Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Веб-сайт для бронювання готельних номерів

(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІ-21-4
Євгеній БЕЗРОДНИЙ
(власне ім'я, прізвище)

Спеціальність
123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп'ютерна інженерія
(повна назва освітньої програми)

Керівник: ст. викл. Роман ЯРОШЕВИЧ
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп’ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Безродному Євгенію Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Веб-сайт для бронювання готельних номерів

затверджена наказом по університету від “ 26 ” травня 2025 р. № 424 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 17 червня 2025 р.

3. Вхідні дані до роботи 1) документація мови програмування JavaScript
2) документація по HTML та CSS
3) документація по фреймворку React
4) редактор коду: Visual Studio Code
5) перелік використаних програмних засобів: браузери Google Chrome та Mozilla Firefox
6) операційна система: Linux Mint

4. Перелік питань, що потрібно опрацювати у роботі
1) аналіз проблеми та огляд існуючих рішень
2) вибір мови програмування для розробки
3) вибір бібліотек та фреймворків
4) розробка макету сайту
5) написання коду
6) виправлення помилок
7) висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій Слайд-презентація – 11 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Аналіз проблеми та огляд існуючих рішень	27.05.25-28.05.25	
2	Вибір мови програмування для розробки	29.05.25-30.05.25	
3	Вибір бібліотек та фреймворків	31.05.25-01.06.25	
4	Розробка макету сайту	02.06.25-03.06.25	
5	Написання коду	04.06.25-07.06.25	
6	Виправлення помилок	08.06.25-09.06.25	
7	Оформлення матеріалів кваліфікаційної роботи	10.06.25-11.06.25	
8	Подання кваліфікаційної роботи керівникові та	12.06.25-13.06.25	
	її попередній захист		
9	Подання кваліфікаційної роботи на рецензу-	14.06.25-16.06.25	
	вання		

Дата видачі завдання " 26 " травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ст. викл. Роман ЯРОШЕВИЧ
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 73 с., 21 рис., 2 дод., 7 джерел.

БРОНЮВАННЯ, ВЕБ-САЙТ, JAVASCRIPT, NODE.JS, EXPRESS, REACT, БАЗИ ДАНИХ, ІНТЕРФЕЙС, ГОТЕЛІ, КІМНАТИ, КОРИСТУВАЧ.

Метою кваліфікаційної роботи є розробка сайту для бронювання готельних номерів з доступним та зрозумілим для користувача інтерфейсом з використанням сучасних технологій для розробки веб-сайтів.

У ході виконання кваліфікаційної роботи було розроблено функціональний сайт для бронювання готельних номерів з використанням мови програмування JavaScript. Серверна частина сайту була виконана за допомогою Node.js та Express, інтерфейс користувача – за допомогою React а саме React Redux. Для збереження даних у хмарі використовується сервіс Google Firebase. Для різних функцій використовувалися бібліотеки, доступні завдяки використанню платформи Node.js, такі як dotEnv, JSONWebToken, Helmet та ін. Веб-сайт надає користувачеві такі можливості як: реєстрація та авторизація користувача, перегляд та бронювання номерів, можливість залишати та переглядати відгуки.

ABSTRACT

Bachelor’s thesis: 73 pages, 21 figures, 2 appendices, 7 sources.

BOOKING, WEB-SITE, JAVASCRIPT, NODE.JS, EXPRESS, REACT, DATABASES, INTERFACE, HOTELS, ROOMS, USERS

The major goal of this thesis is to develop a hotel room booking website with an accessible and user-friendly interface using modern web development technologies.

In order to complete the qualification project, a functional hotel booking website was developed using the JavaScript programming language. The server-side of the application was implemented using Node.js and Express, while the user interface was built with React, specifically React Redux. For cloud data storage, the Google Firebase service was used. Various functions were implemented using libraries available through the Node.js platform, such as dotEnv, JSONWebToken, Helmet, among others. The website provides users with features such as user registration and authentication, viewing and booking rooms, and the ability to leave and read reviews.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

API – інтерфейс програмування застосунку (англ., Application Programming Interface)

HTML – мова розмітки гіпертексту (англ., HyperText Markup Language)

CSS – каскадна таблиця стилів (англ., Cascading Style Sheets)

HTTP – протокол передачі даних (англ., HyperText Transfer Protocol)

SDK – комплект для розробки програмного забезпечення (англ., Software Development Kit)

NPM – менеджер пакетів (англ., Node Package Manager)

ВСТУП

У сучасному світі інформаційні технології стрімко проникають у всі сфери діяльності людини, і туристична галузь не є винятком. З розвитком інтернету все більше користувачів надають перевагу онлайн-сервісам, зокрема й для планування свої подорожей. Бо це набагато зручніше, швидше та надійніше: людині не потрібно витрачати свій час на те, щоб їздити містом, обирати номер серед десятків готелів на конкретну дату за прийнятною ціною, ще й коли на вулиці мороз чи спека. Все це можна зробити за 10-15 хвилин всього-на-всього використовуючи свій телефон чи ноутбук.

Наявність ефективного, зручного та функціонального веб-сайту з бронювання готелів є однією з основних умов успішного ведення готельного бізнесу. Завдяки автоматизації процесів користувачі можуть самостійно переглядати доступні варіанти номерів, перевіряти їх наявність у реальному часі, здійснювати оплату, отримувати підтвердження бронювання, а адміністратори – керувати заявками та проводити модерацію, прибираючі підозрілі пропозиції зі сторінок своїх сайтів .

Але в наших сучасних реаліях, нажаль для громадян України, бронювання готельних номерів набуло трошки нового сенсу. Це стосується людей, що евакуюються з територій бойових дій, і дуже часто знаходять собі тимчасовий прихисток у готелях чи мотелях, вирішуючи на місці куди їхати далі або шукаючи квартири на довгострокову перспективу. Це, звісно, дорожче ніж безкоштовне житло у спортзалах, школах та дитсадках, але й комфорту набагато більше і це було нормальною практикою у перші дні війни, коли люди не знали що робити.

Метою даної дипломної роботи є розробка веб-сайту, який дозволить автоматизувати процес бронювання номерів у готелі. Основними завданнями є створення зручного інтерфейсу для користувача, реалізація особистого кабінету адміністратора для керування номерами й бронюваннями.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

Ринок онлайн-бронювання житла в Україні представлений рядом як міжнародних, так і локальних сайтів. Але при цьому можна сказати, що він не є переповненим, та має явних фаворитів через не дуже велике різноманіття. Аналіз сильних та слабких сторін конкурентів є ключовим для позиціонування власного продукту та виявлення потенційних переваг. Було взято найбільші онлайн-платформи, які з’являються у пошуковій системі Google за запитом «забронювати готель».

1.1 Аналіз платформи Booking.com

Уже довгий час фаворитом на міжнародному ринку(в тому числі і в Україні) є сайт Booking.com, і це не дивно – на ньому можна знайти дуже широкий спектр житла, як бюджетного, так і преміального по всіх містах України. Цей сайт є одним з перших сайтів для букінгу, заснований ще у 1996 році. За цей час він заслужив довіру користувачів по всьому світу. Тому користуватися сайтом можна майже у будь-якій країні світу.

Booking має зрозумілий та інтуітивний інтерфейс (рисунок 1.1), готелі можна обирати одразу на мапі з ціною за вказаний термін проживання, тому обрати найкращий номер на потрібні дати взагалі не проблема. І не лише готелі: недовгострокова оренда квартир та кімнат також доступна. Також на сайті діють періодичні знижки і навіть є постійні акції (наприклад, знижка - 10% на перше бронювання новим зареєстрованим користувачам). Але й не без недоліків: власники готелі платять дуже високу комісію за кожне бронювання, а деякі оселі мають «накручені» оцінки і вводять в оману звичайних людей. Також є проблеми з підтримкою користувачів у критичних ситуаціях через деякі пункти в користувацькій угоді, яку не всі читають.

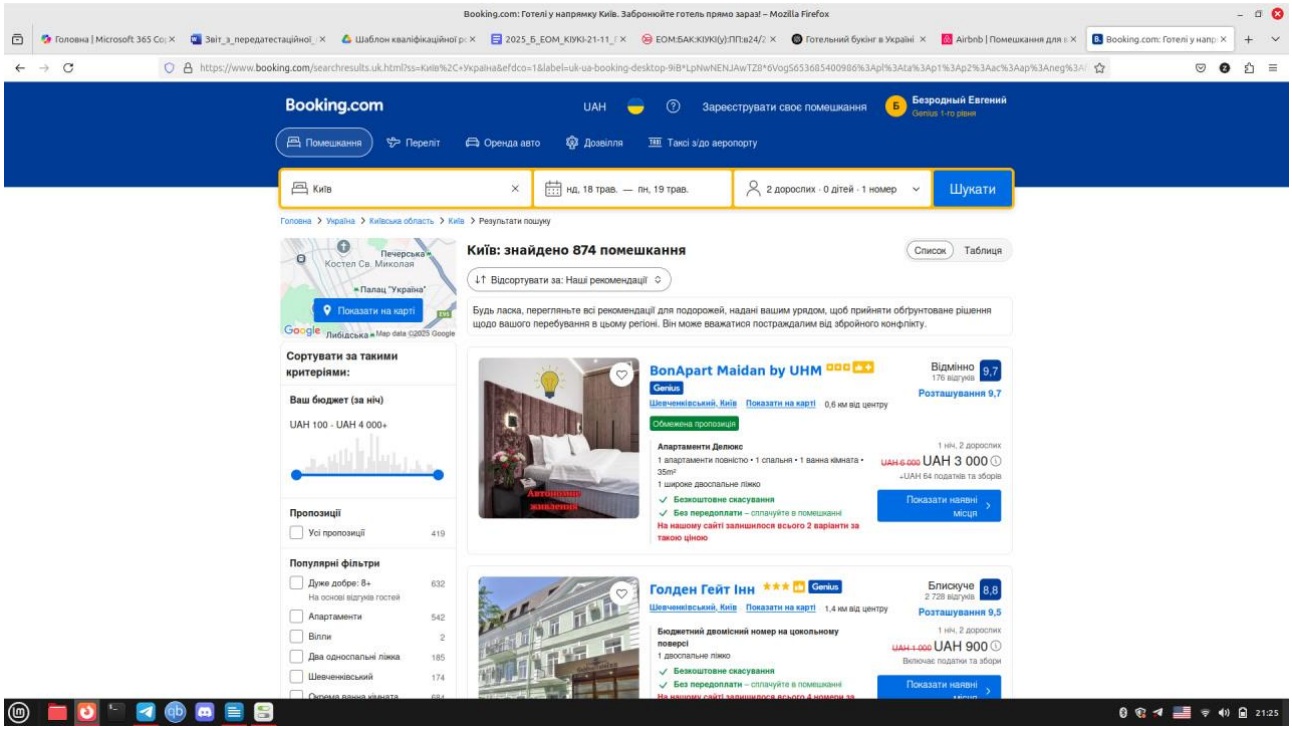

Рисунок 1.1 – Головна сторінка сайту Booking.com

1.2 Аналіз платформи AirBnB

Airbnb – це глобальна онлайн-платформа для оренди приватного житла. Вона дозволяє людям здавати власне житло (або його частину) туристам і мандрівникам на короткий або тривалий термін. В Україні сервіс активно розвивався до початку повномасштабної війни, а зараз має деякі обмеження.

На відміну від Booking, AirBnB має присутність не в усіх регіонах України: тільки у Києві, Одесі, Львові та Карпатах. Але й тут є перевага в тому, що можна найти унікальні пропозиції житла, відсутні у конкурентів. Сайт має трохи застарілий, але зручний інтерфейс (рисунок 1.2) та надає клієнтам захист у разі критичних ситуацій: на відміну від Booking механізми повернення коштів та скарг для гостей набагато легші та доступні.

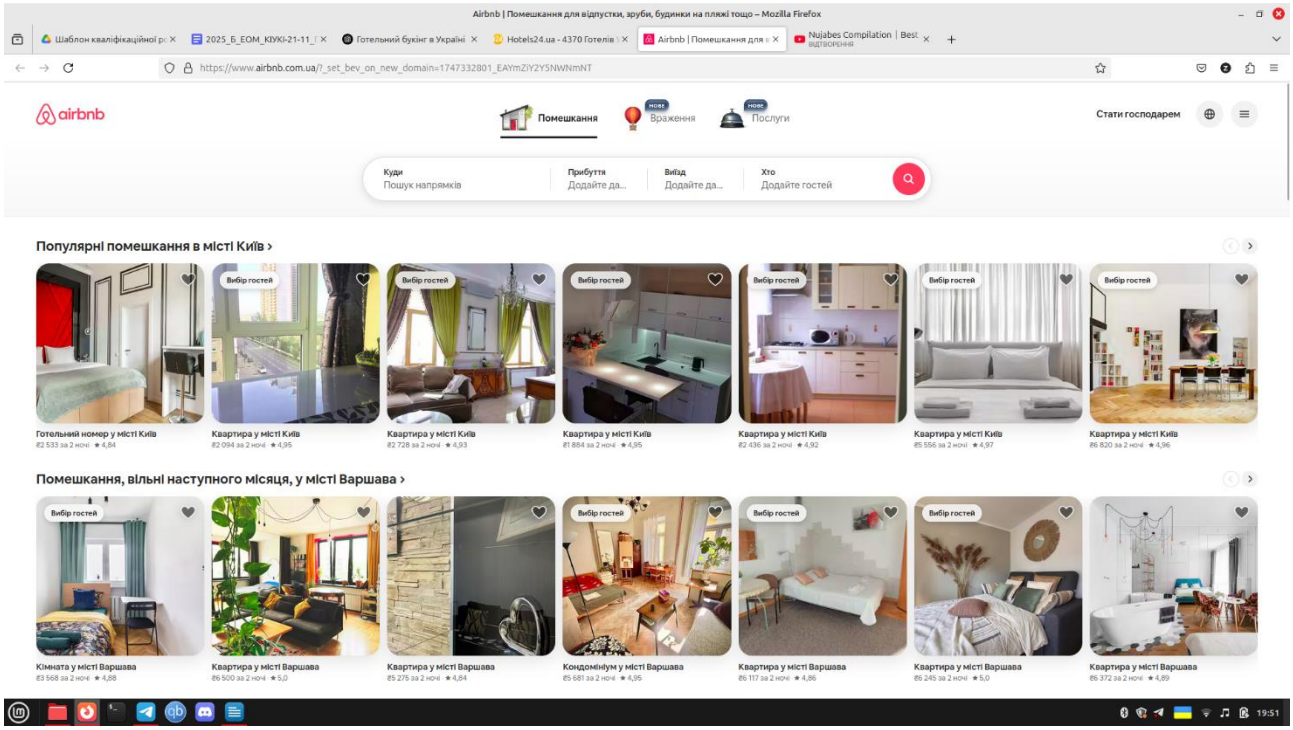

Рисунок 1.2 – Головна сторінка сайту AirBnB

1.3 Аналіз Hotels24.ua

Hotels24.ua – це національна платформа онлайн-бронювання готелів, апартаментів, баз відпочинку та інших форм розміщення по всій території України. Сайт орієнтований переважно на внутрішній ринок і пропонує простий та доступний інтерфейс для українських туристів. Платформа є одним із лідерів у сегменті внутрішнього туризму в Україні.

Хоча сайт і є суто локальним, він має деякі переваги над своїми конкурентами. Платформа має глибоке покриття саме українських об’єктів, включаючи невеликі готелі, приватні садиби, санаторії, бази відпочинку та хостели, які рідко представлені на міжнародних майданчиках. У багатьох випадках бронювання можливе без внесення авансу – оплата здійснюється при поселенні. Це підвищує довіру користувачів, особливо старшого віку. Також Hotels24 часто проводить сезонні кампанії, надає купони, знижки для постійних користувачів та спеціальні пропозиції.

Звісно і не без недоліків. Головним є відсутність мобільного застосунку, лише мобільну версію сайту. Інтерфейс сайту досить функціональний, але не

сучасний за міжнародними стандартами (рисунок 1.3). Відсутні деякі звичні користувачу інструменти (наприклад, гнучкі дати, інтерактивна мапа як основа пошуку). Частина готелів має погано структуровані описи або неактуальні фото.

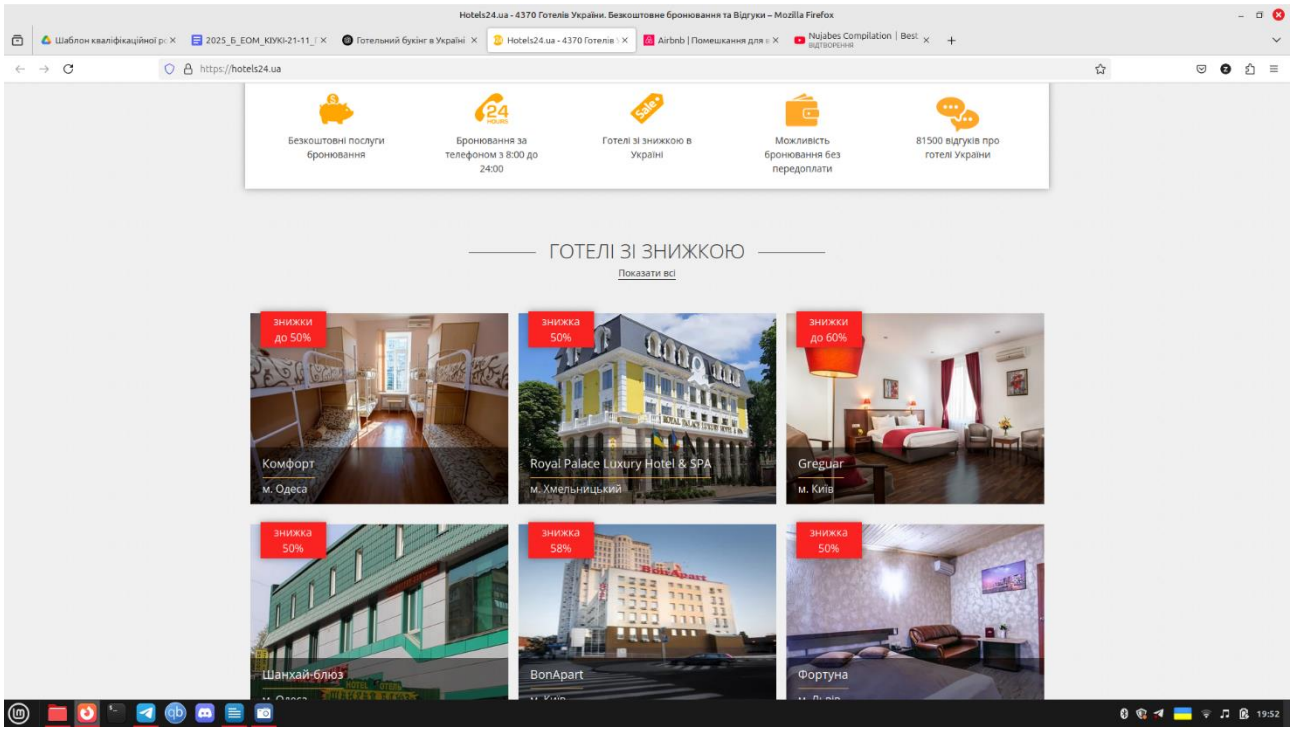

Рисунок 1.3 – Головна сторінка сайту Hotels24.ua

1.4 Постановка завдання

Метою роботи є розробка веб-сайту для бронювання готельних номерів, який дозволить користувачам зручно здійснювати пошук, перегляд та бронювання доступних номерів, а адміністрації – ефективно управляти даними про номери, бронювання та клієнтів.

Проаналізувавши найбільші платформи на ринку сайтів для бронювання готельних номерів, було виконано план по розробці найкращого рішення та виділено головний функціонал, який потрібно реалізувати в першу чергу:

- для найкращого користувацького досвіду сайт повинен мати сучасний та інтуїтивний інтерфейс, в якому людина не заплутається;
- веб-сайт обов’язково повинен мати методи аутентифікації

користувачів: реєстрацію, авторизацію, завершення сесії, збереження даних у локальному сховищі браузера для повторної авторизації;

- особистий кабінет користувача, в якому він зможе побачити що і на який день він забронював;
- зручний список доступних номерів, перегляд інформації про них: ціни, опису, доступних дат та відгуків;
- бронювання номерів за датами, можливість зміни дати бронювання у разі виникнення непередбачених обставин чи помилок, неможливість бронювання одних і тих самих номерів декількома користувачами в однакові календарні дати;
- чітке розділення ролей користувачів, ручна модерація сайту адміністрацією для усунення зловмисників;
- усі дані про номери, бронювання, користувачів повинні зберігатися у хмарній базі даних з безперервним доступом до неї 24 години на добу.

2 АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ

Кваліфікаційну роботу виконано з використанням таких засобів реалізації:

- середовище розробки Visual Studio Code;
- мова програмування JavaScript;
- Node.js – для побудови швидкої серверної частини та зрозумілого API;
- фреймворк Express.js, з використанням різних бібліотек (наприклад, JSONWebToken та DotEnv);
- React Redux – для виконання клієнтської частини;
- мови розмітки HTML та CSS;
- платформа Google Firebase для хмарного середовища.

2.1 Вибір мови програмування

Для виконання даного веб-застосунку було обрано мову програмування JavaScript. Вона є однією з найпопулярніших та найуніверсальніших мов у сучасній веб-розробці. Однією з головних переваг JavaScript є можливість працювати в єдиному середовищі, без використання інших мов для програмування. Велика кількість бібліотек та фреймворків дає великий простір для реалізації цікавих функцій та рішень.

2.1.1 Основні відомості про JavaScript

JavaScript – це високорівнева, динамічна мова програмування, яка спочатку була створена для додавання інтерактивності до вебсторінок [1]. З часом JavaScript еволюціонував у повноцінну універсальну мову, яка використовується як для фронтенду (інтерфейс користувача), так і для бекенду

(серверна логіка).

Сучасний JavaScript підтримується всіма популярними веббраузерами, а також активно використовується поза межами браузера – завдяки таким середовищам, як Node.js, які дозволяють запускати JavaScript на сервері. Це відкриває можливість створення повноцінних застосунків (у тому числі мобільних та десктопних) лише за допомогою однієї мови.

JavaScript має велику спільноту розробників і багатий набір інструментів: фреймворки (React, Angular, Vue), бібліотеки (jQuery, D3.js), утиліти для тестування, розгортання, обробки даних тощо. Це дозволяє значно пришвидшити розробку, зменшити кількість помилок і забезпечити масштабованість застосунку.

2.1.2 Основні відомості про Node.js

Node.js – це середовище виконання JavaScript, яке дозволяє запускати код поза межами веб-браузера. Воно побудоване на рушії V8 від Google Chrome та призначене для створення швидких, масштабованих мережевих застосунків. На відміну від традиційного підходу, коли JavaScript працює лише на стороні клієнта, Node.js дозволяє використовувати цю мову також і на сервері [2]. Платформа перетворила JavaScript на мову загального використання з великою спільнотою розробників.

Однією з головних особливостей Node.js є асинхронна та подійно-орієнтована модель програмування. Замість того, щоб блокувати виконання коду під час очікування операцій введення-виведення, Node.js використовує неблокувальні операції, дозволяючи додаткам одночасно обробляти безліч запитів. Це робить Node.js дуже ефективним для створення масштабованих та чуйних серверних додатків.

Якщо говорити про переваги Node.js, то варто згадати його екосистему модулів, яка представлена вбудованим менеджером пакетів npm. Npm – це один з найбільших репозиторіїв програмного забезпечення, де розробники

можуть знайти та використовувати тисячі готових модулів та бібліотек для прискорення процесу розробки.

Проте Node.js не обмежується виключно розробкою веб-серверів. На node js можна написати різноманітні додатки, включно з мережевими серверами, мікросервісами, інструментами командного рядка й навіть настільними додатками. Завдяки його гнучкості та потужним можливостям, Node.js став однією з найпопулярніших платформ для розробки.

2.2 Середовище розробки

Visual Studio Code – це потужний текстовий редактор з відкритим вихідним кодом, розроблений компанією Microsoft, який став одним з найпопулярніших інструментів серед розробників у всьому світі. Він поєднує у собі простоту та легкість звичайного текстового редактора з розширеними функціями, які традиційно асоціюються з повноцінними інтегрованими середовищами розробки (IDE). Завдяки своїй гнучкості, швидкодії та великій кількості розширень, VS Code став універсальним інструментом для роботи з різними мовами програмування, фреймворками та технологіями.

Однією з ключових переваг Visual Studio Code є його крос-платформенність – він доступний для операційних систем Windows, macOS та Linux, що робить його зручним вибором для розробників, які працюють у різних середовищах. Архітектура редактора побудована на основі Electron, що дозволяє йому бути легким та швидким, незважаючи на багатий функціонал.

Важливою особливістю VS Code є його розширюваність. Завдяки великій бібліотеці розширень, які доступні через вбудований маркетплейс, розробники можуть налаштувати редактор під свої конкретні потреби. Наприклад, існують розширення для підтримки різних мов програмування (Python, Java, C++ тощо), інтеграції з системами контролю версій (Git), засобами автоматизації (Docker, Kubernetes) та іншими інструментами.

Вбудований термінал у VS Code дозволяє виконувати команди прямо в

редакторі, що значно підвищує продуктивність роботи. Крім того, редактор має потужні засоби для налагодження коду, включаючи точки зупину, покрокове виконання та інспектування змінних.

Visual Studio Code також відзначається високим рівнем інтеграції з хмарними сервісами, такими як GitHub, Azure та AWS, що робить його ідеальним вибором для сучасних розробників, які працюють у хмарних середовищах. Підтримка Remote Development дозволяє працювати з кодом, який знаходиться на віддалених серверах або в контейнерах, без необхідності його локального копіювання.

Зручний інтерфейс, можливість кастомізації (теми, шрифти, розкладка вікон) та регулярні оновлення від Microsoft роблять Visual Studio Code одним з найкращих інструментів для розробників у 2023 році. Він підходить як для початківців, так і для досвідчених фахівців, пропонуючи потужний, але водночас простий у використанні інструмент для щоденної роботи.

Таким чином, Visual Studio Code – це не просто текстовий редактор, а сучасне багатофункціональне середовище для розробки, яке поєднує у собі легкість, продуктивність та широкі можливості для кастомізації. Його популярність серед розробників обумовлена зручністю використання, активною спільнотою та постійним вдосконаленням функціоналу, що робить його основним інструментом у багатьох IT-проектах.

2.3 Опис застосунків для створення серверної частини

2.3.1 Express.js

Express.js – це мінімалістичний та гнучкий веб-фреймворк для Node.js, який надає простий у використанні набір інструментів для створення серверних додатків та API [3]. Він є одним із найпопулярніших фреймворків у екосистемі Node.js завдяки своїй простоті, швидкості та широким можливостям для розширення через проміжне програмне забезпечення

(middleware). Express.js дозволяє розробникам створювати як прості односторінкові сайти, так і складні багаторівневі веб-додатки з маршрутизацією, обробкою запитів та інтеграцією з базами даних.

Однією з ключових переваг Express.js є його модульність. Завдяки middleware-функціям, розробники можуть легко додавати різноманітну логіку обробки запитів, наприклад, аутентифікацію, логування, стиснення даних або парсинг тіла запиту. Middleware в Express.js працює за принципом ланцюжка обробників, де кожен проміжний обробник може модифікувати запит або відповідь перед передачею управління наступній функції. Це дозволяє створювати гнучкі та масштабовані архітектури.

Маршрутизація в Express.js реалізована інтуїтивно зрозумілим способом, що дає змогу визначати URL-шляхи та методи HTTP (GET, POST, PUT, DELETE тощо) для обробки різних типів запитів [4]. Додатково, Express.js підтримує динамічні маршрути за допомогою параметрів URL, що дозволяє обробляти змінні частини, такі як ідентифікатори користувачів.

Ще однією сильною стороною Express.js є його сумісність з численними сторонніми бібліотеками та плагінами, такими як Passport.js для аутентифікації, Mongoose для роботи з MongoDB, або Helmet для забезпечення безпеки. Це робить його ідеальним вибором для розробників, які хочуть швидко створювати продуктивні додатки без необхідності реалізовувати базову функціональність з нуля.

Незважаючи на свою простоту, Express.js залишається потужним інструментом, який підходить як для невеликих проектів, так і для великих корпоративних рішень. Він активно підтримується спільнотою, має велику кількість документації та прикладів, що значно прискорює процес навчання та розробки. Завдяки своїй продуктивності, Express.js продовжує залишатися стандартом де-факто для створення серверних додатків на Node.js.

2.3.2 JSONWebToken

JSONWebToken (JWT) – це відкритий стандарт (RFC 7519) для безпечного передавання даних між сторонами у вигляді компактного та самодостатнього об’єкта JSON. Він широко використовується в сучасній веб-розробці для аутентифікації та обміну інформацією, оскільки дозволяє легко перевіряти цілісність даних за допомогою цифрових підписів або шифрування. У контексті Node.js, JWT часто застосовується для створення токенів доступу, які клієнт (наприклад, веб-додаток або мобільний застосунок) може використовувати для ідентифікації користувача після успішної авторизації.

Основна перевага JWT полягає в тому, що він не вимагає зберігання стану на сервері, оскільки вся необхідна інформація міститься в самому токені. Це робить його ідеальним рішенням для розподілених систем та мікросервісних архітектур, де традиційні сесії на основі cookies можуть бути менш ефективними. Токен складається з трьох частин: заголовка (header), корисного навантаження (payload) та підпису (signature). Заголовок містить інформацію про алгоритм шифрування, payload – дані користувача (наприклад, ідентифікатор або ролі), а signature забезпечує захист від підробки шляхом перевірки цілісності токена.

У Node.js для роботи з JWT найчастіше використовується бібліотека jsonwebtoken, яка надає простий API для створення, підпису та верифікації токенів. Наприклад, після успішної аутентифікації сервер може згенерувати JWT, підписати його секретним ключем або приватним ключем RSA, і відправити клієнту. Клієнт зберігає цей токен (зазвичай у localStorage або cookies) і надсилає його з кожним наступним запитом до сервера у заголовку Authorization. Сервер, у свою чергу, перевіряє підпис токена та його валідність перед тим, як надати доступ до захищених ресурсів.

Однією з ключових особливостей JWT є можливість використання як симетричного (HS256), так і асиметричного (RS256) шифрування. У першому випадку для підпису та перевірки використовується один і той же секретний

ключ, що підходить для внутрішніх систем. У другому – застосовуються відкритий та приватний ключі, що дозволяє розподіляти перевірку токенів між різними сервісами без необхідності передавати секретні дані.

Незважаючи на свою популярність, JWT має певні недоліки. Наприклад, якщо токен буде викрадено (наприклад, через XSS-атаку), зловмисник зможе використовувати його до закінчення терміну дії, оскільки сервер не може відкликати окремі токени без додаткових механізмів (наприклад, чорних списків). Також, якщо в payload міститься занадто багато даних, розмір токена може значно зростати, що впливає на продуктивність при кожному запиті.

Тим не менш, JSON Web Token залишається одним із найпоширеніших рішень для аутентифікації в сучасних веб-додатках завдяки своїй простоті, масштабованості та підтримці в більшості мов програмування. У поєднанні з Node.js та фреймворками на кшталт Express.js, він дозволяє швидко реалізовувати безпечні механізми авторизації, що робить його незамінним інструментом у арсеналі розробника.

2.3.3 DotEnv

Dotenv – це невелика, але надзвичайно корисна бібліотека для Node.js, яка автоматизує процес роботи з конфігураційними змінними. Вона дозволяє розробникам зберігати чутливі дані, такі як API-ключі, паролі до баз даних, налаштування з’єднань і інші параметри, окремо від основного коду. Це не тільки підвищує безпеку, але й спрощує процес розробки, оскільки дозволяє легко змінювати налаштування без необхідності редагування вихідного коду.

Принцип роботи dotenv дуже простий: під час запуску додатку модуль зчитує файл .env (який зазвичай знаходиться в корені проекту) і додає всі визначені в ньому змінні до глобального об’єкта process.env. Наприклад, якщо у файлі .env міститься рядок DB_PASSWORD=secure123, то після виклику require('dotenv').config() ця змінна стане доступною у коді як process.env.DB_PASSWORD.

Одна з ключових переваг dotenv – це підтримка різних середовищ. Розробники можуть створювати окремі файли, такі як .env.development, .env.test або .env.production, і завантажувати їх залежно від поточного середовища. Це особливо корисно в CI/CD-пайплайнах, де конфігурація може змінюватися автоматично під час розгортання. Крім того, .env файли зазвичай додаються до .gitignore, щоб уникнути випадкового витоку конфіденційних даних у публічні репозиторії.

Незважаючи на свою простоту, dotenv інтегрується з більшістю сучасних фреймворків і бібліотек Node.js, таких як Express.js, NestJS, Next.js тощо. Він також добре працює у поєднанні з іншими інструментами, наприклад, Docker, де змінні середовища можуть передаватися через аргументи контейнера або окремі файли конфігурації.

Однак варто пам’ятати, що dotenv не є єдиним способом керування конфігурацією. У складних системах можуть використовуватися більш просунуті рішення, такі як AWS Secrets Manager, HashiCorp Vault або Kubernetes ConfigMaps, які забезпечують централізоване керування секретами. Але для невеликих та середніх проектів dotenv залишається оптимальним вибором завдяки своїй простоті та ефективності.

Таким чином, модуль dotenv є важливим інструментом в арсеналі Node.js-розробника, який допомагає забезпечити безпеку, гнучкість і зручність роботи з конфігурацією додатків. Його використання дозволяє уникнути жорсткого кодування конфіденційних даних, спрощує розгортання та підтримку проекту на різних етапах життєвого циклу розробки.

2.3.4 Firebase

Firebase – це комплексна хмарна платформа від Google, яка надає розробникам набір інструментів для швидкого створення, масштабування та підтримки веб- і мобільних додатків. Вона поєднує в собі можливості backend-розробки, аналітики, автентифікації, зберігання даних та інших критично

важливих функцій, дозволяючи командам зосередитися на створенні якісного користувацького досвіду без необхідності налаштування інфраструктури.

Однією з ключових переваг Firebase є те, що це база даних реального часу (Realtime Database), яка автоматично синхронізує дані між усіма клієнтами та зберігає їх у хмарі. Це дозволяє створювати інтерактивні додатки, де зміни відображаються миттєво без необхідності оновлювати сторінку. Згодом Firebase також представив Firestore – більш масштабовану та гнучку NoSQL-базу даних з підтримкою складних запитів та офлайн-доступу.

Для автентифікації користувачів Firebase пропонує Firebase Authentication, який підтримує різні методи входу, такі як електронна пошта, соціальні мережі (Google, Facebook, Twitter) та навіть телефонні номери. Це значно спрощує процес додавання системи входу в додаток, оскільки Firebase бере на себе всю логіку перевірки, безпеки та управління сесіями.

Ще одна потужна функція Firebase – хмарні функції (Cloud Functions), які дозволяють виконувати серверний код у відповідь на події в додатку, такі як зміни в базі даних, HTTP-запити або системні сповіщення. Це дозволяє створювати складну бізнес-логіку без необхідності розгортати власний сервер.

Firebase також включає інструменти для моніторингу продуктивності (Performance Monitoring), аналітики (Analytics) та тестування (Test Lab), що допомагає розробникам оптимізувати свої додатки та покращувати користувацький досвід. Крім того, Firebase Hosting забезпечує швидке і безпечне розгортання статичних веб-сайтів із підтримкою HTTPS, CDN та автоматичним масштабуванням.

Таким чином, Firebase є ідеальним рішенням для стартапів, невеликих команд і навіть великих компаній, які хочуть швидко запускати якісні додатки з мінімальними витратами на інфраструктуру. Його модульність, інтеграція з іншими сервісами Google та простота використання роблять його одним із найпопулярніших інструментів у сучасній веб- і мобільній розробці.

2.3.5 Helmet

Helmet – це middleware-технологія (програмний шар, що забезпечує взаємодію та інтеграцію між різними компонентами великих програмних систем) для Node.js та Express, яка підвищує безпеку застосунку від різних веб-вразливостей шляхом заміни HTTP-заголовків.

Без Helmet заголовки, які за замовчуванням повертає Express, розкривають конфіденційну інформацію та роблять ваш Node.js-додаток вразливим до зловмисників. Натомість використання Helmet у Node.js захищає ваш застосунок від XSS-атак, уразливостей, пов'язаних із політикою безпеки контенту (CSP), та інших проблем із безпекою.

2.3.6 Nodemon

Nodemon – це зручний інструмент для автоматичного перезапуску серверного додатку під час розробки. Його основне завдання полягає в тому, щоб позбавити розробника необхідності щоразу вручну перезапускати сервер після змін у коді. Працюючи як обгортка над звичайною командою запуску Node.js, Nodemon відстежує зміну файлів у проєкті й одразу після їх збереження ініціює повторне виконання програми. Це суттєво пришвидшує процес тестування і налагодження коду, оскільки дозволяє миттєво бачити результати змін без зайвих дій. Встановлення утиліти є простим: її можна додати як залежність у проєкт через npm, після чого запуск серверного файлу здійснюється не командою node, а nodemon, що забезпечує автоматизацію та зручність у щоденній роботі. Завдяки такому підходу робота над серверною частиною сайту стає не лише продуктивнішою, а й приємнішою.

2.3.7 Multer

Multer – це спеціальна бібліотека для Node.js, яка дозволяє обробляти файли, надіслані з форм на сайті. Браузери, коли користувач завантажує зображення чи інші документи через форму, надсилають такі дані у форматі multipart/form-data. Multer допомагає серверу правильно прийняти ці дані, розпізнати їх як файли, зберегти їх у відповідному місці на диску та передати далі для обробки. Найчастіше Multer використовується з Express – популярним фреймворком для побудови серверної частини вебдодатків. За допомогою Multer можна налаштувати, у яку папку зберігати файли, як їх перейменовувати, які типи дозволені (наприклад, тільки зображення), і що робити у випадку помилок. Після обробки Multer додає інформацію про файл до об’єкта запиту, що дозволяє розробнику легко отримати доступ до нього. У підсумку Multer значно спрощує реалізацію функціоналу завантаження файлів, беручи на себе всі технічні деталі цього процесу.

2.4 React та Redux Toolkit для створення інтерфейсу користувача

React – це сучасна JavaScript-бібліотека для створення інтерактивних користувацьких інтерфейсів, яка кардинально змінила підхід до веб-розробки. Її головною особливістю є компонентний підхід, який дозволяє розбивати складний інтерфейс на незалежні частини [5]. У контексті роботи з HTML і CSS, React пропонує свої унікальні підходи та рішення.

В React традиційний HTML трансформується у JSX - спеціальний синтаксичний розширення JavaScript, яке дозволяє писати HTML-подібний код прямо в JavaScript-файлах. JSX поєднує логіку та розмітку в одному місці, що робить код більш зрозумілим і підтримуваним. Наприклад, замість розділення HTML і JavaScript у різних файлах, розробник може описати структуру компонента, його стилі та поведінку в єдиному логічному блоці.

Робота з CSS у React має кілька підходів. Класичний метод підключення

зовнішніх CSS-файлів залишається можливим, але більш сучасні методи включають CSS-in-JS (наприклад, styled-components або emotion) , які дозволять описувати стилі прямо в JavaScript-коді. Такі бібліотеки генерують унікальні імена класів, вирішуючи проблеми з областю видимості стилів, і надають можливість динамічно змінювати стилі на основі стану компонента.

Окремо варто відзначити модульні CSS (CSS Modules), який автоматично створює унікальні імена класів, дозволяючи використовувати звичайний CSS-синтаксис без ризику конфліктів імен. Це рішення особливо популярне в великих проектах, де важлива підтримка та масштабованість.

React також впроваджує концепцію "вбудованих стилів" (inline styles), де стилі передаються як JavaScript-об'єкти. Хоча цей підхід має свої обмеження (наприклад, неможливість використовувати псевдокласи чи медіа-запити без додаткових бібліотек), він добре підходить для динамічних стилів, які залежать від стану компонента.

Важливою перевагою роботи з HTML і CSS у React є можливість створення динамічних інтерфейсів. Компоненти можуть змінювати стилі в залежності від стану додатку. Це дозволяє створювати складні інтерактивні інтерфейси, які важко було б реалізувати за допомогою традиційного підходу з окремими HTML і CSS файлами.

Redux Toolkit – це офіційна бібліотека, яка спрощує використання Redux у розробці додатків на React [6]. Вона була створена для того, щоб подолати головні недоліки класичного Redux – надмірну кількість шаблонного коду, складність налаштування та розробки, а також загальну заплутаність.

Перевагою Redux Toolkit є вбудована підтримка асинхронних операцій [7]. Вона дозволяє легко працювати з API-запитами, автоматично обробляючи стани завантаження, успіху та помилки, без потреби вручну створювати окремі дії для кожного етапу запиту.

Крім того, Redux Toolkit включає функцію configureStore, яка не тільки полегшує створення сховища, а й автоматично підключає корисні інструменти, як-от Redux DevTools та middleware для перевірки помилок.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Загальна структура веб-сайту

Веб-сайт реалізовано у вигляді двох основних частин: клієнтської (frontend) та серверної (backend), кожна з яких організована у відповідній директорії – front та back.

Серверна частина розташована в директорії back (рисунок 3.1), де ключовим є файл server.js, який ініціалізує сервер на платформі Node.js із використанням фреймворку Express.

Всередині папки src розміщено основну логіку серверної частини. Тут міститься папка controllers, у якій зберігаються контролери для різних частин системи: обробка бронювань, взаємодія з номерами та управління користувачами. Папка routes містить відповідні маршрути до кожного контролера, які формують API-сервіс. Папка middleware реалізує: обробку помилок, авторизацію, завантаження файлів. В config зберігаються налаштування бази даних та константи доступу. Додатково в проекті присутня папка uploads, яка використовується для збереження зображень.

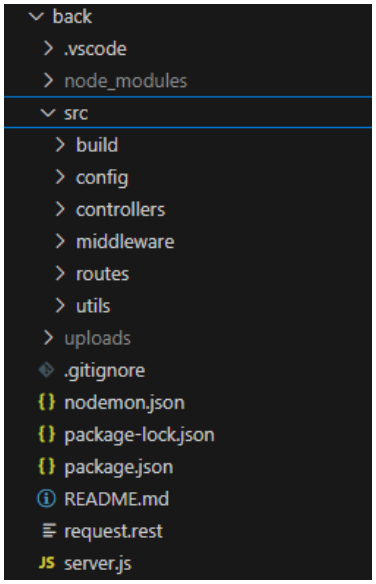

Рисунок 3.1 – Структура директорії back

Всю логіку та інтерфейс користувача за допомогою бібліотеки React реалізовано в директорії front (рисунок 3.2), папці src. На верхньому рівні містяться головні файли: App.js відповідає за маршрути між сторінками та загальну структуру інтерфейсу, а index.js є точкою входу, звідки застосунок запускається в браузері. Для оформлення стилю використовується файл app.styles.scss, а також базові стилі в index.css.

У папці app знаходиться файл store.js, який відповідає за збереження глобального стану програми з використанням Redux. Логіка, пов’язана з окремими частинами функціоналу, така як аутентифікація користувачів, бронювання чи управління кімнатами, розташовується у папці features.

Графічний інтерфейс користувача побудований з компонентів, розміщених у папці component. Тут є компоненти для заголовка, списку кімнат та списку бронювань. Кожен компонент має свій окремий файл зі стилями.

Усі сторінки сайту згруповані в директорії Pages. Серед них є головна сторінка, сторінка входу, реєстрації, перегляду кімнат, оформлення бронювання, панель адміністратора. Окремо винесено налаштування для підключення до Firebase – вони зберігаються у файлі fb.js, розташованому в папці config. А для допоміжних функцій існує папка helper

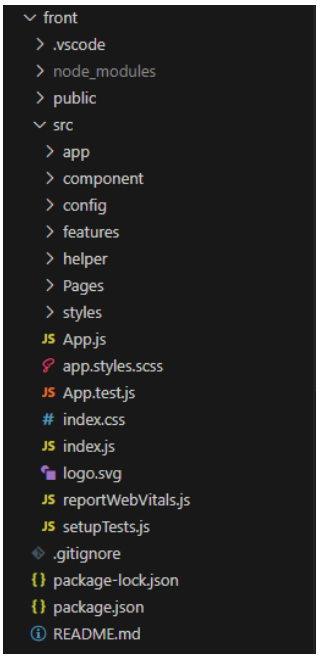

Рисунок 3.2 – Структура директорії front

3.2 Опис функціоналу

В цьому розділі вибірково будуть описані функції, методи та алгоритми що використовуються як в серверній частині, так і в клієнтській, та складають функціонал сайту, такий як реєстрація, логін та вихід користувача та інше.

3.2.1 Файл authMiddleware.js

Файл authMiddleware.js відповідає за перевірку автентичності користувачів, що надсилають запити до серверної частини системи. Його основна функція – перевірити, чи користувач надав правильний токен доступу, і якщо так – дозволити йому виконати запит.

У ньому міститься дві основні функції: authMiddleware (лістинг 3.1) та adminOnlyMiddleware (лістинг 3.2). Функція authMiddleware перевіряє користувача за допомогою токена у форматі Bearer. При кожному запиті до захищеного ресурсу сервер намагається зчитати заголовок Authorization, де очікується наявність токена у форматі Bearer. Якщо цей заголовок відсутній або не відповідає очікуваному формату, сервер повертає статус 401 з повідомленням про відсутність авторизації. У випадку, коли токен присутній, система передає його у функцію перевірки verifyIdToken, яка зазвичай реалізується за допомогою Firebase або іншого постачальника аутентифікації. Якщо токен є дійсним, інформація про користувача додається до об’єкта запиту, що дозволяє мати доступ до даних автентифікованого користувача.

У разі виникнення помилки під час перевірки, система повертає відповідний статус та повідомлення залежно від типу помилки: це може бути, наприклад, закінчення терміну дії токена, його неправильна структура або заблокований акаунт користувача. При цьому передбачена обробка винятків для середовища розробки, коли сервер може включити додаткову інформацію про помилку у відповідь. Також реалізована перевірка на випадок, якщо відповідь вже була частково надіслана – у такому разі система лише фіксує

проблему у логах, не намагаючись повторно відправити дані клієнту.

Лістинг 3.1 – Функція authMiddleware

```
const authMiddleware = async (req, res, next) => {
  try {
    // Extract token from header
    const authHeader = req.headers.authorization;
    if (!authHeader || !authHeader.startsWith('Bearer ')) {
      return res.status(401).json({
        message: 'Unauthorized - No token provided'
      });
    }

    const idToken = authHeader.split(' ')[1].trim();

    const decodedToken = await auth.verifyIdToken(idToken);
    req.user = decodedToken;

    next();
  } catch (error) {
    console.error('Authentication error:', error.message);

    let statusCode = 401;
    let message = 'Unauthorized - Invalid token';

    if (error.code === 'auth/id-token-expired') {
      message = 'Token expired';
    } else if (error.code === 'auth/argument-error') {
      message = 'Malformed token';
    } else if (error.code === 'auth/user-disabled') {
      message = 'User account disabled';
      statusCode = 403;
    }

    if (!res.headersSent) {
      return res.status(statusCode).json({
        message,
        error: process.env.NODE_ENV === 'development' ?
error.message : undefined
      });
    }

    console.warn('Authentication failed after headers were
sent');
  }
};
```

Функція adminOnlyMiddleware виконує роль додаткового рівня захисту в серверному додатку, надаючи доступ до певних маршрутів лише для

користувачів із роллю адміністратора. Вона працює як middleware у середовищі Express і використовується після проходження користувачем первинної аутентифікації, яка зберігає інформацію про нього в об'єкті req.user.

Спочатку функція перевіряє, чи взагалі аутентифікований користувач і чи містить об’єкт req.user унікальний ідентифікатор користувача. Якщо ні – повертається відповідь зі статусом 401, що означає відсутність авторизації. Далі система звертається до бази даних і намагається знайти документ користувача за цим ідентифікатором. Якщо користувача не знайдено, повертається помилка зі статусом 404.

Якщо користувач існує, з його документа у базі витягується об’єкт з даними, зокрема – його роль. У випадку, коли роль користувача не є «admin», сервер повертає статус 403 із повідомленням про заборону доступу, а також вказує, яку роль повинен мати користувач і яку він має насправді.

У разі, якщо роль є коректною, повні дані користувача (як з токена, так і з бази) зберігаються в req.userData для подальшого використання наступними функціями або контролерами. Завершальним кроком є виклик функції next(), яка передає управління наступному обробнику.

Якщо в процесі перевірки виходить помилка, наприклад, через перевищення ліміту запитів або відсутність прав доступу до Firebase, система обробляє її та повертає відповідне повідомлення і статус.

Лістинг 3.2 – Функція adminOnlyMiddleware

```
const adminOnlyMiddleware = async (req, res, next) => {
  try {
    if (!req.user || !req.user.uid) {
      return res.status(401).json({ message: 'Unauthorized -
User not authenticated' });
    }

    const userDoc = await
db.collection('users').doc(req.user.uid).get();

    if (!userDoc.exists) {
      return res.status(404).json({ message: 'User not found'
});
    }
```

```
    const userData = userDoc.data();

    if (userData.role !== 'admin') {
      return res.status(403).json({
        message: 'Forbidden - Admin privileges required',
        requiredRole: 'admin',
        yourRole: userData.role || 'user'
      });
    }

    req.userData = {
      ...req.user,
      ...userData
    };

    next();
  } catch (error) {
    console.error('Admin check error:', error.message);

    let status = 500;
    let message = 'Internal server error';

    if (error.code === 'resource-exhausted') {
      status = 429;
      message = 'Too many requests';
    } else if (error.code === 'permission-denied') {
      status = 403;
      message = 'Firestore permission denied';
    }

    if (!res.headersSent) {
      return res.status(status).json({
        message,
        error: process.env.NODE_ENV === 'development' ?
error.message : undefined
      });
    }
  }
};
```

3.2.2 Файл uploadMiddleware.js

У цьому файлі виконується збереження файлів у локальну директорію uploads (лістинг 3.3). А якщо її не існує, то вона створюється автоматично у папці з проєктом. Імена файлів формуються динамічно, з урахуванням часу завантаження та випадкового суфікса, щоб уникнути конфліктів і перезапису

вже наявних файлів. Дозволяється завантажувати лише зображення формату JPEG, PNG або WebP. Якщо файл має інший формат, користувач побачить помилку із відповідним повідомленням.

Лістинг 3.3 – Функція uploadMiddleware

```
const storage = multer.diskStorage({
    destination: (req, file, cb) => {
        const uploadDir = 'uploads/';
        if (!fs.existsSync(uploadDir)) fs.mkdirSync(uploadDir, {
recursive: true });
        cb(null, uploadDir);
    },
    filename: (req, file, cb) => {
        const uniqueSuffix = Date.now() + '-' +
Math.round(Math.random() * 1e9);
        const ext = path.extname(file.originalname);
        cb(null, `room-${uniqueSuffix}${ext}`);
    }
});

const fileFilter = (req, file, cb) => {
    const allowedTypes = ['image/jpeg', 'image/png',
'image/webp'];
    if (allowedTypes.includes(file.mimetype)) {
        cb(null, true);
    } else {
        cb(new AppError('Only JPEG, PNG, and WebP images are
allowed', 400), false);
    }
};

const upload = multer({
    storage,
    fileFilter,
    limits: {
        fileSize: 5 * 1024 * 1024 // 5MB
    }
});
```

3.2.3 Файл roomController.js

У цьому файлі знаходяться функції для керування готельними номерами на сервері: створення, видалення, отримання інформації тощо. Усього в модулі реалізовано п’ять основних функцій.

1. getRooms – функція, яка виконує запит до бази даних для отримання

всіх існуючих номерів (лістинг 3.4). Якщо колекція порожня, повертається порожній масив. У іншому випадку, дані з документів перетворюються у зручний формат, де кожен номер має свій унікальний id та відповідні поля (назва, ціна, опис тощо).

Лістинг 3.4 – Функція getRooms

```
const getRooms = async (req, res, next) => {
  try {
    const snapshot = await db.collection(roomsDoc).get();

    if (snapshot.empty) {
      return res.status(200).json([]);
    }

    const rooms = snapshot.docs.map(doc => ({
      id: doc.id,
      ...doc.data()
    }));

    return res.status(200).json(rooms);
  } catch (error) {
    next(error);
  }
};
```

2. createRoom – функція для створення нового номера (лістинг 3.5). Вона очікує отримати в тілі запиту такі поля, як name (назва номера), price (ціна), desc (опис), roomNumbers (перелік номерів), а також img (посилання на зображення). У випадку відсутності обов'язкових полів (name, price, roomNumbers), сервер повертає помилку 400. При успішному створенні номеру в базі формується відповідь з усіма параметрами та ідентифікатором нового ресурсу.

Лістинг 3.5 – Функція createRoom

```
const createRoom = async (req, res, next) => {
  try {
    const { name, price, desc, roomNumbers, img } = req.body;

    if (!name || !price || !roomNumbers) {
      return res.status(400).json({ error: "Missing required
fields" });
```

```
    }

    const roomRef = await db.collection(roomsDoc).add({
      name,
      price,
      desc: desc || "",
      roomNumbers,
      imgUrl: img || "",
      createdAt: new Date(),
      updatedAt: new Date()
    });

    const newRoom = {
      id: roomRef.id,
      name,
      price,
      desc,
      roomNumbers,
      imgUrl: img
    };

    return res.status(201).json(newRoom);
  } catch (error) {
    next(error);
  }
};
```

3. getRoom – функція, яка дозволяє отримати інформацію про конкретний номер за його унікальним ідентифікатором (id) (лістинг 3.6). Якщо документ із таким id не існує, сервер повертає помилку 404.

Лістинг 3.6 – Функція getRoom

```
const getRoom = async (req, res, next) => {
  try {
    const roomId = req.params.id;
    const doc = await db.collection(roomsDoc).doc(roomId).get();

    if (!doc.exists) {
      return res.status(404).json({ error: "Room not found" });
    }

    const roomData = doc.data();
    const room = {
      id: doc.id,
      ...roomData
    };
```

```
    return res.status(200).json(room);
  } catch (error) {
    next(error);
  }
};
```

4. updateRoom – функція для оновлення даних про існуючий номер (лістинг 3.7). Вона дозволяє змінювати як обов’язкові, так і додаткові поля. Всі параметри є необов’язковими, тому оновлення відбувається лише для тих полів, які були передані в запиті. Перед оновленням здійснюється перевірка, чи існує документ у базі. У разі успіху повертається оновлений об’єкт.

Лістинг 3.7 – Функція updateRoom

```
const updateRoom = async (req, res, next) => {
  try {
    const roomId = req.params.id;
    const { name, price, desc, roomNumbers, img } = req.body;

    const roomRef = db.collection(roomsDoc).doc(roomId);
    const doc = await roomRef.get();

    if (!doc.exists) {
      return res.status(404).json({ error: "Room not found" });
    }

    const updateData = {
      updatedAt: new Date()
    };

    if (name) updateData.name = name;
    if (price) updateData.price = price;
    if (desc) updateData.desc = desc;
    if (roomNumbers) updateData.roomNumbers = roomNumbers;
    if (img) updateData.imgUrl = img;

    await roomRef.update(updateData);

    const updatedRoom = await roomRef.get();
    return res.status(200).json({
      id: updatedRoom.id,
      ...updatedRoom.data()
    });
  } catch (error) {
    next(error);
  }
};
```

5. deleteRoom – функція, яка реалізує видалення номера з бази даних (лістинг 3.8). Після перевірки наявності номера з вказаним id, документ видаляється, і користувач отримує підтвердження про успішне видалення.

Лістинг 3.8 – Функція deleteRoom

```
const deleteRoom = async (req, res, next) => {
  try {
    const roomId = req.params.id;
    const roomRef = db.collection(roomsDoc).doc(roomId);
    const doc = await roomRef.get();

    if (!doc.exists) {
      return res.status(404).json({ error: "Room not found" });
    }

    await roomRef.delete();
    return res.status(200).json({
      id: roomId,
      message: "Room deleted successfully"
    });
  } catch (error) {
    next(error);
  }
};
```

3.2.4 Файл authSlice.js

Файл authSlice.js знаходиться в директорії front, та відповідає за усі операції, пов’язані з даними та аутентифікацією користувачів у клієнтській частині: реєстрація, логін, вихід з системи та запис користувацьких даних до локального сховища браузера.

У файлі містяться функції для реєстрації нового користувача (registerUser), входу в систему (loginUser) і виходу (logoutUser). Під час успішної аутентифікації дані користувача зберігаються в localStorage, що дозволяє зберігати сесію навіть після оновлення сторінки.

1. Функція registerUser реалізує логіку асинхронної реєстрації нового користувача у клієнтській частині застосунку (лістинг 3.9). Вона виконує роль посередника між інтерфейсом користувача та серверною частиною,

забезпечуючи обробку запиту на реєстрацію та відповідної відповіді.

У тілі функції ініціюється HTTP-запит методом POST на маршрут http://localhost:5000/api/users/register, що передає у форматі JSON дані, введені користувачем. Далі запит надсилається на локальний сервер. Після отримання відповіді з сервера відбувається її декодування у формат JSON, а результат зберігається за допомогою виклику функції storeUser(data). Ця функція відповідає за збереження користувацьких даних у localStorage, що дозволяє користувачеві залишатися авторизованим без необхідності повторного входу.

У разі виникнення помилки під час виконання запиту, функція повертає помилку з текстовим повідомленням. Це дає змогу компонентам інтерфейсу отримати доступ до інформації про помилку та відобразити її користувачеві.

Лістинг 3.9 – Функція registerUser

```
const storeUser = (data) => localStorage.setItem("user",
JSON.stringify(data));
const user = JSON.parse(localStorage.getItem("user"));

export const registerUser = createAsyncThunk(
  "auth/register",
  async ({ email, password, name }, thunkApi) => {
    try {
      const res = await
fetch("http://localhost:5000/api/users/register", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({ email, password, name }),
      });

      const data = await res.json();

      storeUser(data);
      return data;
    } catch (error) {
      return thunkApi.rejectWithValue(error.message);
    }
  }
);
```

2. Функція loginUser відповідає за авторизацію користувача у

клієнтській частині застосунку (лістинг 3.10). В ній відбувається аутентифікація через базу даних із додатковою перевіркою на серверній стороні, що забезпечує правильну обробку входу користувача в систему.

На першому етапі здійснюється автентифікація через метод signInWithEmailAndPassword (один з методів, імпортованих з Firebase). Цей метод перевіряє коректність введених облікових даних – електронної адреси та пароля. У разі успішної перевірки застосунок отримує об’єкт userCredential, з якого за допомогою getIdToken витягується ID токен. Цей токен підтверджує особу користувача та є необхідним для здійснення захищених запитів.

Далі функція працює за тим же принципом, що і registerUser: HTTP-запит методом POST, але на маршрут http://localhost:5000/api/users/login, декодується у формат JSON, а користувацькі дані записуються до локального сховища. У разі виникнення помилок повертається функція з текстом опису помилки, щоб користувач знав, у чому проблема.

Лістинг 3.10 – Функція loginUser

```
export const loginUser = createAsyncThunk(
  "auth/login",
  async ({ email, password }, thunkApi) => {
    try {
      const userCredential = await
signInWithEmailAndPassword(auth, email, password);

      const idToken = await userCredential.user.getIdToken();

      const res = await
fetch("http://localhost:5000/api/users/login", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({ email, password }),
      });

      const data = await res.json();
      localStorage.setItem("user", JSON.stringify({ ...data,
idToken }));
      return data;
    } catch (error) {
      return thunkApi.rejectWithValue(error.message);
    }
```

```
  }
);
```

3. Функція logoutUser виконує вихід авторизованого користувача із системи у клієнтській частині застосунку, із завершенням сесії користувача та очисткою даних з локального сховища (лістинг 3.11).

Усередині функції викликається метод Firebase signOut, який забезпечує завершення сеансу користувача на стороні бази даних шляхом скасовування дійсності токенів, пов’язаних з поточним обліковим записом, і блокування можливості доступу до захищених ресурсів без повторної авторизації. Після цього викликається функція removeUser, яка відповідає за очищення localStorage. У результаті видаляються збережені раніше токени аутентифікації та інші користувацькі дані, що сприяє завершенню сесії. У разі успішного виконання обох операцій функція повертає логічне значення true, що сигналізує про завершення процесу виходу. Якщо ж під час виконання виникає помилка, вона передається користувачеві через метод rejectWithValue.

Лістинг 3.11 – Функція logoutUser

```
export const logoutUser = createAsyncThunk(
  "auth/logout",
  async (_, thunkApi) => {
    try {
      await signOut(auth);
      removeUser();
      return true;
    } catch (error) {
      return thunkApi.rejectWithValue(error.message);
    }
  }
);
```

4 ТЕСТУВАННЯ ВЕБ-САЙТУ

На головній сторінці сайту (рисунок 4.1) нас зустрічає кнопка-перехідник «Explore Rooms», яка перенаправляє на список доступних номерів, та хедер з логотипом (який при натисканні повертає на головну сторінку) та кнопками логіну, реєстрації та сторінки усіх доступних номерів. На даний момент сайт виконано з використанням англійської мови.

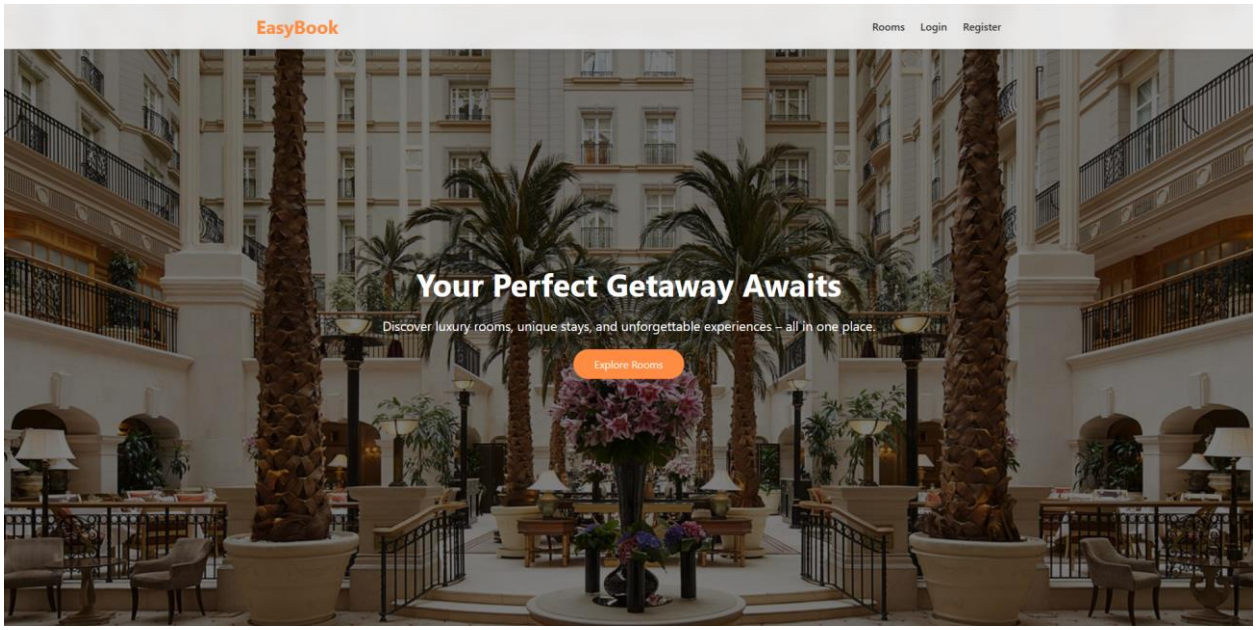

Рисунок 4.1 – Головна сторінка сайту

При натисканні кнопки «Register» (рисунок 4.2) користувача перенаправляє на сторінку з полем реєстрації, де йому потрібно ввести своє ім’я, електронну пошту та пароль. Н На даний момент, обмеження щодо паролю це введені мінімум 6 символів. У майбутньому це буде виправлено для більш надійної безпеки даних користувачів.

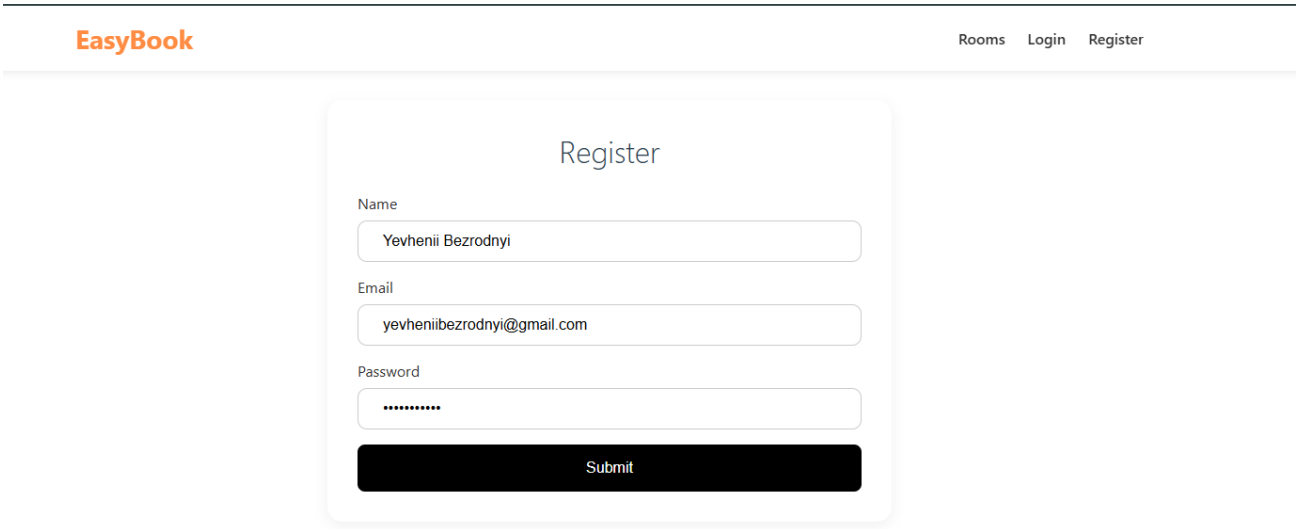

Рисунок 4.2 – Реєстрація користувача

Відкриємо адміністраторську панель нашої бази даних (рисунок 4.3) щоб перевірити, що дані було збережено коректно. Обравши таблицю «users» бачимо всіх користувачів, серед яких тільки що створений нами з даними, які було введено в поле реєстрації, унікальним ідентифікатором користувача та інформацією про дату реєстрації користувача.

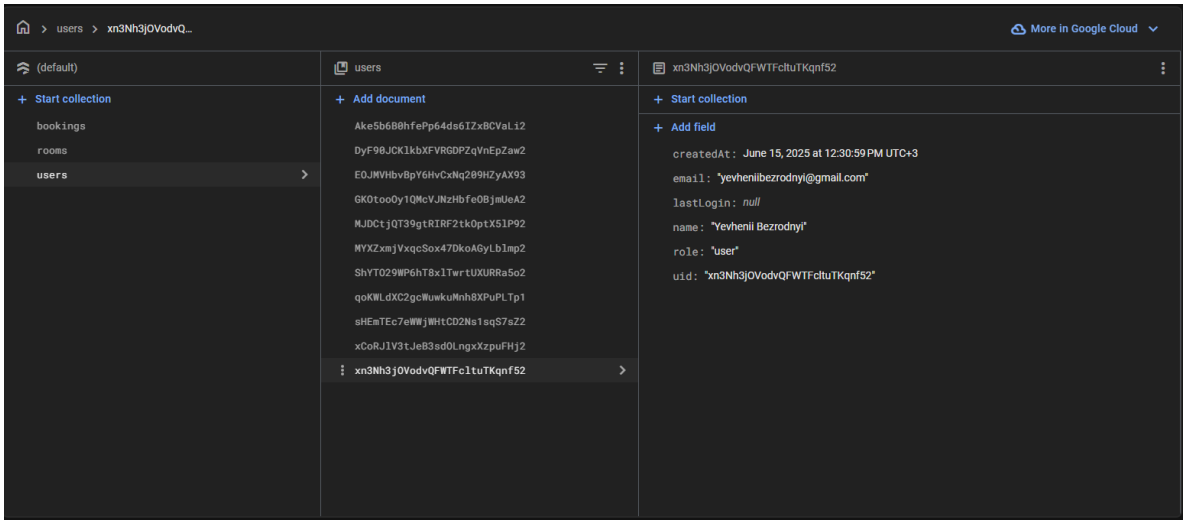

Рисунок 4.3 – Дані користувача у базі даних

На цій же сторінці можна вручну змінити дані в усіх доступних полях (рисунок 4.4), тож поміняємо роль користувача з «user» на «admin» для того, щоб після авторизації отримати більше дозволів для роботи з функціоналом.

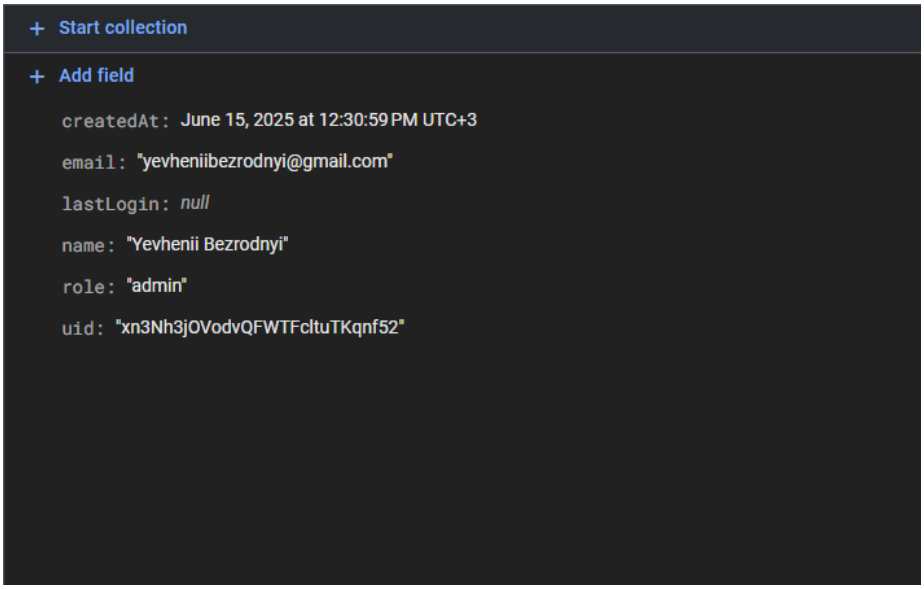

Рисунок 4.4 – Зміна ролі користувача у базі даних

Повернемося до інтерфейсу сайту. Після натискання кнопки «Submit» відкривається поле для авторизації користувача (рисунок 4.5). Вводимо сюди дані з поля реєстрації користувача.

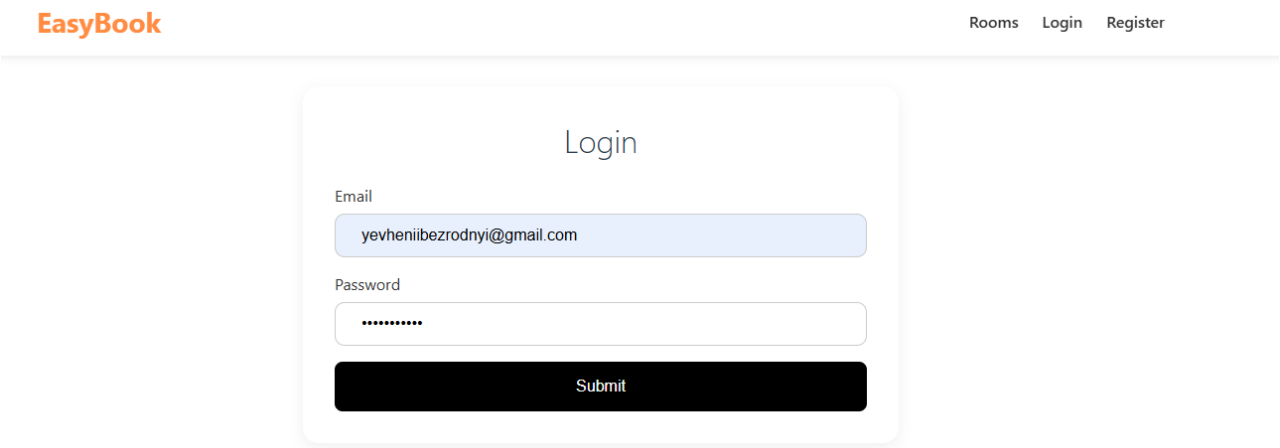

Рисунок 4.5 – Авторизація користувача

Після авторизації користувач (як admin, так і user) опиняється на сторінці «Dashboard», на якій можна побачити усі бронювання. Різниця буде в тому, що admin бачить бронювання усіх користувачів, а user – тільки свої власні. Хедер авторизованного користувача з правами адміністратора має кнопки «Rooms»,

«Dashboard» та «Create».

Продемонструємо функціонал створення кімнат та бронювань для них (саме для цього нашому користувачу і було надано права адміністратора). Перейдемо на сторінку «Create» (рисунок 4.6). Тут потрібно ввести ім’я, вартість бронювання у розрахунку на одну добу, опис, надати кімнаті номер та завантажити зображення. Заповнимо дані та створимо кімнату.

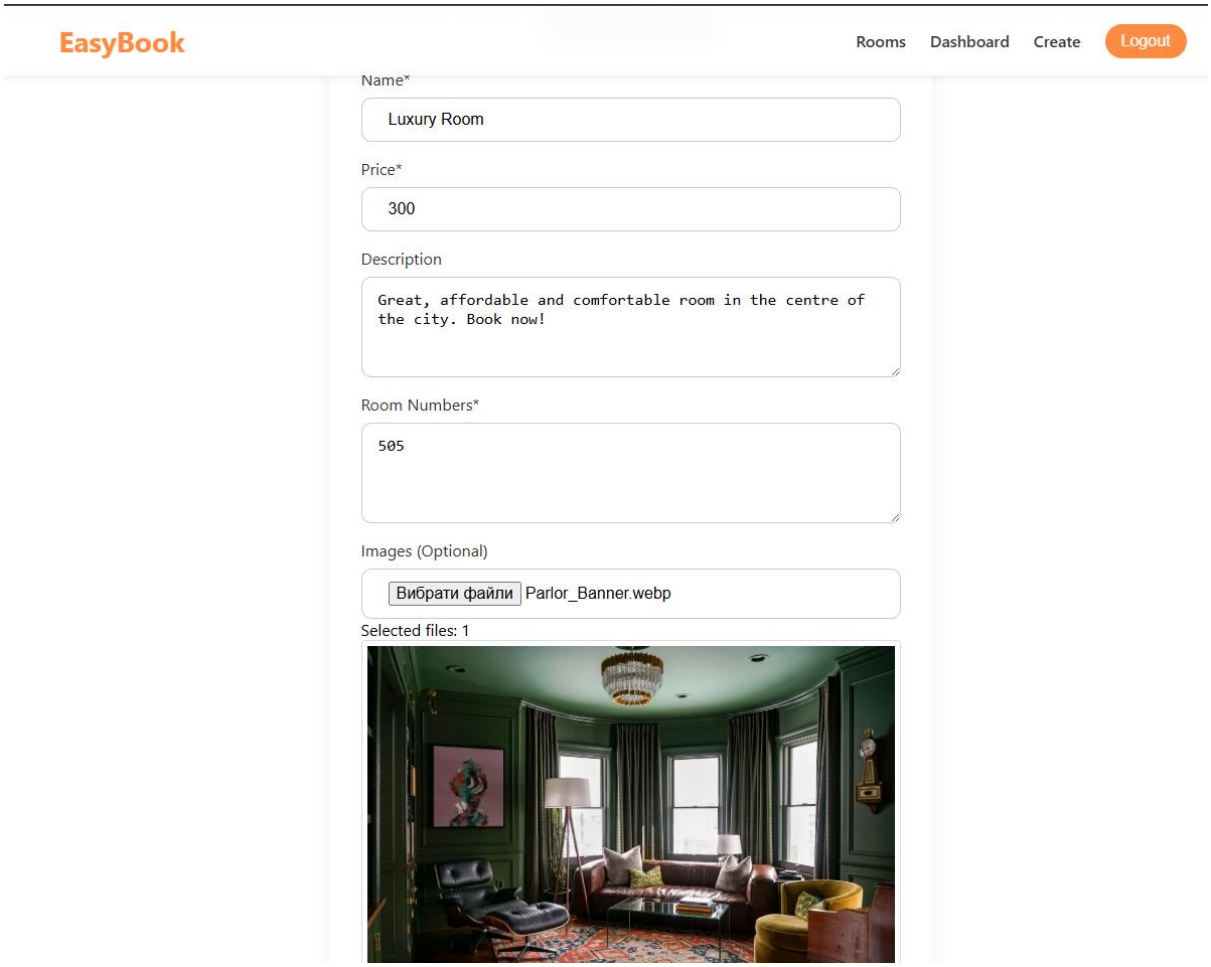

Рисунок 4.6 – Створення кімнати

Після створення, номер стає видимим та доступним на сторінці «Rooms», де можна переглянути його та всі інші доступні для бронювання кімнати. Тепер можна забронювати його, або інший, натиснувши на нього для переходу на сторінку з описом (рисунок 4.7).

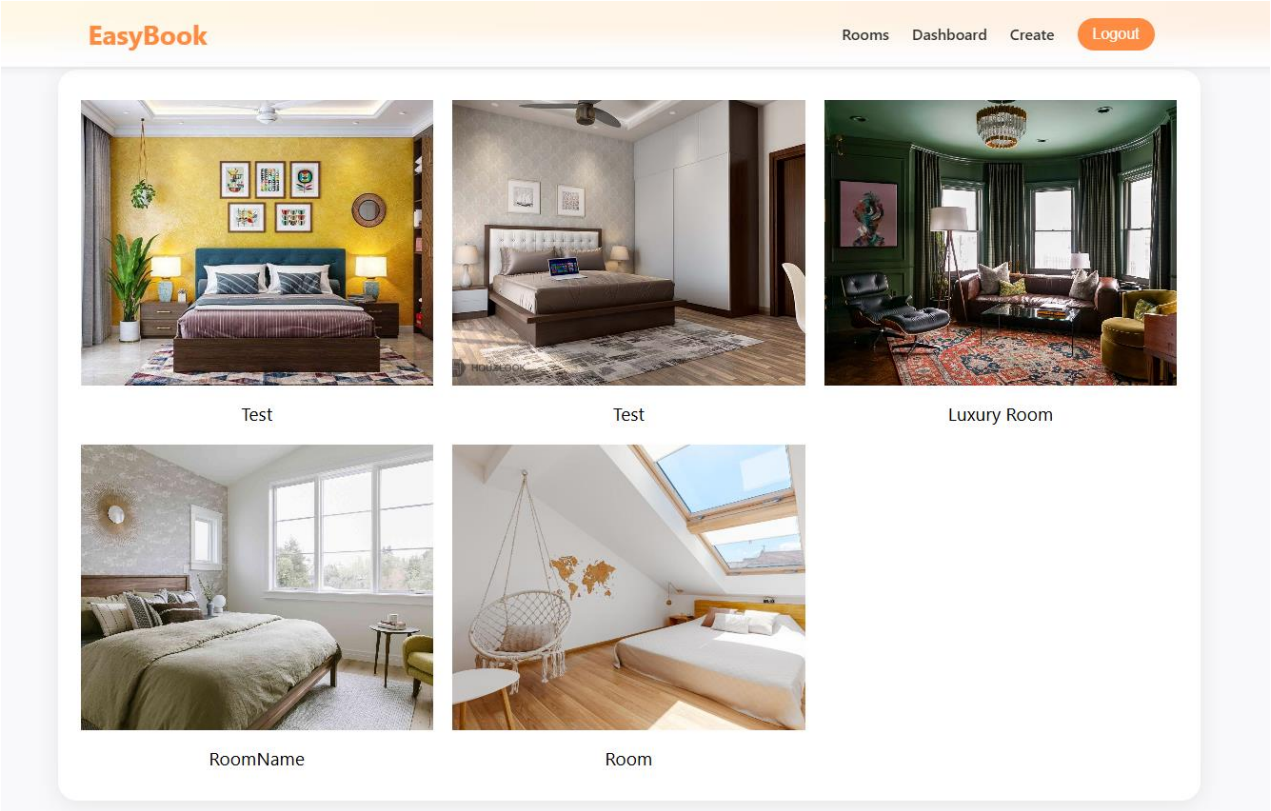

Рисунок 4.7 – Список доступних кімнат

Оберемо новостворений на сторінці «Rooms» готельний номер, щоб подивитись, як виглядає інтерфейс сторінки (рисунок 4.8). Користувач з роллю адміністратора може змінити, видалити номер та побачити відгуки користувачів. Також доступне поле для того, щоб залишити власний відгук, але здебільшого саме адміністратору це потрібно для комунікації з іншими користувачами.

Відредагуємо номер, змінивши його вартість та опис. Натиснувши «Edit Room» (рисунок 4.9) вводимо нові дані у поля, які хочемо змінити. Зображення змінити не можна, але цей функціонал буде додано в майбутньому. Застосуємо зміни натиснувши «Submit» та переглянемо результат (рисунок 4.10).

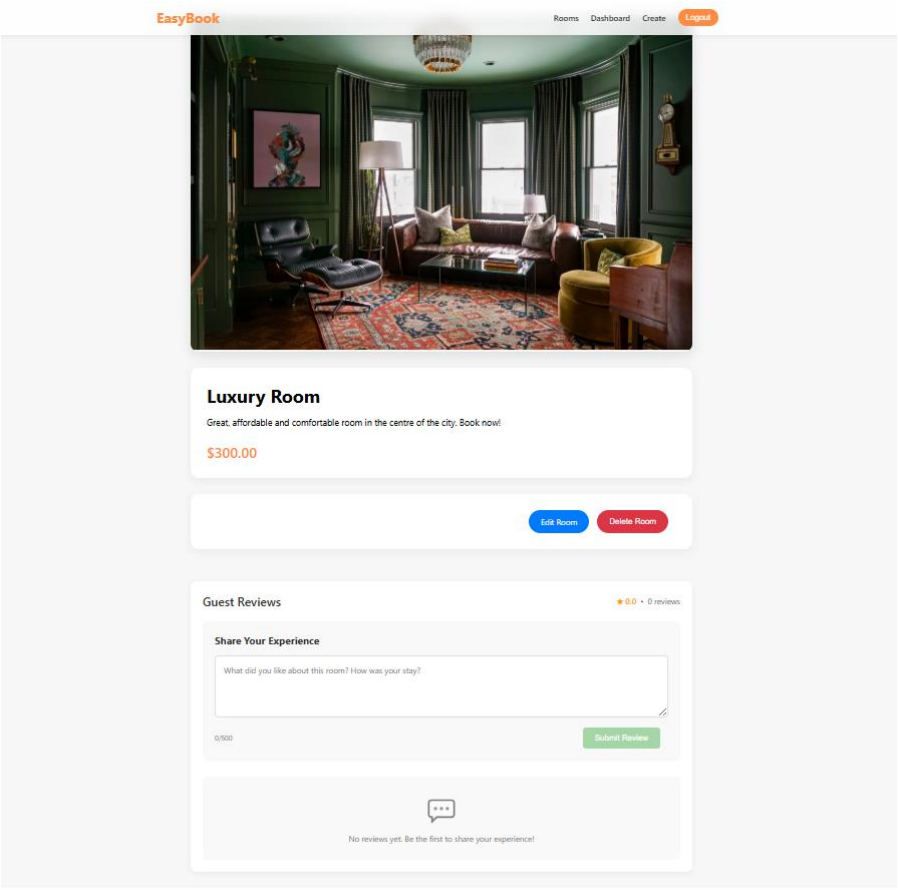

Рисунок 4.8 – Інтерфейс сторінки кімнати для адміністратора

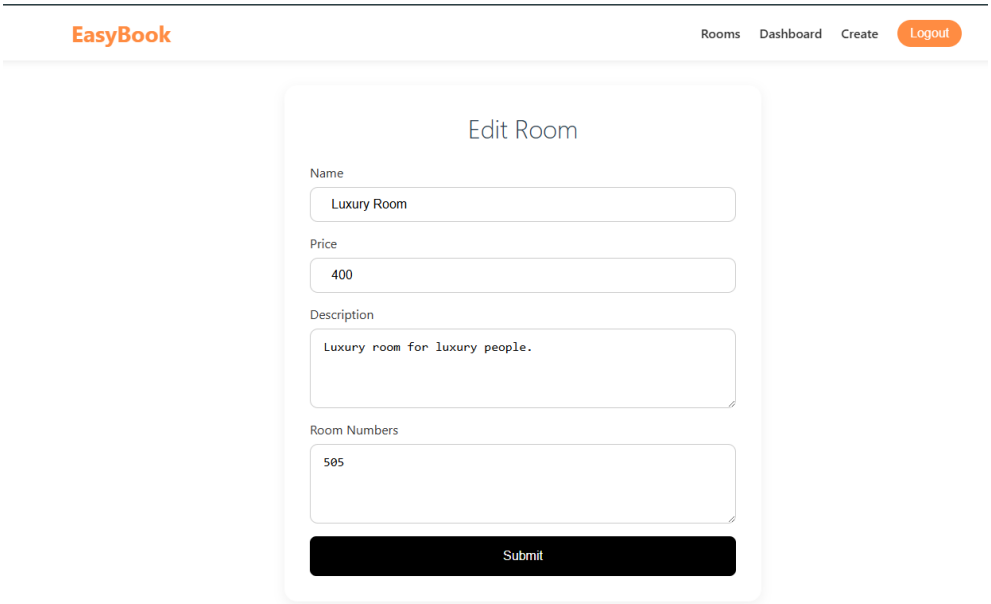

Рисунок 4.9 – Зміна даних кімнати

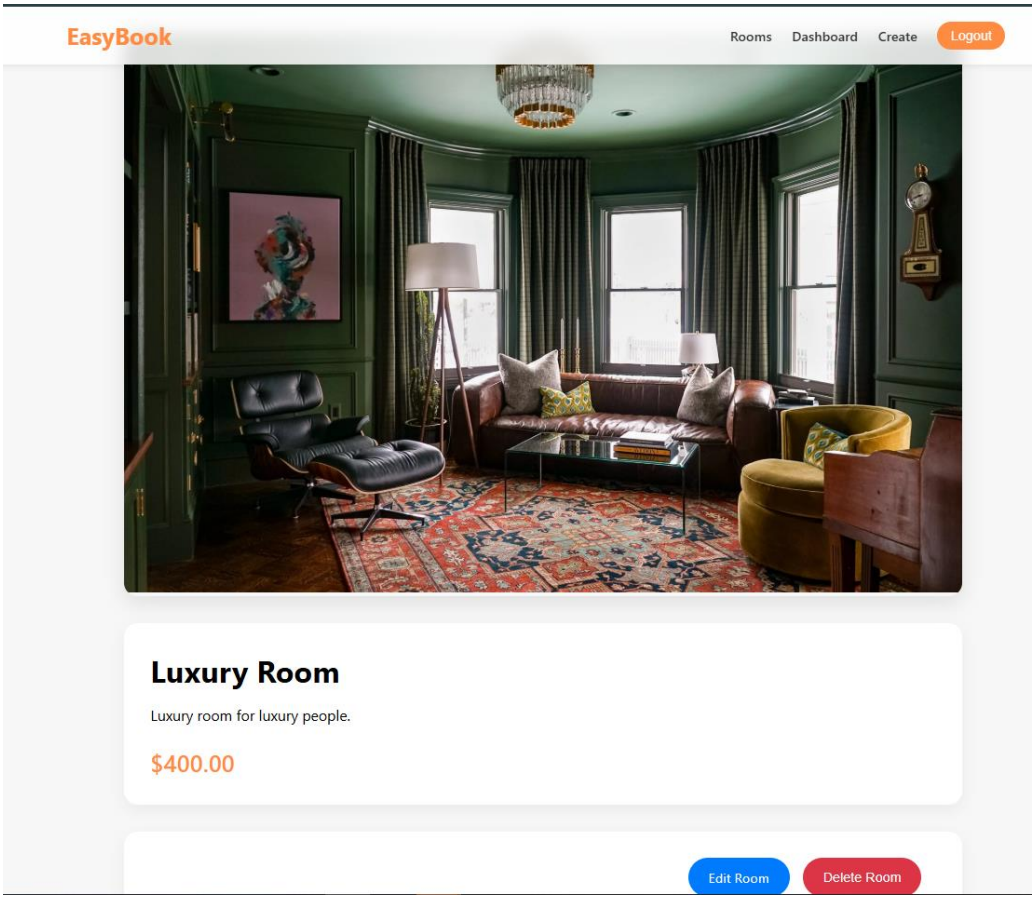

Рисунок 4.10 – Результат зміни даних

Щоб перевірити правильну роботу бронювання номерів, змінимо користувача на одного зі створених в процесі самостійного тестування, який не матиме прав адміністратора. Одночасно перевіряється і робота кнопки «Logout», яка закінчує поточну сесію.

Обираємо номер (рисунок 4.11) натискаємо «Book Now» (рисунок 4.12), заповнюємо повне ім’я, електронну пошту користувача та обираємо початкову та кінцеву дати через календар. Вартість бронювання рахується від кількості ночей. Після натискання кнопки «Confirm» наше бронювання з’являється на сторінці «Dashboard», де можна побачити всі бронювання, зроблені цим користувачем та інформацію про них (рисунок 4.13). Змінимо відповідні поля у нашому бронюванні, щоб перевірити роботу функції (рисунок 4.14).

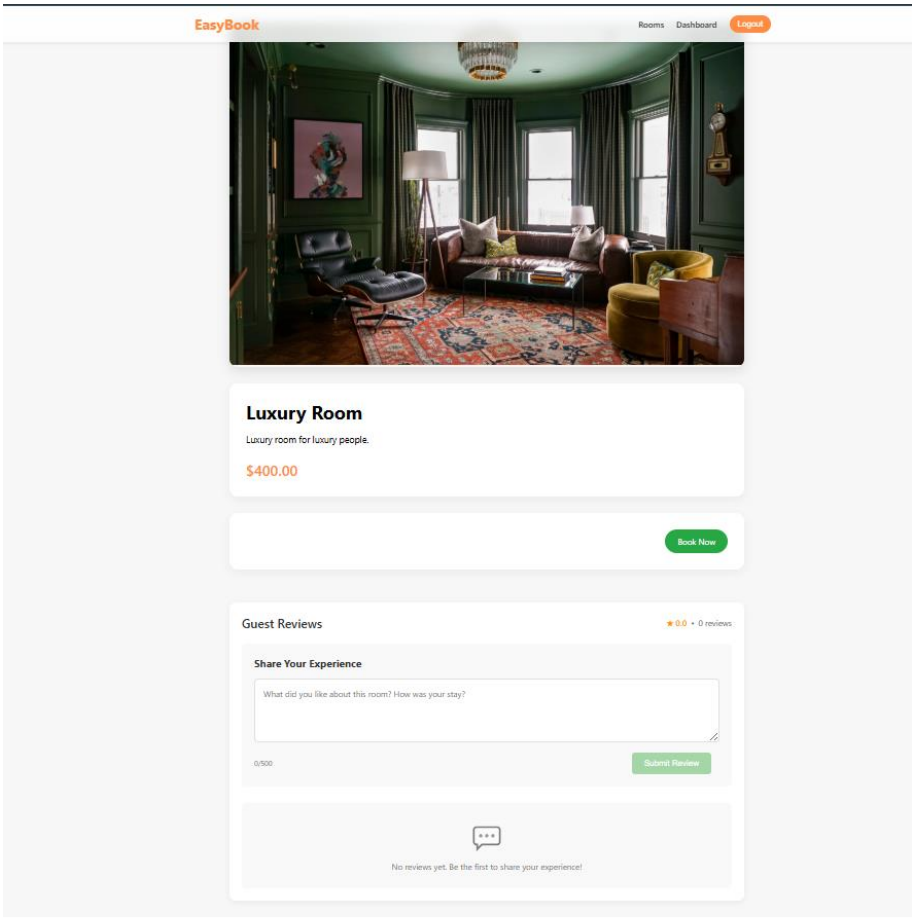

Рисунок 4.11 – Інтерфейс сторінки кімнати для користувача

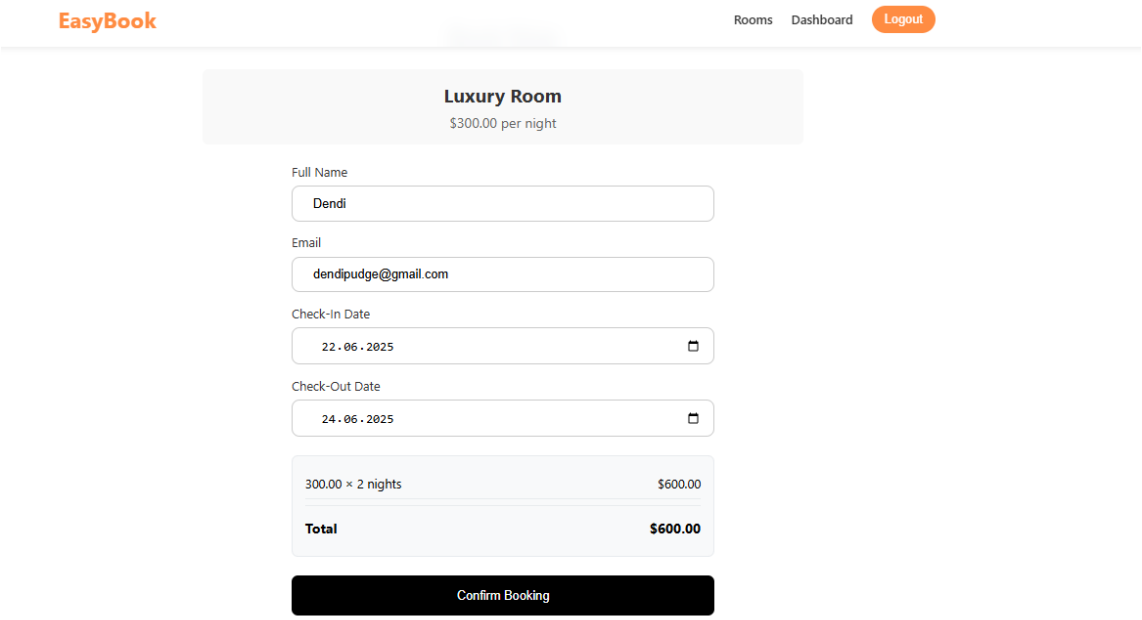

Рисунок 4.12 – Замовлення бронювання

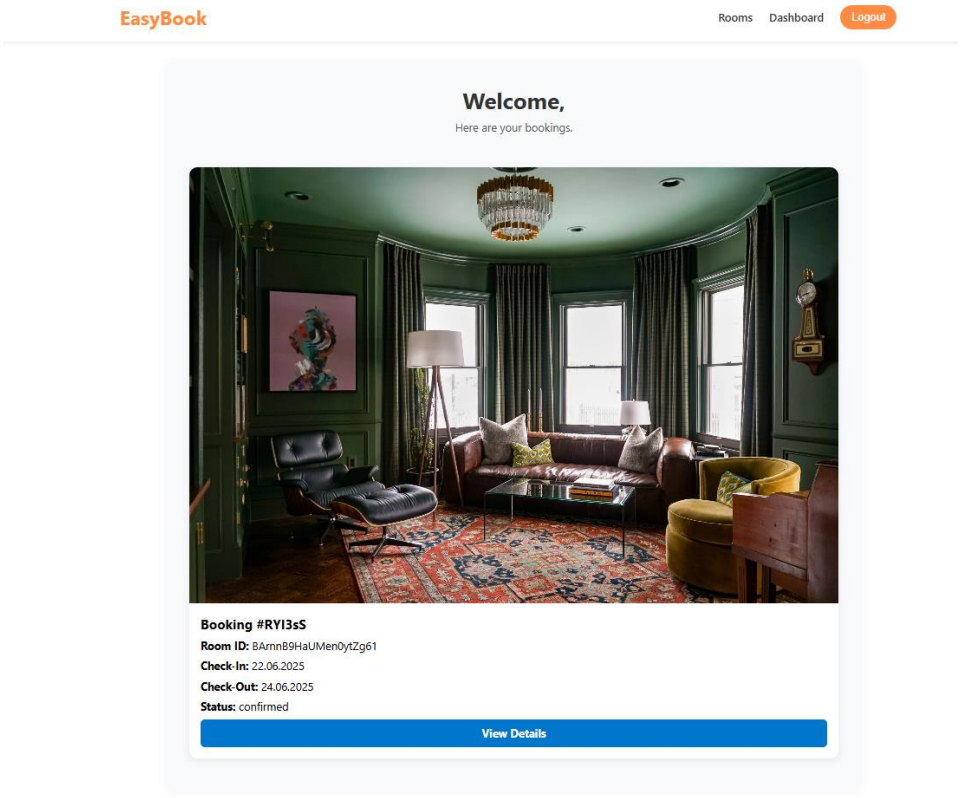

Рисунок 4.13 – Сторінка з усіма замовленнями користувача

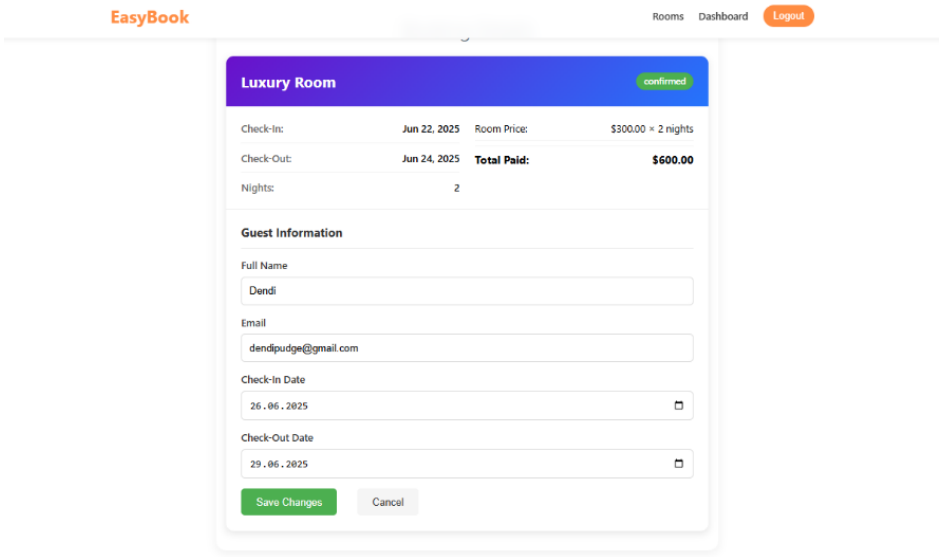

Рисунок 4.14 – Зміна бронювання

Для завершення тестування функцій, знов почнемо сесію користувача з правами адміністратора, та видалимо кімнату (рисунок 4.15). Після видалення кімнати також видаляється і бронювання на неї, що можна дізнатися з таблиці «bookings» бази даних (рисунок 4.16).

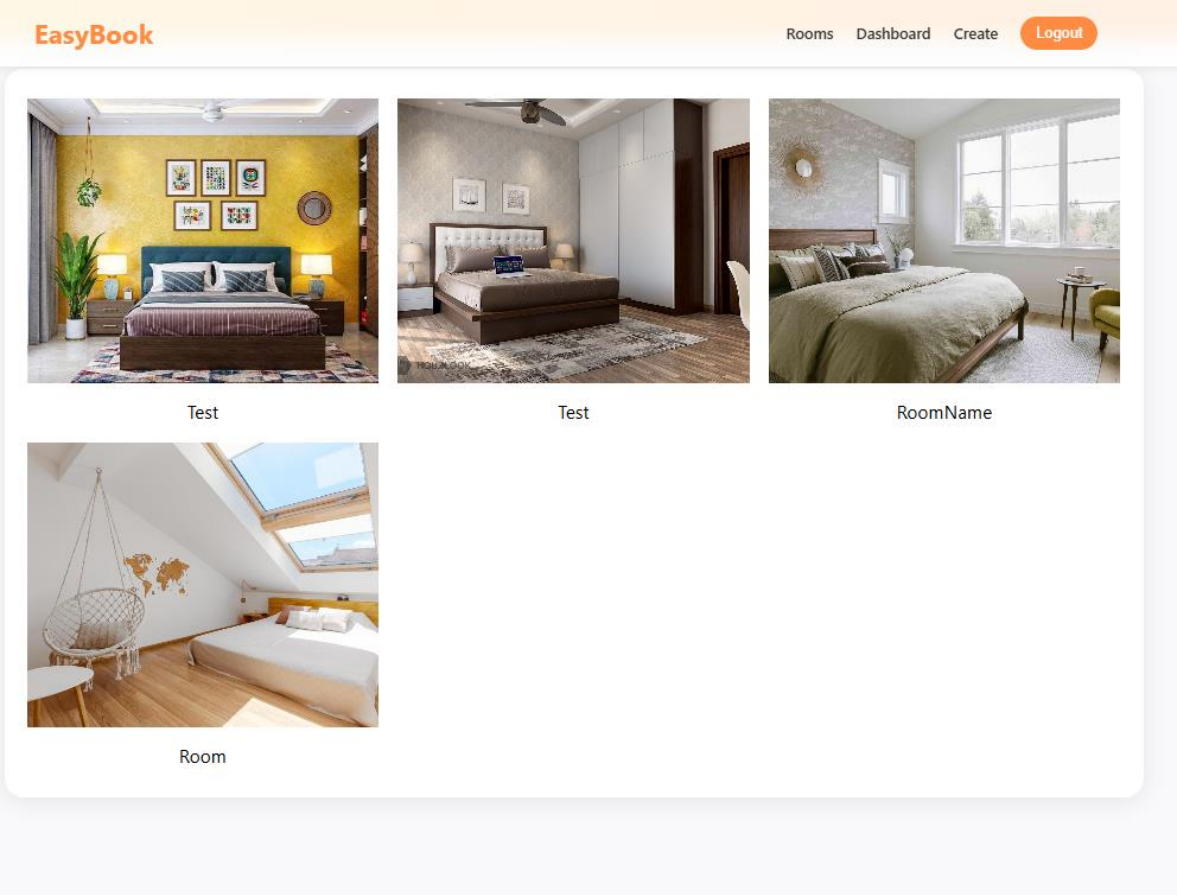

Рисунок 4.15 – Результат видалення кімнати

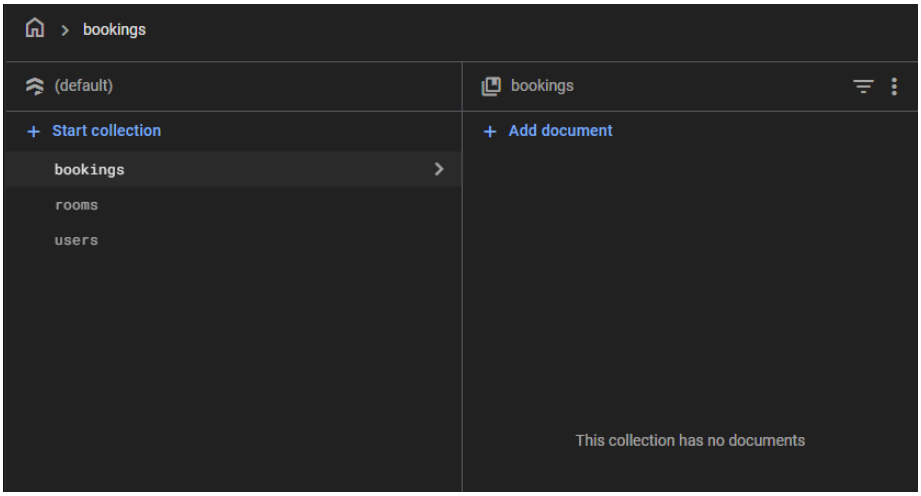

Рисунок 4.16 – Перевірка бази даних

ВИСНОВКИ

Під час виконання роботи було розроблено сайт для букінгу готельних номерів. На основі аналізу конкурентів було визначено пріоритети у розробці та реалізовано: сучасний та зручний інтерфейс, функції, необхідні користувачам: реєстрація, авторизація, перегляд кімнат, бронювання та редагування бронювань, можливість залишати відгуки.

Для розробки було обрано мову програмування JavaScript, бо це найкраще рішення для реалізації серверної та клієнтської частини без використання декількох мов програмування. Такий підхід дозволив розробити повноцінну веб-платформу єдиною мовою програмування у єдиному середовищі розробки, що значно спростило процес інтеграції між компонентами та пришвидшило розробку. Серверна логіка реалізована за допомогою платформи Node.js, яка забезпечує надійне асинхронне оброблення запитів і дозволяє створювати масштабовані API, що особливо важливо для систем з активною взаємодією користувачів і динамічною базою даних. Клієнтська частина побудована на компонентній бібліотеці React, яка дозволяє створювати швидкий, зручний та адаптивний інтерфейс із сучасним рівнем взаємодії. Завдяки такому поєднанню забезпечується висока швидкість роботи додатку та гнучкість у розробці.

У майбутньому на сайті також можуть бути реалізовані важливі функції: пошук кімнат за назвою чи описом, додання інтерактивної мапи де можна подивитись розташування готелів та номерів, чат для комунікації користувачів та адміністрації. Це насправді необхідно та може стати перевагою над іншими платформами. Також сайт буде адаптовано під мобільну версію.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Wandschneider M. Learning Node.js: A Hands-On Guide to Building Web Applications in JavaScript, 2013. – 301 с. – ISBN 978-0321910578.

2. Cantelon M., Harter M., Holowaychuk T., Rajlich N. Node.js in Action. 2014. – 400 с. – ISBN 978-1617290572.

3. Krasner K. Express.js Guide: The Comprehensive Book on Express.js. 2014. – 252 с. – ISBN 978-1782162320.

4. Buehler A. Learning Express.js: Building Web Applications with Node and Express. 2015. – 210 с. – ISBN 978-1784392251.

5. Banks A., Porcello E. Learning React: Functional Web Development with React and Redux. 2017. – 350 с. – ISBN 978-1491954621.

6. Abramov D., Clark A. Redux Essentials. 2021. – 215 с. – https://redux.js.org (офіційна документація).

7. Gheorghita B. Mastering Redux Toolkit: Efficient Redux Development for Modern React Applications. 2021. – 180 с. – ISBN 979-8715654054.