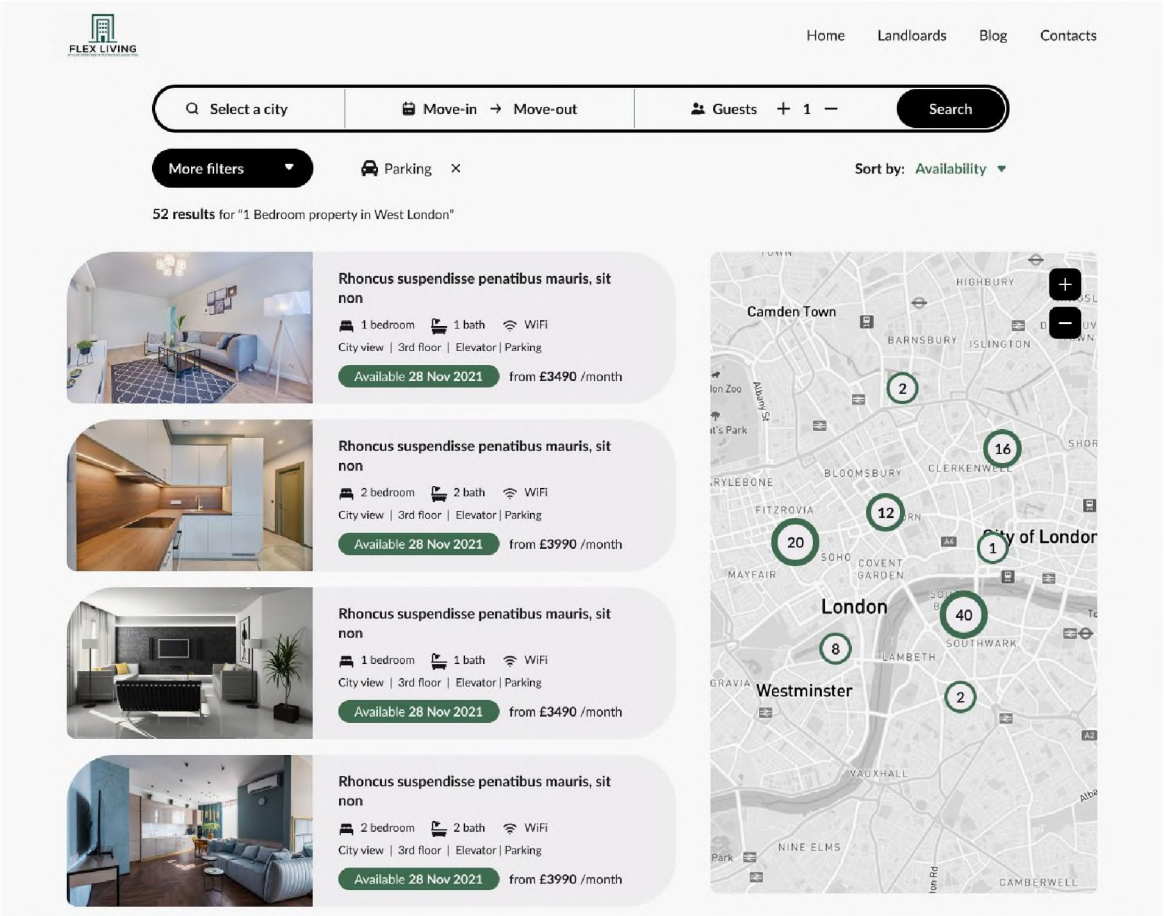

Рисунок 2.2 – Сторінка пошуку нерухомості

Основні елементи прототипу:

1) пошукова панель: верхня частина сторінки містить пошукову панель, де користувачі можуть вибрати місто, дати заїзду та виїзду, кількість гостей, а також скористатися фільтрами, такими як наявність парковки;

2) список результатів: ліва частина сторінки відведена під список результатів пошуку. Кожен об'єкт представлений у вигляді картки з фотографією, коротким описом, кількістю спалень та ванних кімнат, наявністю Wi-Fi та іншими зручностями. Також вказана ціна та доступність об'єкта;

3) карта: права частина сторінки містить карту, на якій відображені розташування об'єктів нерухомості. Це дозволяє користувачам візуально оцінити місцезнаходження кожного варіанта та вибрати найбільш зручний для них.

Цей прототип є першим кроком у створенні зручного та ефективного інтерфейсу для нашої платформи оренди житла, базуючись на перевірених принципах UX/UI дизайну та досвіді Airbnb.

Принципи UX/UI дизайну є важливими для створення зручного та привабливого інтерфейсу. Основні принципи включають простоту та інтуїтивність, консистентність, зворотний зв'язок, адаптивність та доступність.

Вони допомагають забезпечити позитивний досвід користувача, полегшуючи навігацію та взаємодію з продуктом.

Простота та інтуїтивність: інтерфейс повинен бути простим у використанні, з мінімальною кількістю непотрібних елементів. Необхідно прагнути забезпечити інтуїтивно зрозумілий дизайн, щоб користувачі могли легко знаходити необхідну інформацію та виконувати потрібні дії. Простота досягається за рахунок використання чіткої типографіки, великих кнопок дій та простих форм для заповнення.

Консистентність: всі елементи інтерфейсу повинні бути узгоджені за стилем, кольоровою гамою та розташуванням. Сайт демонструє чудовий приклад консистентності в дизайні, який будемо використовувати як орієнтир. Консистентність створює відчуття єдності та стабільності, що сприяє підвищенню довіри користувачів до платформи (рис. 2.3).

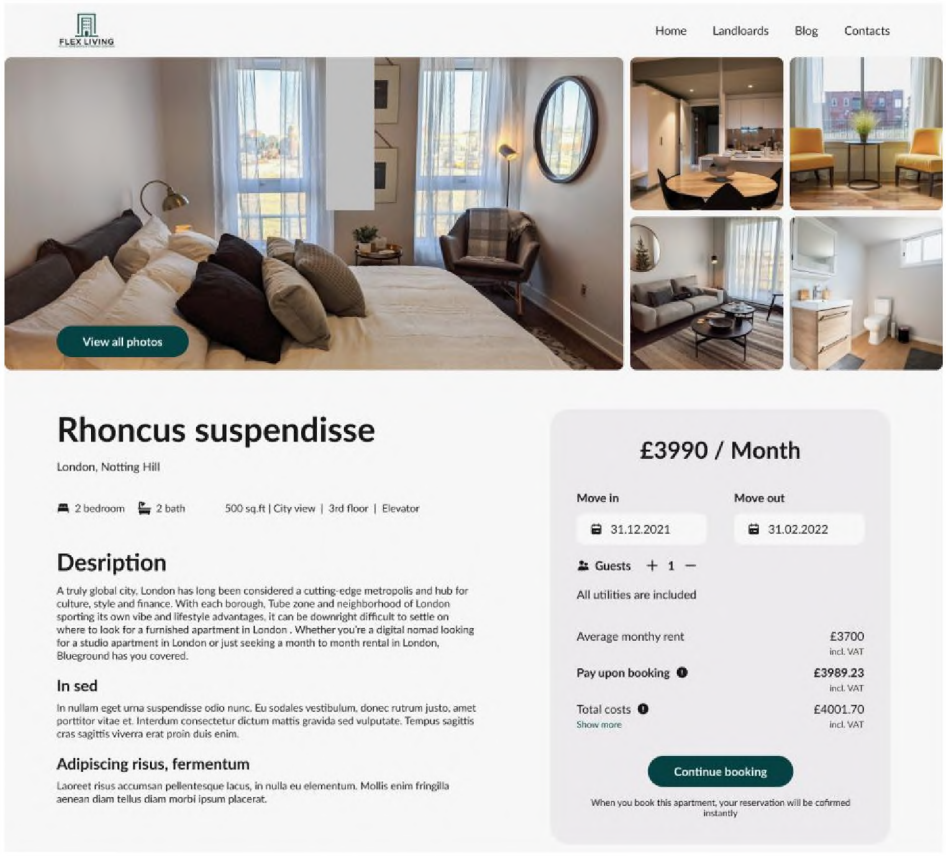

Рисунок 2.3 – Фінальна версія дизайну

Зворотний зв'язок: система повинна надавати користувачам зворотний зв'язок про їхні дії, наприклад, підтвердження успішного бронювання або повідомлення про помилку (рис. 2.4). Вебсайт буде забезпечувати миттєвий зворотний зв'язок для підвищення задоволеності користувачів. Зворотний зв'язок може бути реалізований у вигляді спливаючих повідомлень, змін кольору елементів або анімацій.

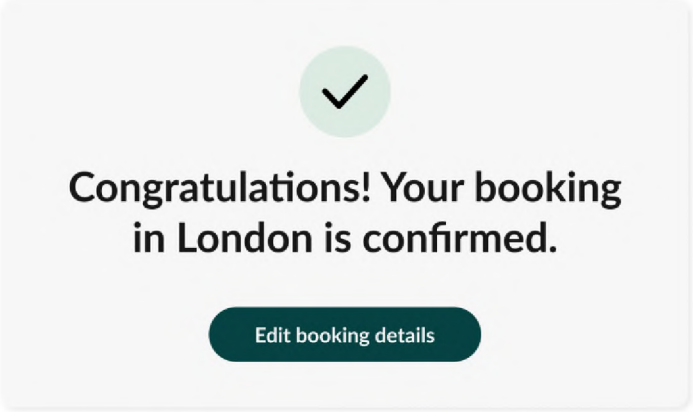

Рисунок 2.4 – Модальне вікно підтвердження

Адаптивність: інтерфейс повинен коректно відображатися на різних пристроях, включаючи комп'ютери, планшети та смартфони. Адаптивний дизайн забезпечує доступ до всіх функцій сайту незалежно від типу пристрою, що використовується.

Доступність: врахування потреб користувачів з обмеженими можливостями. Це включає підтримку екранних читачів, використання контрастних кольорів та забезпечення можливості навігації за допомогою клавіатури. Доступність є важливим аспектом, що забезпечує рівний доступ до платформи для всіх користувачів.

Для забезпечення доступності необхідно дотримуватися стандартів Web Content Accessibility Guidelines (WCAG) [24]. Це включає використання альт-текстів для зображень, що допомагає людям з вадами зору, забезпечення достатнього контрасту між текстом та фоном, можливість навігації за допомогою клавіатури для людей з вадами моторики, а також використання семантичних HTML-елементів для покращення доступності.

Крім того, WCAG рекомендує враховувати потреби користувачів з різними типами вад слуху та когнітивними обмеженнями, забезпечуючи альтернативні форми сприйняття інформації, такі як підписи до відео або просте та зрозуміле описання контенту. Важливо пам'ятати, що дотримання стандартів WCAG дозволяє зробити вебсайти та вебдодатки доступними для максимально можливого кола користувачів, забезпечуючи рівність у доступі до інформації та сервісів в Інтернеті.

Для приклада сайт Airbnb в якому використовується ряд прийомів згідно зі стандартами WCAG для забезпечення доступності для всіх користувачів. Наприклад, у розділі пошуку помешкань, кожне зображення має відповідний альтернативний текст, який допомагає користувачам з вадами зору розуміти зміст зображення. Крім того, колірна палітра та контрастність вебсайту досить ретельно підібрані, щоб забезпечити чітке виділення тексту та інших елементів для полегшення сприйняття інформації (рис. 2.5). Сайт також надає можливість навігації за допомогою клавіатури, що забезпечує зручний доступ для

користувачів з вадами моторики. Такі заходи допомагають зробити вебплатформу більш доступною для всіх користувачів, незалежно від їхніх індивідуальних потреб та обмежень.

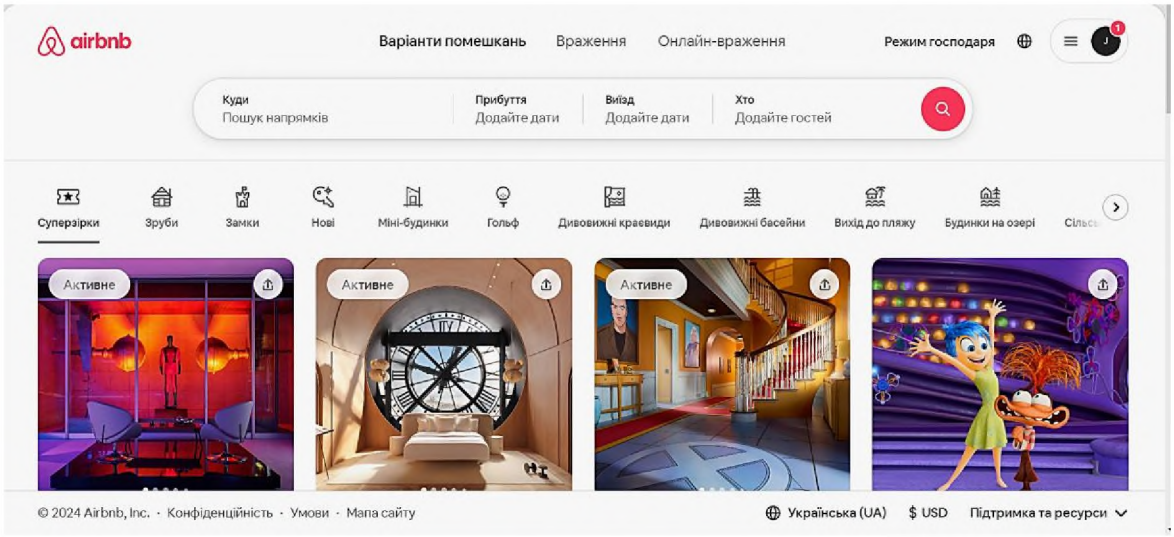

Рисунок 2.5 – Використання WCAG на платформі

Тестування прототипів з реальними користувачами допоможе визначити проблеми з доступністю та зручністю, а також знайти шляхи їх вирішення. Адаптивний дизайн забезпечить коректне відображення сайту на різних пристроях, а постійне вдосконалення інтерфейсу на основі зворотного зв'язку від користувачів та результатів тестування допоможе створити сучасний та зручний вебсайт оренди приміщень.

2.2 Реалізація клієнтської частини вебсайту

Розробка клієнтської частини вебсайту є критично важливим етапом, оскільки саме з ним взаємодіє кінцевий користувач. Для реалізації даного етапу обрали React, зважаючи на його популярність, ефективність та гнучкість. Як було зазначено в першому розділі, React використовується для створення динамічних та інтерактивних користувацьких інтерфейсів, що дозволяє забезпечити високий рівень зручності користування.

Для ініціалізації проєкту вирішив використати Vite. Vite – це сучасний інструмент для створення та збірки вебдодатків, який забезпечує швидке та ефективне розгортання проєктів [25]. Він розроблений таким чином, щоб мінімізувати час збірки та спростити процес розробки, що особливо важливо при створенні великих і складних додатків.

Його основні переваги включають миттєвий старт проєкту, гаряче перезавантаження модулів (HMR) для миттєвих оновлень під час розробки, та оптимізовану збірку для фінального розгортання. Vite підтримує сучасні стандарти JavaScript та може працювати з різними фреймворками, такими як React, Vue, Svelte та інші.

Для того, щоб розпочати розробку проєкту на основі Vite та React, необхідно виконати наступні кроки:

1) встановлення Node.js та npm: Перш за все, необхідно переконатися, що на комп'ютері встановлені Node.js та npm (Node Package Manager) [26]. Це інструменти, які забезпечують середовище для запуску JavaScript на сервері та керування залежностями проєкту відповідно;

2) створення нового проєкту за допомогою Vite: для створення нового проєкту використовуємо команду npm create vite@latest, яка запустить процес ініціалізації. Після цього система запитає ім'я проєкту та вибір шаблону. Обираємо шаблон для React (рис. 2.6);

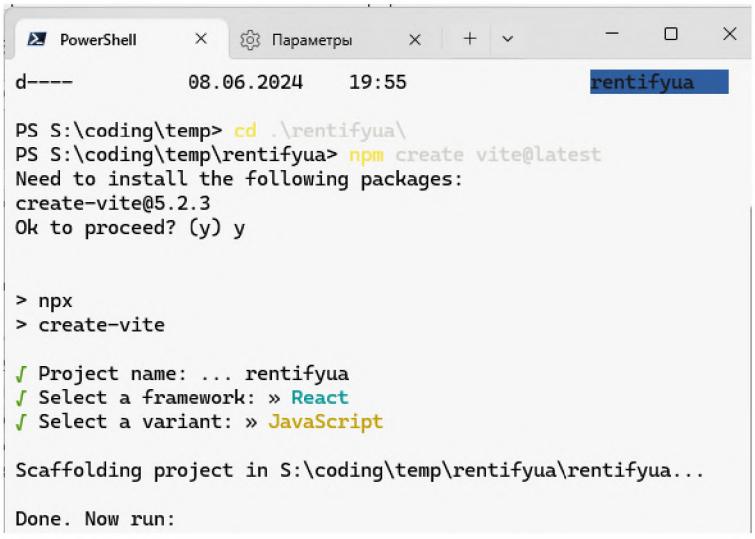

Рисунок 2.6 – Ініціалізація нового проєкту

3) встановлення залежності: Після створення проєкту переходимо в його директорію та встановлюємо всі необхідні залежності за допомогою команди npm install (рис. 2.7);

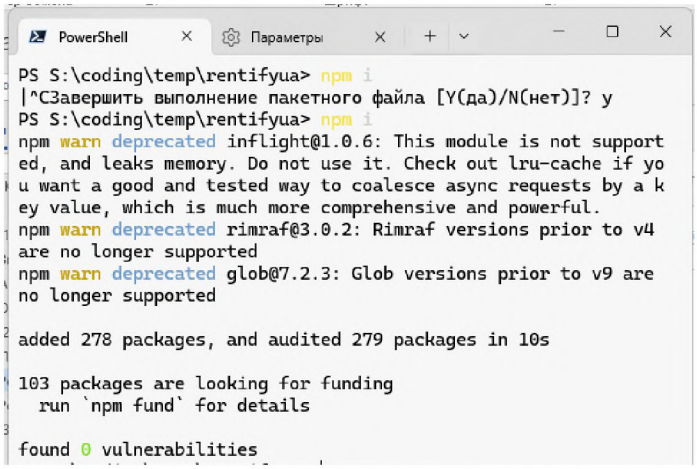

Рисунок 2.7 – Встановлення залежностей

4) запуск локального сервера: Для того, щоб почати роботу над проєктом та переглядати зміни в реальному часі, запускаємо локальний сервер за допомогою команди npm run dev. Це відкриє проєкт у браузері, де можна побачити всі внесені зміни без необхідності перезавантажувати сторінку (рис. 2.8).

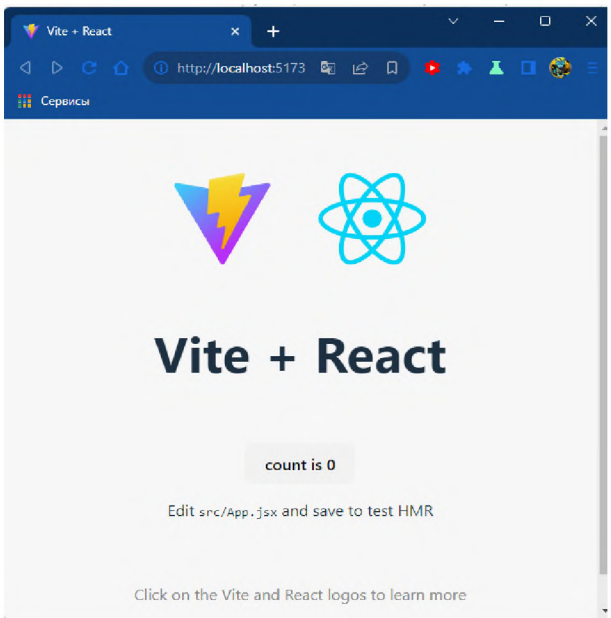

Рисунок 2.8 – Вигляд створеного проєкту

Після успішної ініціалізації проєкту за допомогою Vite та React, наступним важливим кроком є налаштування конфігураційних файлів для забезпечення зручності розробки та організації коду [27]. Ці файли допомагають оптимізувати процес розробки, роблячи його більш ефективним та структурованим. Розглянемо налаштування конфігураційних файлів vite.config.js та jsconfig.json.

Файл vite.config.js відіграє ключову роль у конфігурації збирача Vite(рис. 2.9). Цей файл дозволяє налаштовувати різні аспекти процесу збірки, підключати плагіни та задавати спеціальні параметри для вашого проєкту .

vite.config.js

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
import * as path from 'path'

// https://vitejs.dev/config/
export default defineConfig( config: {
  plugins: [react()],
  resolve: {
    alias: {
      '~app': path.resolve( paths: 'src/app'),
      '~entities': path.resolve( paths: 'src/entities'),
      '~features': path.resolve( paths: 'src/features'),
      '~pages': path.resolve( paths: 'src/pages'),
      '~shared': path.resolve( paths: 'src/shared'),
      '~widgets': path.resolve( paths: 'src/widgets')
    }
  },
  server: {
    host: true
  }
})
```

Рисунок 2.9 – Програмний код файлу vite.config.js

Підключення плагінів: В конфігурації використовується плагін @vitejs/plugin-react, який забезпечує підтримку роботи з React. Це дозволяє зручно інтегрувати JSX та інші специфічні можливості React у проєкт.

Налаштування alias: Використання alias дозволяє створювати короткі та зрозумілі шляхи до модулів. Наприклад, замість довгого шляху import MyComponent from '../../components/MyComponent', можна використовувати коротший шлях import MyComponent from '~components/MyComponent'. Це не лише спрощує процес імпорту, але й підвищує читабельність коду. Така

організація шляху відповідає потребам методології модульного підходу до розробки.

Налаштування сервера: Параметр host: true дозволяє запускати сервер розробки, який буде доступний по мережі. Це корисно для тестування на різних пристроях та забезпечення колаборації в команді розробників.

Файл jsconfig.json використовується для налаштування параметрів компіляції JavaScript проєктів та забезпечення автозаповнення шляхів і типів у редакторах коду (рис. 2.10).

jsconfig.json

```
{
  "compilerOptions": {
    "baseUrl": "./src",
    "paths": {
      "~app": ["./app"],
      "~app/*": ["./app/*"],
      "~processes": ["./processes"],
      "~processes/*": ["./processes/*"],
      "~pages": ["./pages"],
      "~pages/*": ["./pages/*"],
      "~widgets": ["./widgets"],
      "~widgets/*": ["./widgets/*"],
      "~features": ["./features"],
      "~features/*": ["./features/*"],
      "~entities": ["./entities"],
      "~entities/*": ["./entities/*"],
      "~shared": ["./shared"],
      "~shared/*": ["./shared/*"]
    }
  },
  "include": ["src"],
  "exclude": ["node_modules"]
}
```

Рисунок 2.10 – Програмний код файлу jsconfig.json

baseUrl: задає базову директорію для відносних шляхів, що робить процес імпорту модулів більш зручним та логічним. У нашому випадку базовий шлях встановлюється на папку ./src.

paths: визначає короткі шляхи для різних частин проєкту. Це налаштування працює у поєднанні з alias у файлі vite.config.js та забезпечує зручність при імпорті модулів у редакторі коду.

include та exclude: ці параметри визначають, які файли та папки повинні бути включені або виключені з компіляції. У нашому випадку включені всі файли з папки src, за винятком node_modules, щоб уникнути зайвих файлів у процесі компіляції.

При розробці кросплатформного вебсайту для оренди приміщень важливо дотримуватись певної архітектури, яка забезпечує легкість підтримки, масштабованість та зручність розробки. Для цього обрали FSD (Feature-Sliced Design) архітектуру, яка розбиває проєкт на логічні модулі відповідно до функціональних областей [28]. Це дозволяє розробникам працювати незалежно над різними частинами проєкту та полегшує управління кодовою базою.

Навіть невелика програма матиме багато файлів з кодом. Якщо тримати всі ці файли разом, з часом буде важко знайти потрібний файл. Крім того, у деяких випадках різні файли можуть мати однакові назви, але розташовуватися в різних папках. Якщо зберігати все в одній папці, доведеться перейменовувати ці файли, щоб уникнути конфлікту назв.

Для логічної організації файлів з кодом, створимо папку «src». Всередині цієї папки створимо підпапки, назви яких відповідатимуть призначенню файлів, що будуть у них зберігатися (рис. 2.11). На даному етапі розробки поділимо всі майбутні файли коду на категорії які нам вказує FSD архітектура яку обрали в першому розділі:

- «**app**», ця папка містить глобальні налаштування та конфігурації проєкту,
- «**entities**», містить модулі, що відповідають за бізнес-логіку та об'єкти предметної області,
- «**features**», містить модулі, які реалізують окремі функціональні можливості додатку,
- «**pages**», містить сторінки додатку, кожна з яких відповідає за відображення певного набору компонентів,
- «**shared**», містить спільні компоненти, утиліти, хуки та стилі, які використовуються в різних частинах додатку,

- «**widgets**», містить віджети, які є самостійними компонентами інтерфейсу та можуть використовуватися на різних сторінках.

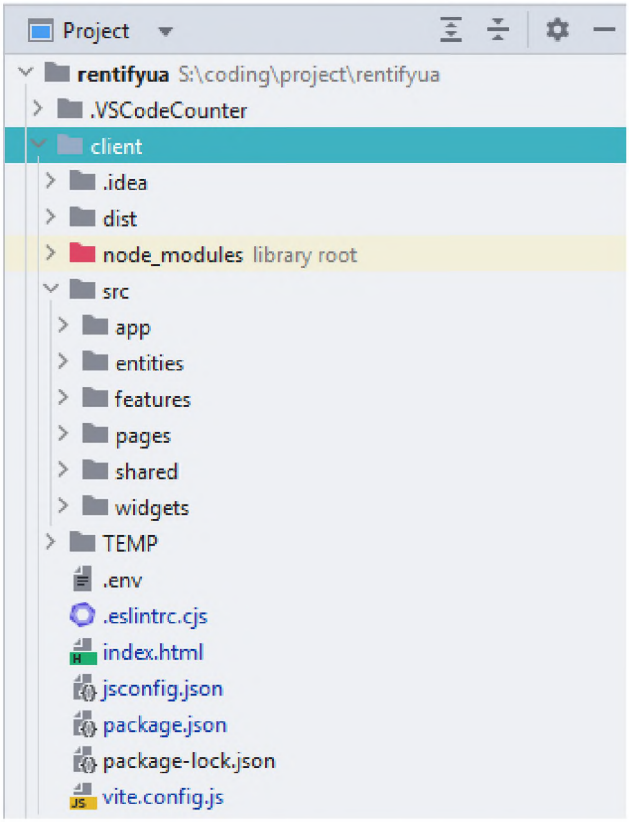

Рисунок 2.11 – Структура папки шаблону вебсайту

Після завершення налаштування проєкту та визначення файлової структури, наступним етапом є розробка клієнтської частини вебсайту. Одним із ключових аспектів цієї розробки є налаштування роутингу, який дозволяє переміщуватися між різними сторінками нашого додатку. Роутинг у вебдодатках забезпечує динамічне завантаження сторінок, підтримку глибоких посилань (deep linking) та покращення загального користувацького досвіду. У цьому проєкті використовуємо бібліотеку React Router для управління маршрутами.

У коді ми застосовуємо функцію lazy з React для динамічного імпорту компонентів, що сприяє скороченню часу завантаження сторінок. Крім того, ми використовуємо компонент Loadable, щоб показувати компонент під час завантаження основного компонента.

Кожен маршрут вказує шлях (path), компонент (component) та макет (layout), який буде використовуватись для відображення цього компоненту. Додатково,

деякі маршрути мають параметри onlyAuth та onlyNoAuth, які визначають доступність маршруту в залежності від стану аутентифікації користувача.

Для імплементації роутера створюємо компонент Router, який використовує Routes з react-router-dom для визначення та рендерингу маршрутів (рис. 2.12).

```
const Router = () => {
  const session = useSession((s) => s.currentSession)
  return (
    <Routes>
      {allRoutes.map(route => {
        const { path, component: Component, layout: Layout } = route
        return (
          <Route key={path} path={ROUTER_PATHS.INDEX}
          element={<Layout sessinon={session} />}>
          <Route
          key={path}
          path={path}
          element={<CheckRoute isAuth={!!session} route={route}>
          <Component isAuth={!!session} sessinon={session} />
          </CheckRoute>} /></Route>)})}
    </Routes>)}
```

Рисунок 2.12 – Програмний код файлу app-router.jsx

Компонент ListingsPage відповідає за відображення сторінки зі списком оголошень про оренду приміщень. Він використовує декілька інших компонентів для побудови інтерфейсу, включаючи SearchTab для пошуку, ListingsList для відображення списку оголошень та MapInput для інтеграції карти (рис. 2.13).

page.jsx

```
1  import { useQueryParams } from '~shared/lib/react.jsx'
2  import { ListingsList } from '~entities/listings'
3  import { SearchTab } from '~widgets/search-tab/index.js'
4  import { MapInput } from '~widgets/map/index.js'
5
6  export const ListingsPage = () => {
7    const urlParams = useQueryParams()
8    console.log(urlParams)
9
10   return (
11     <div className='listings'>
12       <SearchTab className='listings__tab' />
13       <ListingsList params={urlParams}/>
14       <MapInput className='listings__map'/>
15     </div>
16   )
17 }
```

Рисунок 2.13 – Програмний код файлу listings-page.jsx

Компонент ListingsList відповідає за отримання та відображення списку оголошень про оренду. Він використовує хук useQuery для здійснення запитів до API та отримання даних. У разі успішного запиту, компоненти PropertyCard використовуються для відображення кожного оголошення (рис. 2.14).

listings-list.jsx

```jsx
import { useQuery } from '~shared/lib/react.jsx'
import * as api from '~shared/api/requests.js'
import { PropertyCard } from '~entities/listings/ui/property-card.jsx'

Show usages  new *
export const ListingsList = ({ params }) => {
  const { data: listings, isLoading, error } = useQuery(api.getListings, params, options: { loading: true })
  return (
    <>
      {isLoading && <div>Loading...</div>}
      {error && <div>Error: {error.message}</div>}
      <ul className='listings__list'>
        {listings?.map((p) => <PropertyCard key={p.id} property={p}/>)}
      </ul>
    </>
  )
}
```

Рисунок 2.14 – Програмний код файлу listings-list.jsx

Компонент PropertyCard відповідає за відображення окремого оголошення про оренду. Він включає інформацію про нерухомість, таку як назва, кількість спалень, ванних кімнат, ліжок, ціна, доступність та інші зручності.

property-card.jsx

```jsx
import ...

Show usages  new *
export const PropertyCard = ({ property }) => {
  console.log('PropertyCard', property)
  const linkTo = `/rooms/${property.id}`
  return (
    <Link className="listings__item" to={linkTo}>
      <div className="listings__image">
        <img src={'http://localhost:5000/api/' + property.uploads[0]} alt={property.title} />
      </div>
      <div className="listings__info">
        <div className="listings__title">{property.name.uk}</div>
        <div className="listings__points"...>
        <div className="listings__amenities">City view | 3rd floor | Elevator | Parking </div>
        <div className="listings__status">
          <div className="listings__availability"...>
          <div className="listings__price"...>
        </div>
      </div>
    </Link>
  )
}
```

Рисунок 2.15 – Програмний код файлу property-card.jsx

Використання компонентного підходу в розробці клієнтської частини вебсайту забезпечує високу гнучкість та модульність коду. Компоненти, такі як ListingsPage, ListingsList та PropertyCard, дозволяють легко керувати та розширювати функціональність додатку. Завдяки цьому можемо створювати зручний і ефективний інтерфейс для користувачів нашого сайту.

2.3 Розробка серверної частини вебсайту

Розробка серверної частини вебсайту є критично важливим аспектом, який забезпечує обробку даних, виконання бізнес-логіки та взаємодію з базою даних. У цій частині ми розглянемо концепцію Onion Architecture (цибулинна архітектура), яка допоможе створити гнучку та масштабовану архітектуру для нашого проєкту, а також детально розглянемо доменні сутності, впровадження залежностей та реалізацію бізнес-логіки [29].

Onion Architecture – це архітектурний підхід, який фокусується на розділенні коду на шари з чітко визначеними залежностями. Основна ідея полягає в тому, що залежності спрямовані всередину, а не навпаки. Це означає, що зовнішні шари можуть залежати від внутрішніх, але внутрішні шари не можуть залежати від зовнішніх (рис. 2.16).

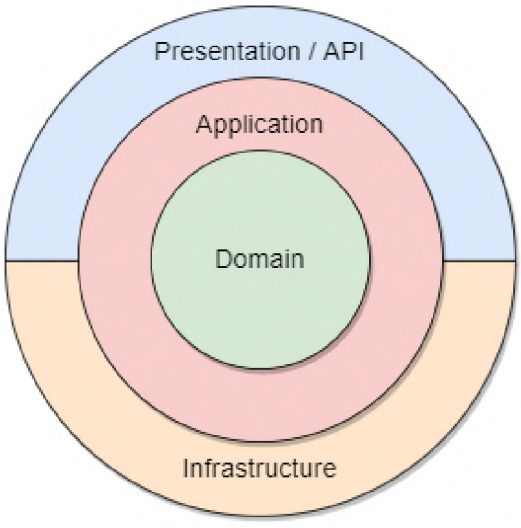

Рисунок 2.16 – Вигляд структури архітектури

Основні шари Onion Architecture:

1) доменний шар (Domain Layer) – містить доменні сутності та бізнес-логіку;

2) шар застосування (Application Layer) – містить інтерфейси, які використовують доменні сутності для виконання бізнес-логіки;

3) інфраструктурний шар (Infrastructure Layer) – містить реалізації інтерфейсів та взаємодію з зовнішніми системами (наприклад, базами даних, API) [29].

Доменні сутності – це основні об'єкти бізнес-логіки, які відображають реальні об'єкти та їх поведінку у нашій системі. Кожна доменна сутність повинна бути самодостатньою та інкапсулювати свою логіку (рис. 2.17).

model.js

```
Show usages  new *
class PropertyModel {
  Show usages  new *
  constructor (data) {
    this.id = data.id
    this.userId = data.userId
    this.createdAt = data.createdAt
    this.updatedAt = data.updatedAt || null
    this.name = data.name || {}
    this.description = data.description || {}
    this.deletedAt = data.deletedAt || null
    this.isActive = data.isActive
    this.status = data.status
    this.creationStatus = data.creationStatus
    this.verificationStatus = data.verificationStatus
    this.geoLocation = data.geoLocation || {}
    this.price = data.price
    this.bedrooms = data.bedrooms
    this.bathrooms = data.bathrooms
    this.beds = data.beds
    this.parkingSpaces = data.parkingSpaces
    this.maxGuests = data.maxGuests
    this.propertyTypeId = data.propertyTypeId
    this.amenities = data.amenities?.filter(item => item !== null) || []
    this.uploads = data.uploads || []
    this.reviews = data.reviews || []
  }
}
```

Рисунок 2.17 – Програмний код файлу property-model.js

Впровадження залежностей (Dependency Injection) – це патерн, який дозволяє підвищити модульність та тестованість коду. Це досягається шляхом

передачі залежностей об'єктам через конструктор або властивості, а не створення їх безпосередньо всередині об'єктів.

У контексті Onion Architecture, впровадження залежностей дозволяє нам легко замінювати реалізації інтерфейсів для різних середовищ (наприклад, для тестування або продакшн-оточення) (рис. 2.18).

DI.js

```
19
20  const Database = require('./db')
21
22  const container = awilix.createContainer()
23  container.register( nameAndRegistrationPair: {
24    userService: awilix.asClass(UserService),
25    userRepository: awilix.asClass(PostgresUserRepository),
26    authService: awilix.asClass(AuthService),
27    authRepository: awilix.asClass(PostgresAuthRepository),
28    propertyService: awilix.asClass(PropertyService),
29    propertyRepository: awilix.asClass(PostgresPropertyRepository),
30    imageService: awilix.asClass(ImageService),
31    imageRepository: awilix.asClass(PostgresImageRepository),
32    searchMicroservice: awilix.asClass(SearchMicroservice),
33    tokenService: awilix.asClass(JwtService),
34    mailService: awilix.asClass(MailerService),
35    multerService: awilix.asClass(MulterService),
36    sharpService: awilix.asClass(SharpService),
37    db: awilix.asValue(new Database())
38  })
```

Рисунок 2.18 – Програмний код файлу DI.js

Бізнес-логіка – це правила та процедури, які визначають, як дані можуть бути створені, зберігатися та змінюватися. В Onion Architecture бізнес-логіка реалізується навколо доменних сутностей у доменному шарі. Це означає, що всі бізнес-правила та обробка даних зосереджені в одному місці, що полегшує підтримку та модифікацію програми (рис. 2.19).

Доменний шар містить основні бізнес-об'єкти, такі як сутності та агрегати. Сутності представляють основні об'єкти в бізнесі, що мають унікальну ідентичність, наприклад, користувачі, замовлення або продукти. Агрегати – це групи пов'язаних сутностей, які обробляються як єдине ціле.

property.service.js

```
const APIException = require('~infrastructure/helpers/api-error')
new *
module.exports = class PropertyService {
  Show usages  new *
  constructor ({ propertyRepository, imageRepository, sharpService, searchMicroservice }) {
    this.propertyRepository = propertyRepository
    this.imageRepository = imageRepository
    this.imageProcessor = sharpService
    this.searchMicroservice = searchMicroservice
  }

  Show usages  new *
  async createProperty (data, userId) {
    const property = await this.propertyRepository.createProperty(userId)
    const geoLocation = await this.propertyRepository.createGeoLocation(data, property.id)
    return this.propertyRepository.getPropertyById(property.id)
  }

  Show usages  new *
  async getPropertyById (id, userId) {
    const property = await this.propertyRepository.getPropertyById(+id)
    if (!property) throw new APIException('PROPERTY_NOT_FOUND')
    const isOwner = property.userId === userId
    const uploads = await this.imageRepository.getImagesByPropertyId(+id)
    return { ...property, uploads: this.imageProcessor.mapImages(uploads), isOwner }
  }
```

Рисунок 2.19 – Програмний код файлу property.service.js

Інфраструктурний шар забезпечує взаємодію з зовнішніми системами, такими як база даних або інші сервіси. Він містить реалізації інтерфейсів, визначених у шарі застосування (рис. 2.20). Наприклад, якщо у шарі застосування визначений інтерфейс для доступу до даних, у інфраструктурному шарі буде знаходитися його конкретна реалізація, що використовує певну базу даних або інший механізм збереження даних.

Крім того, інфраструктурний шар відповідає за обробку запитів, що надходять від користувачів чи інших систем. Він забезпечує ефективну комунікацію між різними компонентами програмної системи та контролює доступ до ресурсів. Цей шар також відповідає за управління ресурсами, такими як пам'ять, обчислювальна потужність та мережеві ресурси, для забезпечення оптимальної продуктивності та надійності системи.

Інфраструктурний шар включає засоби моніторингу та логування для виявлення і усунення несправностей. Він відіграє ключову роль у забезпеченні безпеки даних та захисті системи від зовнішніх загроз.

```
repository.js
new *
module.exports = class PostgresPropertyRepository extends PropertyRepository {
  Show usages  new *
  constructor ({ db }) {
    super()
    this.db = db
    this.propertyMapper = PropertyMapper
    this.locationMapper = LocationMapper}
  Show usages  new *
  async createProperty (userId) {
    const query = `
      INSERT INTO listings (user_id, creation_status)
      VALUES ( $1, 'STRUCTURE' )
      RETURNING *`
    const values = [userId]
    const listings = await this.executeQuery(query, values,
      this.propertyMapper.toEntity)
    return listings[0]
  }
```

Рисунок 2.20 – Програмний код файлу repository.js

Запуск HTTP сервера є важливим етапом розробки серверної частини вебсайту. Для цього використовуються різні фреймворки і бібліотеки, які спрощують процес створення серверної частини. У нашому випадку ми використовуємо фреймворк Express, який дозволяє швидко і ефективно створювати вебсервери на базі Node.js.

Основний серверний файл (server.js) відповідає за конфігурацію і запуск сервера. Він включає в себе налаштування різних middleware, підключення роутерів та обробку помилок (рис. 2.21).

```
server.js
module.exports = async () => {
  const app = express()
  app.use(scopePerRequest(container))
  app.use('/pp', express.static( root: 'pp'))
  app.use(express.json())
  app.use(cookieParser())
  app.use(cors( o: {
    origin: CLIENT,
    credentials: true
  }))
  app.use(userAgentMiddleware)
  app.use(decoratorMiddleware)
  app.use('/api', userRouter())
  app.use('/api', authRouter())
  app.use('/api', propertyRouter())
  app.use('/api', imageRouter())
  app.use(errorMiddleware)
  app.listen(PORT,
    callback: () => console.log(`server started ${PORT}`))
}
```

Рисунок 2.21 – Програмний код файлу server.js

Роутер (router.js) відповідає за обробку конкретних маршрутів. Тут налаштовуються шляхи та методи запитів, які будуть оброблятися відповідними контролерами (рис. 2.22).

server.js router.js

```
const express = require('express')
const router = express.Router()
const DI = require('~infrastructure/api/DI.api')
const validationSchema = require('~infrastructure/middlewares/validate-middleware')
const { property, geoProperty } = require('./validation.schema')

module.exports = () => {
  const controller = DI.resolve( name: 'propertyController')
  const authCheck = DI.resolve( name: 'authExistsMiddleware')
  const userCheck = DI.resolve( name: 'userExistsMiddleware')
  router.get('/property/:propertyId', userCheck, controller.getProperty.bind(controller))
  router.get( path: '/listings', controller.getProperties.bind(controller))
  router.get( path: '/listings-region/:query', controller.searchRegion.bind(controller))
  router.post('/property/create', authCheck, validationSchema(geoProperty), controller.createProperty.bind(controller))
  router.patch('/property/:propertyId/active', authCheck, controller.toggleProperty.bind(controller))
  router.patch('/property/:propertyId/deactivate', authCheck, controller.toggleProperty.bind(controller))
  router.put('/property/:propertyId', authCheck, validationSchema(property), controller.updateProperty.bind(controller))
  router.delete('/property/:propertyId', authCheck, controller.deleteProperty.bind(controller))
  router.get( path: '/geo', controller.getGeo.bind(controller))
  router.get( path: '/listings/types', controller.getTypes.bind(controller))
  router.get( path: '/listings/amenities', controller.getAmenities.bind(controller))
  return router
}
```

Рисунок 2.22 – Програмний код файлу router.js

Контролер (controller.js) включає методи для створення, оновлення, видалення та отримання інформації про нерухомість. вони обробляють HTTP-запити, виконують бізнес-логіку та повертають відповідні відповіді клієнту. Контролери взаємодіють з сервісами та репозиторіями для виконання запитів і реалізації бізнес-логіки. Нижче ми розглянемо приклад реалізації контролера для роботи з об'єктами нерухомості.

Серверна частина, заснована на Onion Architecture, забезпечує модульність та підтримуваність коду, дозволяючи ефективно обробляти запити користувачів та взаємодіяти з базою даних. Створення доменних сутностей, впровадили залежності та налаштували HTTP сервер, який обробляє запити, відповідає за маршрутизацію та забезпечує безпеку. Це забезпечує надійну та ефективну роботу вебсайту, дозволяючи фронтенду коректно відображати та обробляти дані для зручного.

РОЗДІЛ 3
РОЗГОРТАННЯ ТА ВПРОВАДЖЕННЯ ВЕБСАЙТУ

3.1 Компіляція клієнтського додатку

У другому розділі ми детально розглянули Vite та його переваги як інструменту для розробки фронтенд-додатків. Vite забезпечує швидке та ефективне розгортання проєктів, завдяки своїм можливостям миттєвого запуску сервера розробки, гарячого перезавантаження модулів та оптимізованої збірки. Тепер ми розглянемо, як за допомогою Vite можна скомпілювати клієнтський додаток для розгортання у продакшн середовищі [25].

Першим кроком є ініціалізація проєкту та встановлення всіх необхідних залежності. Це забезпечує, що всі потрібні пакети встановлені та готові до використання (рис. 3.1).

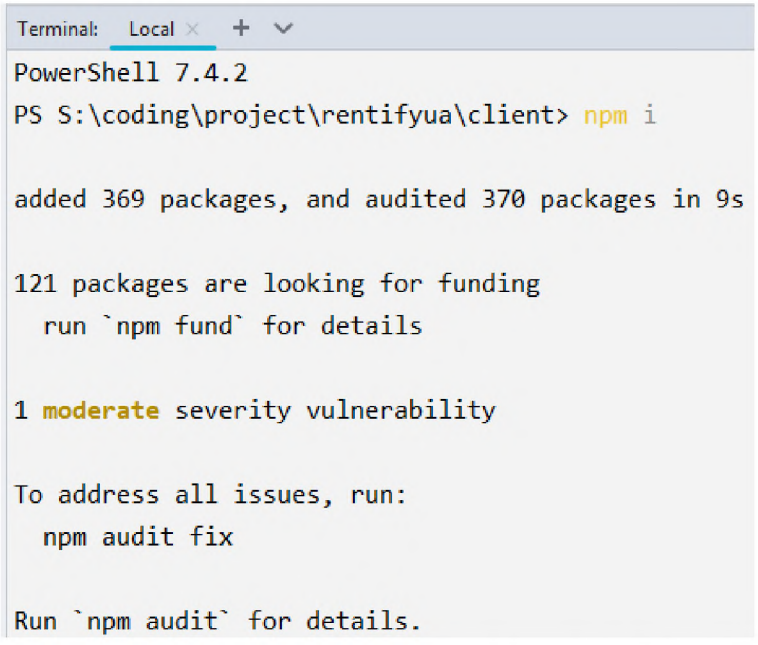

Рисунок 3.1 – Встановлення залежності проєкту

Після того, як проєкт налаштовано, ми можемо приступити до створення фінальної збірки клієнтської частини додатку. Для цього використовується команда npm run build

Ця команда запускає процес збірки, який створює оптимізовану версію додатку у папці dist. Збірка включає мінімізацію коду, розбиття та інші оптимізації, що забезпечують швидке завантаження та високу продуктивність додатку у продакшн середовищі.

Vite автоматично виконує ряд оптимізацій під час збірки, включаючи мінімізацію коду та розбиття. Проте, можна додатково налаштувати процес збірки, щоб врахувати специфічні вимоги проєкту. Це може включати налаштування код-спліттингу, використання плагінів для оптимізації зображень та інших статичних ресурсів, а також налаштування коду для підтримки старих браузерів.

Після успішної збірки ми маємо фінальний розмір файлів додатку, який становить 875 Кб. Для порівняння, стандартні розміри збірок сучасних веб-додатків зазвичай знаходяться в межах 500 Кб. до 2 Мб.

Розмір збірки має значення з кількох причин:

1) Швидкість завантаження сторінки: Чим менше розмір файлів, тим швидше вони завантажуються. Це особливо важливо для користувачів з повільним інтернет-з'єднанням. Швидке завантаження сторінок покращує користувацький досвід, знижуючи час очікування.

2) Показники SEO: Швидкість завантаження сайту впливає на його рейтинг у пошукових системах. Пошукові системи, такі як Google, враховують час завантаження сторінок при ранжуванні сайтів. Оптимізований сайт з меншим розміром файлів може отримати кращий рейтинг, що підвищує видимість у пошукових результатах [30].

3) Мобільні користувачі: Зростання використання мобільних пристроїв для доступу до інтернету робить оптимізацію розміру файлів ще більш важливою. Мобільні користувачі часто мають обмежену швидкість інтернету та обмежені обсяги даних. Менші файли допомагають забезпечити швидке завантаження і зменшити споживання мобільних даних.

4) Зменшення витрат на трафік: Менші файли означають менше використання трафіку. Це важливо як для користувачів, так і для власників

вебсайту, які можуть знизити витрати на трафік і хостинг.

5) Поліпшення продуктивності: Зменшення розміру файлів дозволяє браузеру швидше обробляти і рендерити сторінки. Це особливо важливо для складних веб-додатків з великою кількістю інтерактивних елементів.

Поточний розмір збірки становить 875 Кб, що є оптимальним і відповідає сучасним стандартам розробки веб-додатків.

Після завершення збірки важливо перевірити працездатність додатку перед його розгортанням. Для цього можна використовувати локальний сервер, щоб переглянути згенеровану збірку. Одним з простих способів є використання пакету serve, який дозволяє запускати локальний сервер для перевірки збірки (рис. 3.2).

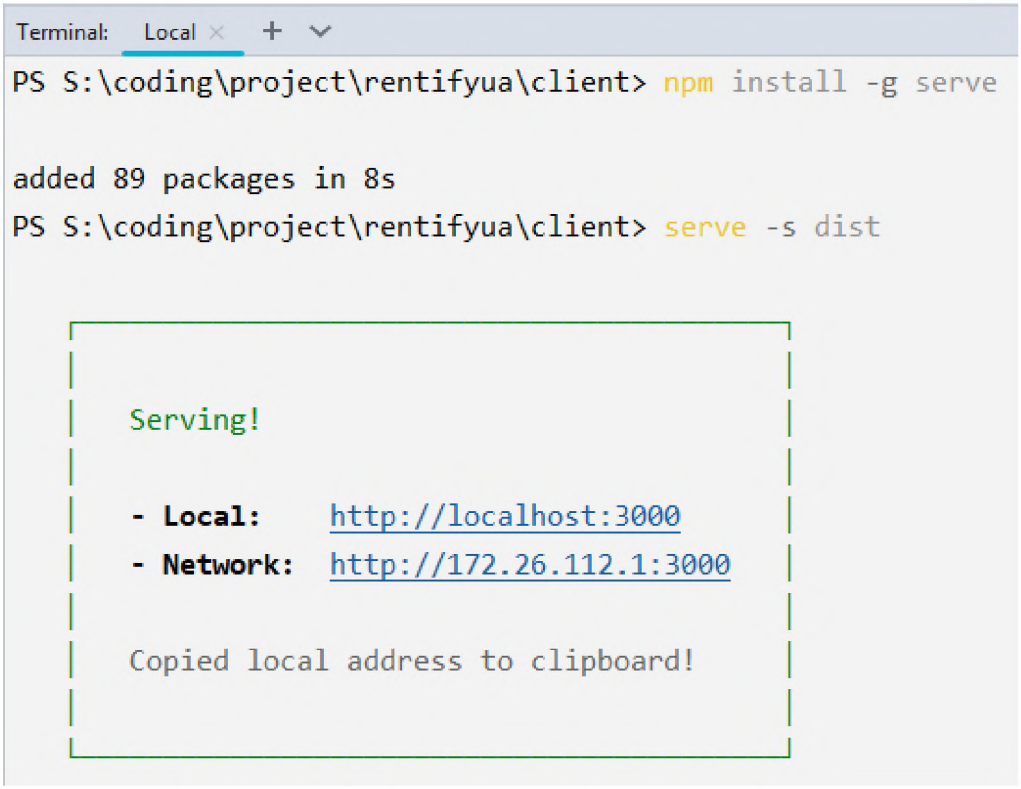

Рисунок 3.2 – Запуск локального серверу

Після запуску команди serve -s dist запускає сервер, який служить статичні файли з папки dist. Перейшовши за адресою http://localhost:3000, є можливість переглянути додаток і переконатися, що він працює належним чином.

Цей процес дозволяє виявити потенційні проблеми та помилки перед розгортанням додатку на живому сервері. Перегляд сайту в локальному середовищі забезпечує впевненість у тому, що всі функції працюють коректно, а користувачі отримають оптимальний досвід при роботі з додатком.

3.2 Встановлення вебсервера на VPS

Розгортання вебсервера на VPS є важливим етапом впровадження вебсайту, що вимагає ретельного планування та виконання кількох ключових завдань. Початок цього процесу включає вибір та реєстрацію доменного імені, налаштування системи керування трафіком, встановлення необхідного програмного забезпечення на сервері та завантаження файлів сайту.

Вибір доменного імені є першочерговим кроком. Використання сервісу GoDaddy для реєстрації домену зумовлено його популярністю та надійністю. GoDaddy пропонує широкий вибір доменних зон, конкурентоспроможні ціни та зручний інтерфейс для управління доменами. Завдяки цьому, користувачі можуть легко знайти та зареєструвати бажане доменне ім'я, забезпечуючи тим самим основу для свого вебсайту [31].

Вибір доменного імені rentifyua.site був свідомим та ретельно обдуманим рішенням. Цей домен максимально відображає суть діяльності, пов'язаної з орендою, та чітко ідентифікує бізнес на українському ринку. Крім того, доменна зона .site є однією з найпопулярніших і найбільш довірених у світі, що додає бренду додаткової надійності та впізнаваності. Використання сервісу GoDaddy для реєстрації домену дозволило скористатися численними перевагами, такими як широкий вибір доменних зон, конкурентоспроможні ціни та зручний інтерфейс для управління доменами. Це забезпечило легкий процес реєстрації та управління новим доменом, що є важливим кроком у створенні вебсайту та подальшого розвитку бізнесу (рис. 3.3).

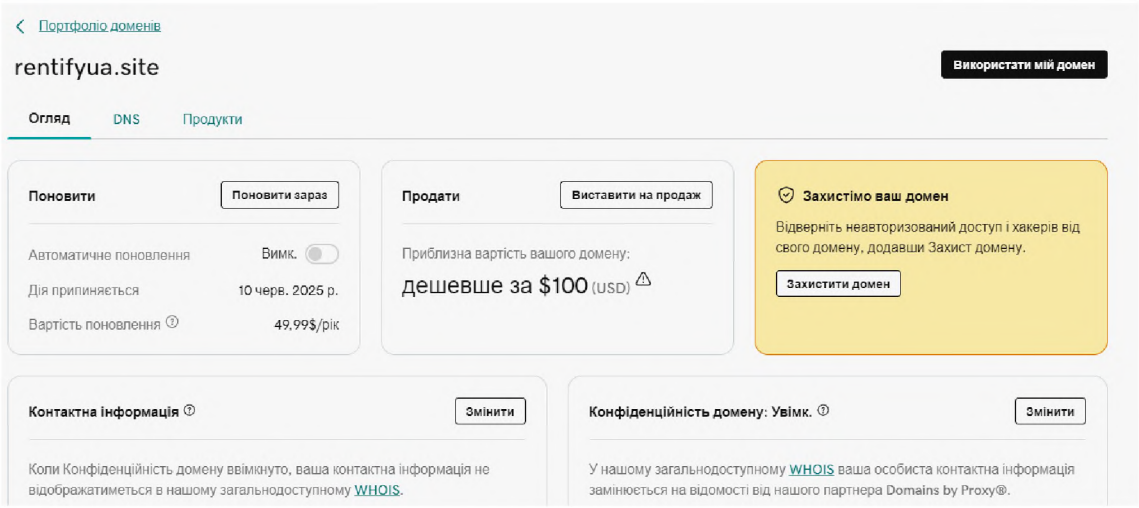

Рисунок 3.3 – Інформація зареєстрованого домену

Наступним кроком є налаштування Cloudflare для обраного домену. Cloudflare надає можливість значно покращити продуктивність та безпеку вебсайту через використання глобальної мережі CDN (Content Delivery Network). Використання Cloudflare дозволяє забезпечити швидке завантаження сторінок, захист від DDoS-атак та оптимізацію мережевого трафіку. Важливою перевагою є також можливість використання безкоштовного SSL-сертифікату для забезпечення безпечного з'єднання [32].

Для налаштування Cloudflare необхідно створити обліковий запис, додати домен rentifyua.site до системи та змінити DNS-сервери домену на ті, які надає Cloudflare. Це дозволяє керувати трафіком та забезпечувати додатковий захист сайту (рис. 3.4).

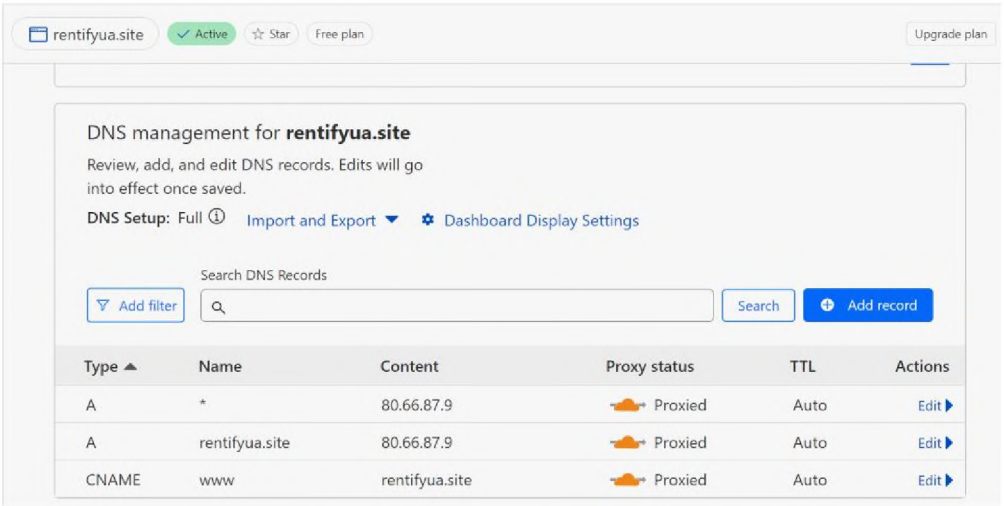

Рисунок 3.4 – Налаштування DNS-серверів

Після завершення налаштувань домену та Cloudflare, переходимо до оренди VPS та налаштування серверного середовища. Використання VPS забезпечує повний контроль над сервером та його конфігурацією, що є важливим для забезпечення належної роботи вебсайту. Для підключення до VPS використовується SSH, що забезпечує безпечний доступ до сервера.

Підключившись до сервера через SSH, ми можемо почати налаштовувати серверне середовище (рис. 3.5). Це забезпечує захищене підключення до сервера, дозволяючи виконувати подальші налаштування.

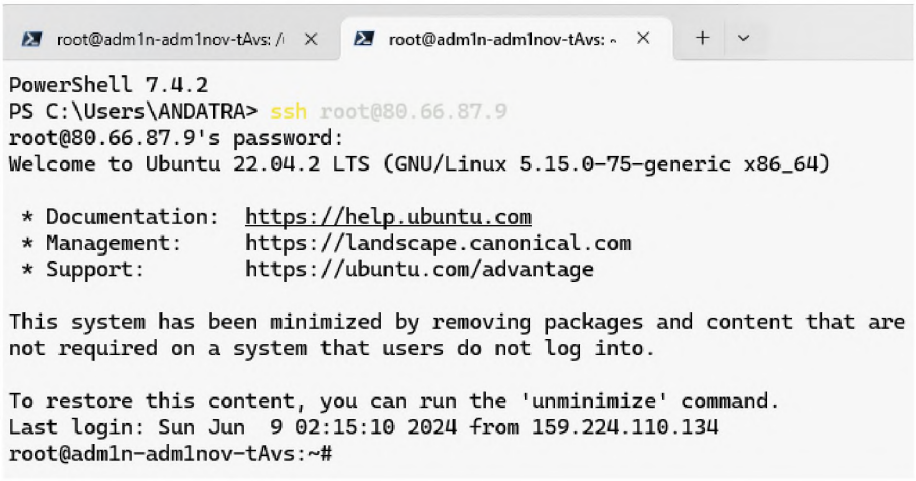

Рисунок 3.5 – Підключення до серверу по SSH протоколу

Після підключення, першим кроком є оновлення системи, щоб переконатися, що всі пакети актуальні. Це допомагає уникнути потенційних проблем з безпекою та сумісністю (рис. 3.6) [33].

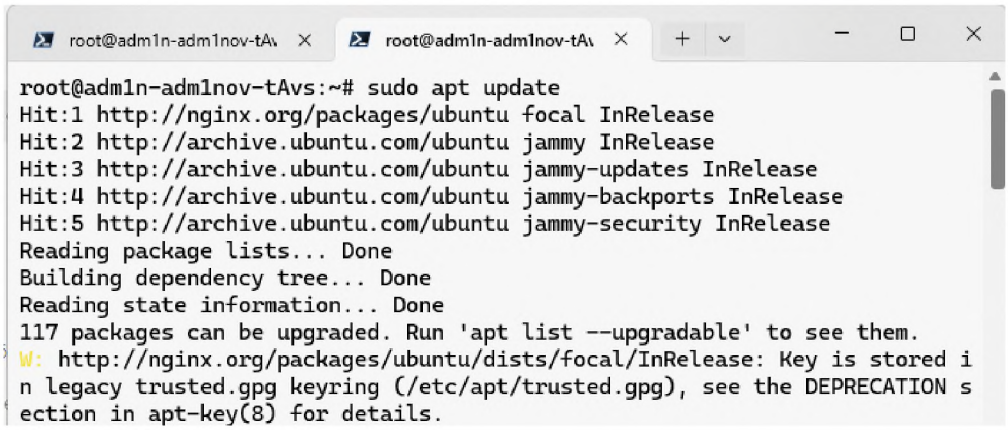

Рисунок 3.6 – Оновлення системи linux

Далі необхідно встановити Nginx – високопродуктивний вебсервер, який буде використовуватися для обробки HTTP-запитів і проксіювання їх до нашого додатку (рис. 3.7) [34].

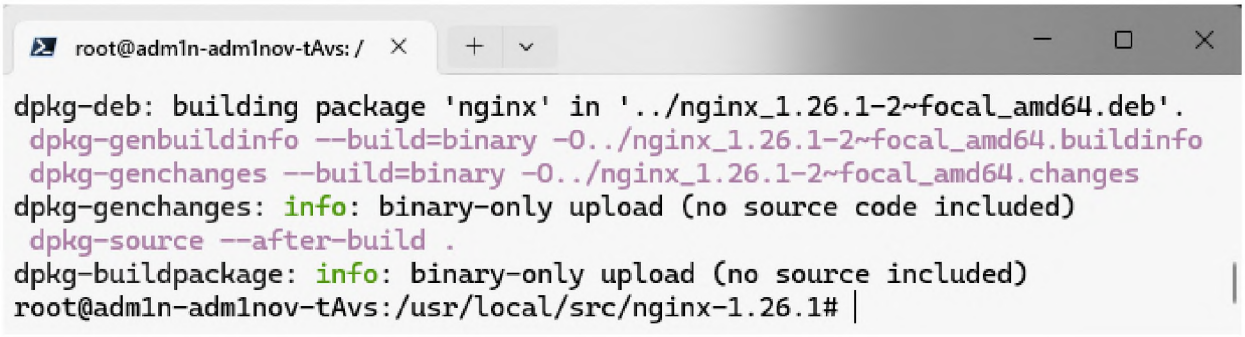

Рисунок 3.7 – Встановлення nginx

Після встановлення Nginx, необхідно налаштувати його для роботи з нашим доменом. Створюємо конфігураційний файл для нашого сайту (рис. 3.8).

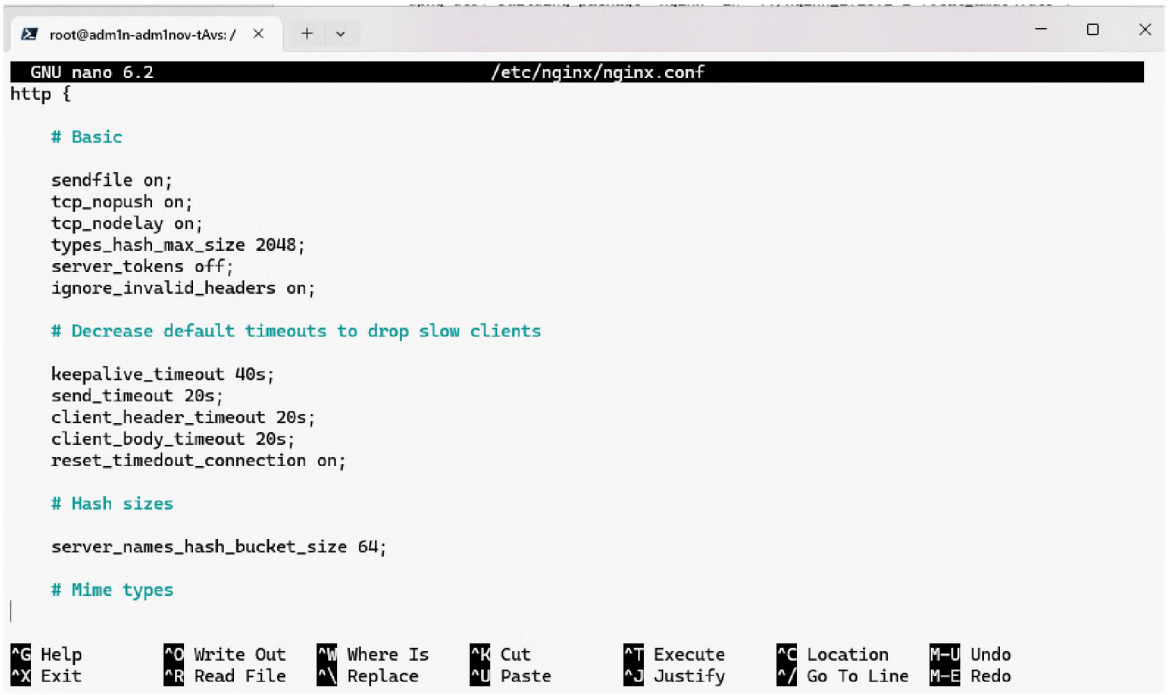

Рисунок 3.8 – Конфігураційний файл nginx

Після внесення змін до конфігураційного файлу Nginx, важливо перевірити, чи немає помилок у нових налаштуваннях, потім перезапустити сервіс (рис. 3.9).

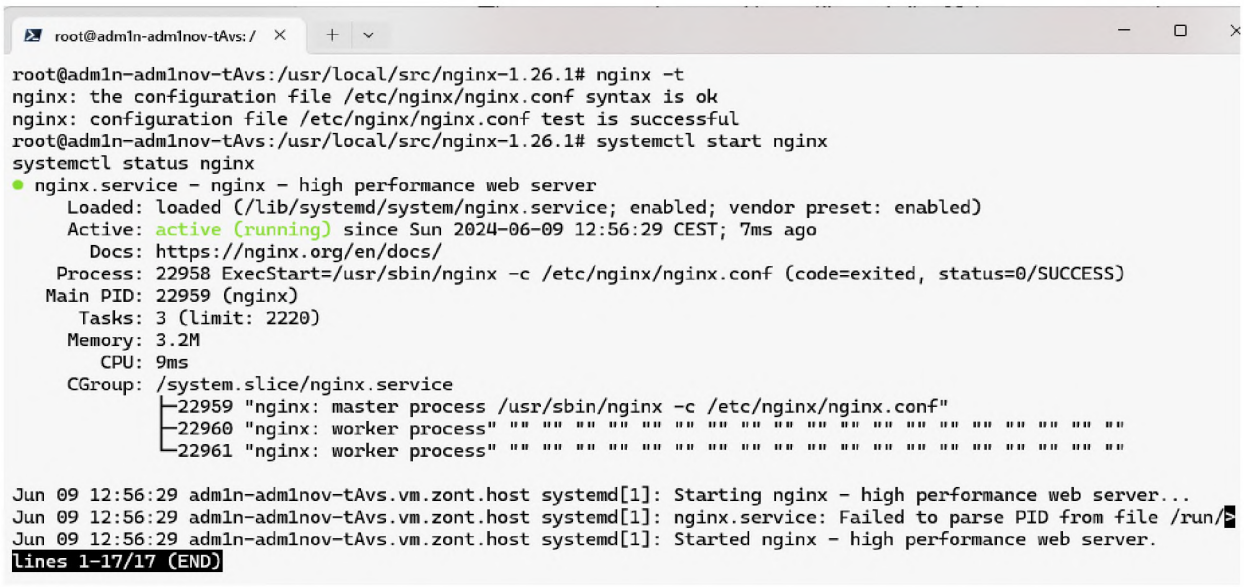

Рисунок 3.9 – Перевірка конфігурації та запуск nginx

Для того, щоб перевірити, чи працює Nginx належним чином, створимо тестову HTML-сторінку, яку будемо відображати на нашому сервері. Спочатку створюємо директорію «**/var/www/rentifyua.com/html**», далі необхідно створити HTML-файл з тестовим кодом (рис. 3.10) [34].

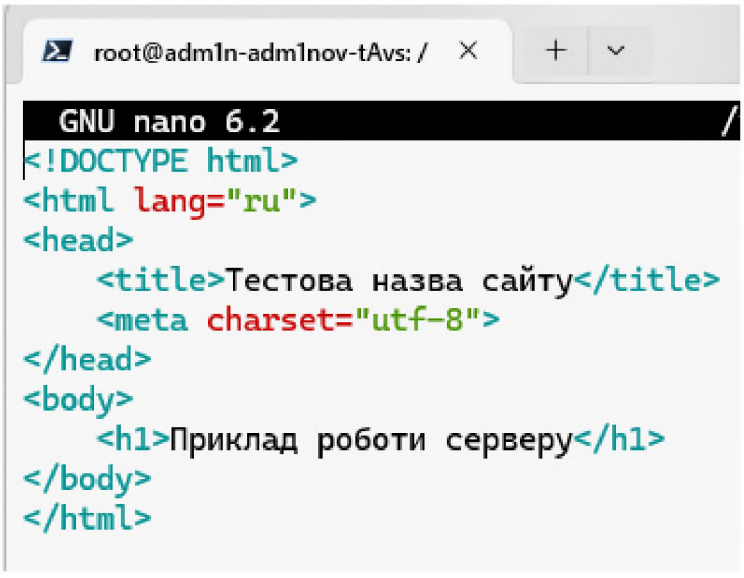

Рисунок 3.10 – Тестовий код для перевірки роботи сайту

Після збереження файлу спробуємо перейти за адресом https://rentifyua.site, щоб побачити створену тестову сторінку (рис. 3.11).

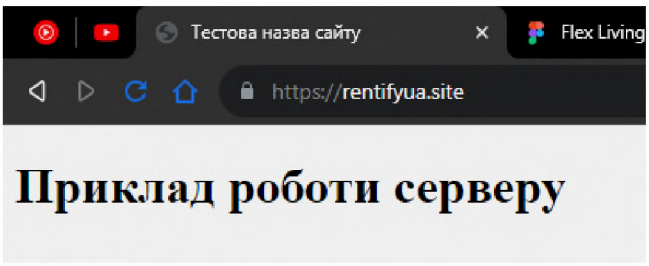

Рисунок 3.11 – Вигляд тимчасової сторінки

Після налаштування серверного середовища та успішного тестування Nginx, ми можемо завантажити наші відкомпільовані файли на сервер. Для цього використовуємо rsync, який дозволяє швидко та ефективно синхронізувати файли між локальним комп'ютером та сервером (рис. 3.12).

Рисунок 3.12 – Завантаження файлів на сервер

Після завершення синхронізації файлів, перейдемо за адресою https://rentifyua.site, щоб перевірити, що сайт працює належним чином та відображає всі необхідні дані (рис. 3.13).

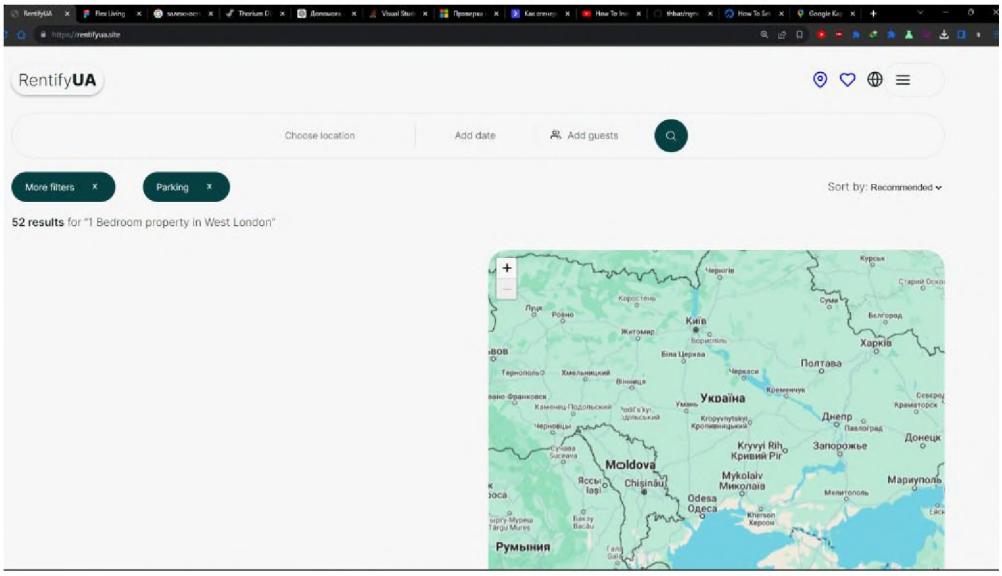

Рисунок 3.13 – Вигляд сторінки пошуку

3.3 Тестування функціювання вебсайту

Після успішного розгортання вебсайту на VPS, важливо продемонструвати його функціонування та основні можливості. Вебсайт RentifyUA створений для оренди житлових приміщень і пропонує зручний інтерфейс для користувачів та орендодавців.

Головна сторінка сайту RentifyUA представляє основну інформацію про платформу, включаючи огляд доступних для оренди приміщень, інформацію про переваги використання платформи та рекомендації для нових користувачів. Інтуїтивний дизайн дозволяє швидко знайти потрібну інформацію та перейти до інших розділів сайту (рис. 3.14).

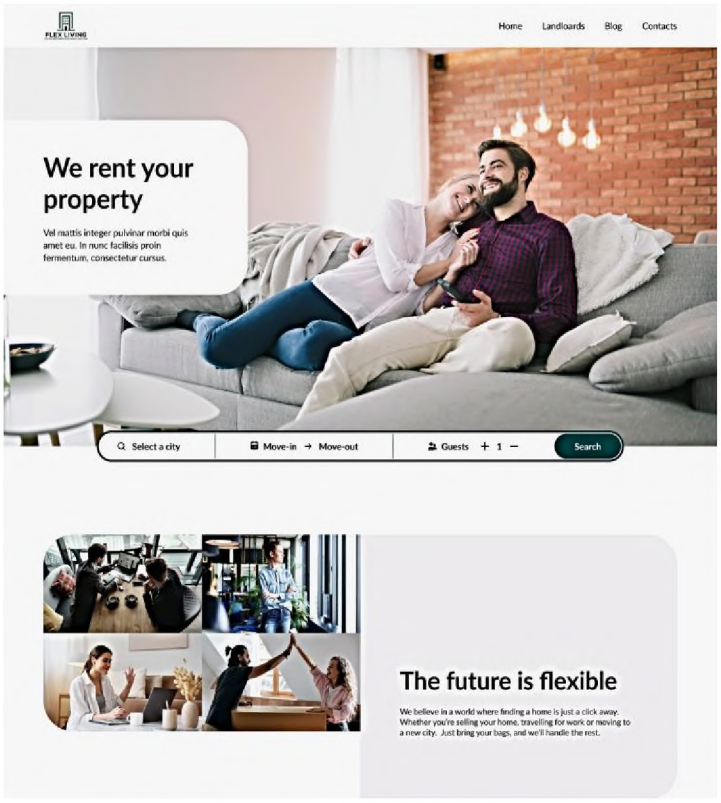

Рисунок 3.14 – Вигляд начальної сторінки проєкту

RentifyUA пропонує зручний механізм реєстрації та авторизації користувачів. Нові користувачі можуть створити обліковий запис, заповнивши просту форму реєстрації. Після підтвердження електронної пошти, користувачі можуть увійти на сайт за допомогою своїх облікових даних.

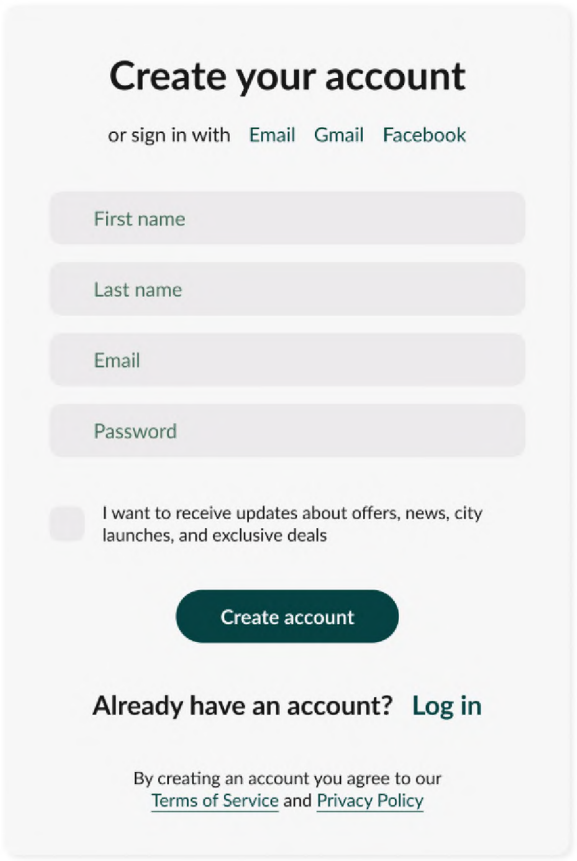

Рисунок 3.15 – Вигляд форми реєстрації

На сайті реалізовано потужний механізм пошуку та фільтрації об'єктів нерухомості. Користувачі можуть задавати різні критерії пошуку, такі як місцезнаходження, ціна, кількість спалень, тип нерухомості тощо. Це дозволяє швидко знайти відповідні варіанти для оренди.

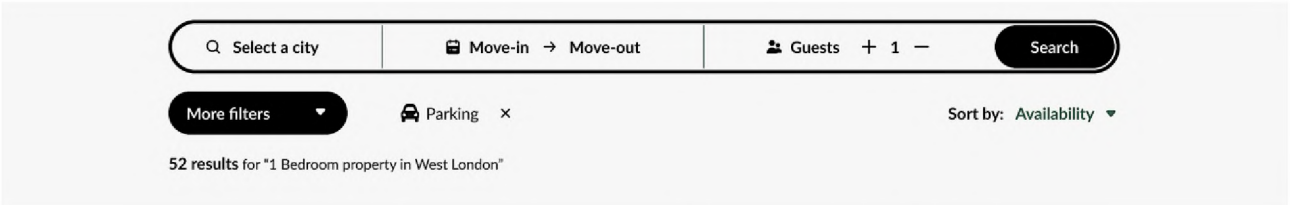

Рисунок 3.16 – Фільтрація нерухомості сайту

При виборі конкретного об'єкта, користувач може переглянути детальну інформацію про нього, включаючи фото, опис, ціни та умови оренди. Також доступні відгуки інших користувачів, що допомагають прийняти рішення про оренду.

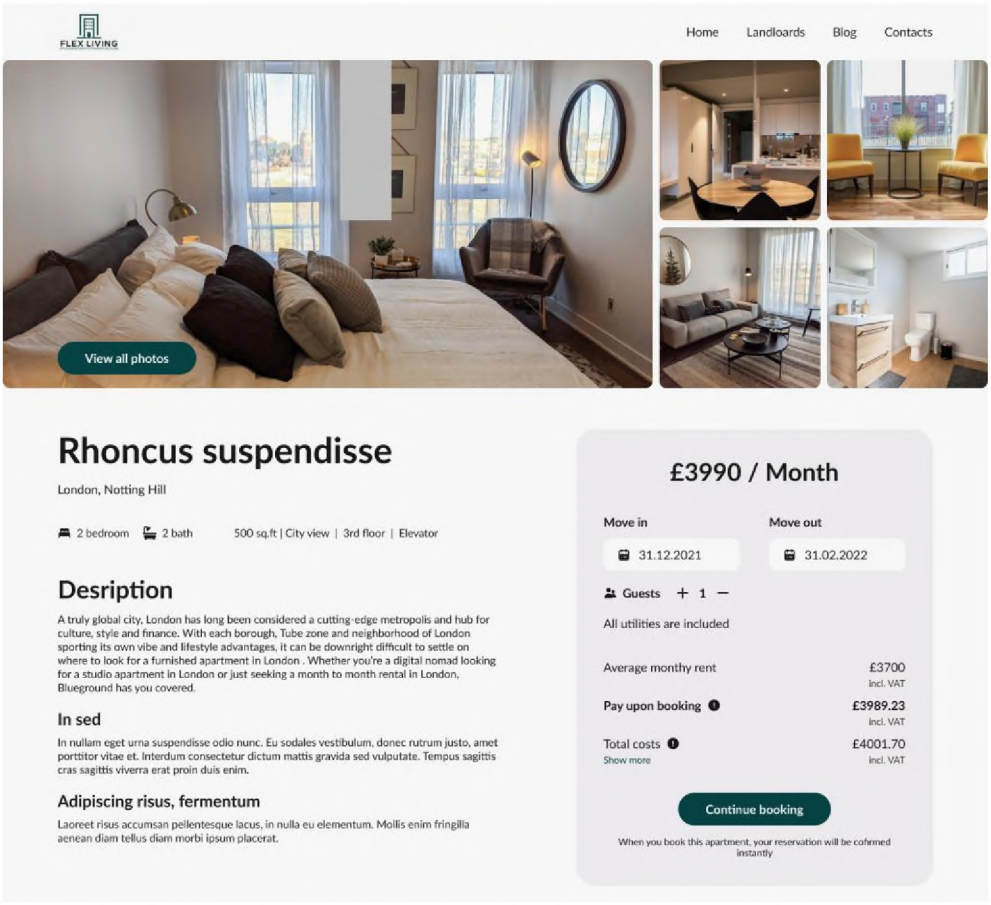

Рисунок 3.17 – Інформаційна сторінка про нерухомість

Орендодавці можуть створювати та управляти своїми оголошеннями через особистий кабінет. Вони можуть додавати нові об'єкти, редагувати існуючі оголошення, переглядати статистику та відповіді на свої оголошення. Це дозволяє ефективно керувати пропозиціями на платформі.

Для зручності спілкування між орендодавцями та орендарями реалізовано систему повідомлень. Користувачі можуть надсилати та отримувати повідомлення безпосередньо через платформу, що забезпечує зручний та безпечний канал комунікації.

Таким чином, вебсайт RentifyUA пропонує зручний та функціональний інтерфейс для оренди житлових приміщень, забезпечуючи користувачам простоту використання та ефективність. Всі основні функції, включаючи реєстрацію, пошук, перегляд деталей, управління оголошеннями, комунікацію та систему рейтингів, створюють комплексне рішення для орендодавців та орендарів.

3.4 Економічна оцінка впровадження

Впровадження вебсайту включає декілька аспектів, що вимагають фінансових інвестицій. Розробка та розгортання проєкту потребують не тільки часу та зусиль розробників, але й певних витрат на інфраструктуру, програмне забезпечення та підтримку. Розглянемо основні витрати, пов'язані з впровадженням вебсайту RentifyUA.

Першочергові витрати включають витрати на доменне ім'я та хостинг. Ми обрали доменне ім'я rentifyua.sites на платформі GoDaddy, яка є одним з найпопулярніших реєстраторів. Доменне ім'я потребує щорічної оплати. VPS надає більше контролю та гнучкості в порівнянні зі звичайним хостингом, і ми обрали середньорівневий план для нашого проєкту:

$$Vd = 15\ usd/\text{рік}\ , \tag{3.1}$$

де Vd – витрати на домен.

$$Vv = 20\ usd/\text{місяць}\ , \tag{3.2}$$

де Vv – витрати на VPS.

Також важливими є витрати на розробку сайту. Це включає оплату праці розробників, дизайнерів, а також витрати на тестування та забезпечення якості. За даними ресурсу Upwork [30], середня погодинна ставка веб-розробника коливається від $20 до $50 за годину, залежно від досвіду та складності проєкту.

Загальна вартість включає як разові витрати на розробку, так і щомісячні та річні витрати на підтримку інфраструктури. Розглянемо ці витрати докладніше.

Розробка вебсайту RentifyUA включає кілька етапів: проєктування, реалізація, тестування та впровадження. Це потребує залучення команди фахівців, таких як веб-розробники, дизайнери, тестувальники та проєктні менеджери. Вартість їхніх послуг може варіюватися, але для нашого проєкту ми розрахували такі витрати:

– Frontend розробка: 200 годин * $30/година = $6000

– Backend розробка: 200 годин * $30/година = $6000
– Дизайн: 100 годин * $25/година = $2500
– Тестування: 100 годин * $20/година = $2000

Для розрахунків загальної вартості, нам необхідно визначити витрати на розробку (Ct):

$$Ct = (Tf \times Sf) + (Tb \times Sb) + (Td \times Sd) + (Tt \times St), \quad (3.3)$$

де Ct – витрати на розробку.

$$Ct = (200 \times 30) + (200 \times 30) + (100 \times 25) + (100 \times 20), \quad (3.4)$$

де Ct – результат витрат на розробку.

Загальна вартість розробки складає $16500 або приблизно 668,000 грн (при курсі 1 USD = 40,48 грн).

Для підтримки роботи вебсайту необхідно мати надійну інфраструктуру. Це включає в себе витрати на хостинг, доменне ім'я та використання послуг Cloudflare:

– Реєстрація доменного імені: $15/рік
– VPS: $20/місяць
– Cloudflare: $20/місяць

Щомісячні витрати складають $40 або 1620 грн. Річні витрати на інфраструктуру складають $495 або 19,428 грн. Для розрахунку щомісячних витрат на інфраструктуру (Ci):

$$Ci = Vv + Vc, \quad (3.5)$$

де Ci – щомісячні витрати на інфраструктуру.

$$Ci = 20 + 20, \quad (3.6)$$

де Ci – результат трат щомісяця на інфраструктуру.

Для розрахунку річних витрат на інфраструктуру (Cri):

$$Cri = (Ci \times 12) + Vd, \quad (3.5)$$

де Cri – річні трати на інфраструктуру.

$$Cri = (40 \times 12) + 15, \quad (3.6)$$

де Vv – результат трат щорічних на інфраструктуру.

Після впровадження вебсайту необхідно забезпечувати постійну підтримку та обслуговування. Це включає оновлення ПЗ, виправлення помилок, та додавання нових функцій. Витрати на підтримку становлять $50/місяць або 2000 грн/місяць, що дорівнює $600 або 24,200 грн на рік.

Вартість впровадження вебсайту RentifyUA включає як початкові інвестиції в розробку та налаштування, так і регулярні витрати на підтримку інфраструктури та обслуговування. Загальні річні витрати складають $17685 або 715822 грн, включаючи реєстрацію доменного імені, хостинг, використання Cloudflare, оплату праці розробників, тестування, та підтримку.

ВИСНОВКИ

В результаті проведеної роботи було розроблено та впроваджено кросплатформний вебсайт для оренди житла RentifyUA. Процес розробки включав декілька ключових етапів: аналіз існуючих рішень на ринку, проєктування інтерфейсу користувача, реалізацію функціональності на основі сучасних технологій, таких як Vite та React, а також розгортання вебсайту на VPS.

На першому етапі було проведено аналіз ринку та визначено основні вимоги до вебсайту. В якості основи для проєктування було обрано існуючий популярний сервіс Airbnb, який став прикладом для реалізації зручного та функціонального інтерфейсу користувача. Основна концепція полягала у створенні платформи, яка б забезпечувала користувачам можливість швидко і зручно знаходити та орендувати житло, а також надавала б орендодавцям інструменти для ефективного управління своїми оголошеннями.

На етапі проєктування інтерфейсу користувача було розроблено прототипи та макети, враховуючи принципи UX/UI дизайну. Це дозволило забезпечити високу зручність використання сайту та доступність для широкого кола користувачів. Використання Vite та React забезпечило швидку та ефективну розробку клієнтської частини додатку, що дозволило зосередитися на реалізації функціональності та дизайну.

Розробка бекенд-частини проєкту включала впровадження архітектури Onion, що забезпечило чітке розмежування бізнес-логіки, доступу до даних та інфраструктури. Це дозволило створити гнучкий та масштабований додаток, який легко підтримувати та розширювати. Використання Node.js та Express забезпечило надійний та продуктивний сервер для обробки запитів користувачів.

Розгортання вебсайту на VPS включало налаштування серверного середовища, встановлення необхідного програмного забезпечення (Nginx, Node.js) та завантаження файлів додатку. Використання Cloudflare забезпечило додатковий захист та оптимізацію продуктивності вебсайту. В результаті, було

створено стабільне та безпечне середовище для функціонування платформи RentifyUA.

Економічна оцінка впровадження проєкту показала, що загальні витрати включають як початкові інвестиції в розробку та налаштування, так і регулярні витрати на підтримку інфраструктури та обслуговування. Остаточна вартість впровадження вебсайту RentifyUA складає $17695 або 715822 грн. Загальні річні витрати є виправданими, враховуючи потенційні прибутки від надання послуг оренди житлових приміщень через платформу RentifyUA.

Висновки, отримані в ході роботи, свідчать про те, що розробка кросплатформного вебсайту для оренди житла з використанням сучасних технологій є доцільною та ефективною. Впровадження RentifyUA дозволить покращити процес оренди житла, забезпечуючи зручність для користувачів та орендодавців, а також сприяти розвитку ринку оренди нерухомості.