МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА СЕРВІСУ ДЛЯ БРОНЮВАННЯ ГОТЕЛЬНИХ НОМЕРІВ З ВИКОРИСТАННЯМ TYPESCRIPT ТА NODE.JS»**

Виконав: студент 4 курсу, групи 6.1221
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми Комп'ютерні науки
(назва освітньої програми)
А.Ю. Пшинишний
(ініціали та прізвище)
Керівник професор кафедри комп'ютерних наук, доцент, д.е.н., Полуектова Н.Р.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри програмної інженерії, доцент, к.ф.-м.н. Лісняк А. О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 122 Комп'ютерні науки
(шифр і назва)

Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук,
д.т.н., професор

_____ Шило Г.М..
(підпис)

" 23 " 12 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Пшинишному Андрію Юрійовичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Розробка сервісу для бронювання готельних номерів з використанням TypeScript та Node.js

керівник роботи Полуектова Наталія Робертівна, д.е.н., доцент
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » грудня 2024 року № 2001-с

2. Строк подання студентом роботи 02.06.2025

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка сервісу для бронювання готельних номерів.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 23.12.2024

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	24.12.2024	
2.	Збір вихідних даних.	10.01.2025	
3.	Обробка методичних та теоретичних	03.02.2025	
	джерел.		
4.	Розробка першого та другого розділу.	17.03.2025	
5.	Розробка третього розділу.	13.04.2025	
6.	Оформлення та нормоконтроль	28.05.2025	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	18.06.2025	

Студент _____ А.Ю. Пшинишний
(підпис) (ініціали та прізвище)

Керівник роботи _____ Н.Р. Полуектова
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г.Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка сервісу для бронювання готельних номерів з використанням TypeScript та Node.js»: 68 с., 18 рис., 11 джерел, 1 додаток.

АДМІНІСТРАТИВНА ПАНЕЛЬ, БАЗА ДАНИХ, БРОНЮВАННЯ, ВЕБ-СЕРВІС, ГОТЕЛЬНІ НОМЕРИ, ІНТЕРФЕЙС КОРИСТУВАЧА, МАРШРУТИЗАЦІЯ, EXPRESS.JS, MONGODB, NODE.JS, TYPESCRIPT.

Об’єкт дослідження – автоматизація процесу онлайн-бронювання готельних номерів за допомогою веб-технологій.

Мета роботи: розробити веб-сервіс для онлайн-бронювання готельних номерів з використанням технологій TypeScript та Node.js, який забезпечує зручний інтерфейс для користувачів і адміністратора, ефективне управління бронюваннями та збереження даних у базі даних.

Методи дослідження – аналіз та синтез, порівняння, опис, експеримент

В результаті розроблено web-додаток для онлайн-бронювання готельних номерів на основі описаних технологій, який може використовуватись на практиці.

SUMMARY

Bachelor's Qualifying Theses «Developing of a Hotel Room Booking Service Using TypeScript and Node.js»: 68 pages, 18 figures, 18 references, 1 supplement.

ADMIN PANEL, BOOKING, DATABASE, EXPRESS.JS, HOTEL ROOMS, MONGODB, NODE.JS, ROUTING, TYPESCRIPT, USER INTERFACE, WEB SERVICE.

Object of the study – automation of the online hotel room booking process using web technologies.

Aim of the study: to develop a web service for online hotel room booking utilizing TypeScript and Node.js technologies, which offers a user-friendly interface for both users and administrators, enables efficient booking management, and ensures secure data storage in a database.

Method of research – analysis and synthesis, comparison, description, experiment.

As a result, a web application for online hotel room booking was developed based on the described technologies, which can be used in practice.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ..... 7
1 Інформаційний огляд 8
1.1 Актуальність веб-додатків для бронювання готелів 8
1.2 Аналіз існуючих систем бронювання 9
1.3 Аналіз технологій, що використовуються у веб-сервісах бронювання 10
2 Розробка сервісу для бронювання готельних номерів 12
2.1 Постановка задачі 12
2.2 Стек технологій для розробки 13
2.3 Опис обраних технологій для frontend..... 15
2.4 Опис обраних технологій для backend..... 16
2.5 Інші бібліотеки та інструменти 17
2.6 Архітектура системи..... 18
3 Програмна реалізація..... 19
3.1 Реалізація backend 19
3.2 Реалізація frontend..... 22
3.3 Опис функцій проекту 24
3.4 Інструкція по роботі з програмним продуктом 24
Висновки 31
Перелік посилань..... 32
Додаток А..... 34

ВСТУП

У сучасному світі сфера туризму та готельного бізнесу активно розвивається, що створює підвищений попит на ефективні та зручні цифрові рішення для організації бронювання готельних номерів. Традиційні способи бронювання часто виявляються недостатньо гнучкими та швидкими для користувачів, тому створення інноваційних веб-сервісів, які забезпечують автоматизацію цього процесу, є актуальним завданням.

Метою даної роботи є розробка сервісу для бронювання готельних номерів із застосуванням сучасних технологій – TypeScript та Node.js. Використання цих технологій дозволяє створити надійний, масштабований та зручний у підтримці сервіс, що відповідає вимогам користувачів і особливостям готельного бізнесу. У роботі буде розглянуто архітектуру системи, її основні функціональні можливості, а також проведено тестування розробленого сервісу.

Розробка такого сервісу сприятиме підвищенню якості обслуговування клієнтів, скороченню часу на бронювання та оптимізації роботи готельного персоналу. Крім того, впровадження цифрових інструментів допоможе готелям залишатися конкурентоспроможними на ринку та швидко адаптуватися до змін у вимогах користувачів.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Актуальність веб-додатків для бронювання готелів

Онлайн-бронювання готелів сьогодні є невід'ємною частиною туристичного ринку: у 2024 році глобальний ринок онлайн-бронювання оцінювався в 14,63 млрд USD із прогнозованим зростанням до 526,40 млрд USD до 2031 року (CAGR 15,25 %) [1].

Загалом, у 2024 році світовий онлайн-туристичний ринок досяг 523 млрд USD і, за прогнозами, зросте до 1,3 трлн USD до 2030 року, демонструючи 13,1 % щорічного зростання, що підкреслює активне зміщення користувачів у бік цифрових платформ [2].

Впровадження мобільних додатків для бронювання дозволило готелям суттєво підвищити власну видимість і залучити широку аудиторію, адже 74 % опитаних планують здійснити внутрішні поїздки, а 59 % – міжнародні, використовуючи онлайн-сервіси у 2025 році [3].

Згідно з даними SiteMinder, у Сполучених Штатах лише готельний ринок перевищує 106 млрд USD, а до 2027 року очікується залучення понад 160 млн користувачів до цифрових каналів бронювання [4].

Поява веб-сервісів бронювання дозволила оптимізувати внутрішні процеси готелів: автоматизація обробки заявок і платежів знизила витрати на комісії та скоротила час обслуговування гостей [5].

Користувачі цінують миттєве підтвердження бронювання та можливість самостійно обирати й змінювати дати перебування в режимі 24/7, що підвищує задоволеність і лояльність клієнтів [6].

Одночасно індустрія готельних послуг стикається з високою конкуренцією: онлайн-агрегатори на кшталт Booking.com та Airbnb стимулюють готелі впроваджувати власні веб-движки, щоб зменшити комісію й отримати прямі продажі [7].

Сучасні тренди 2025 року в готельному бізнесі включають розвиток мобільних чек-інів, персоналізовані пропозиції на основі AI-аналітики та екологічні ініціативи, що також інтегруються у веб-додатки бронювання для задоволення очікувань гостей [8].

Надання користувачеві можливості порівнювати ціни, вибирати додаткові послуги (трансфер, сніданок) і користуватися системою відгуків безпосередньо на сайті готелю підвищує конверсію та прибутковість підприємства [9].

Зростання попиту на «bleisure» туризм, поєднання бізнес та відпочинкового відвідування, призвело до збільшення середнього терміну перебування і середнього чеку, що також враховується у функціоналі веб-додатків для бронювання [10].

Висока динаміка розвитку онлайн-платформ створює необхідність постійного вдосконалення безпеки даних і захисту платежів через інтеграцію сучасних протоколів шифрування та відповідність стандартам PCI DSS й GDPR [11].

Таким чином, розробка власного веб-додатку для бронювання готельних номерів забезпечує бізнесу незалежність від сторонніх OTA, підвищує якість обслуговування та відкриває нові можливості монетизації без значних операційних витрат.

1.2 Аналіз існуючих систем бронювання

На сучасному ринку існує велика кількість систем онлайн-бронювання готельних номерів, серед яких найбільш відомими є такі сервіси, як Booking.com, Airbnb, Hotels.com, Expedia, Trip.com та інші. Вони пропонують широкий спектр можливостей: пошук житла за різними параметрами (ціна, зручності, локація), відгуки користувачів, онлайн-оплата, календар доступності, підтвердження бронювання електронною поштою та інші функції. Ці сервіси мають зручний інтерфейс, підтримують різні мови та дозволяють швидко знайти і забронювати житло у більшості країн світу.

Попри велику функціональність і популярність, існуючі системи мають і певні недоліки. Зокрема, вони часто орієнтовані на великий або міжнародний ринок, тому процес реєстрації та управління об'єктами може бути складним і не зовсім зручним для власників невеликих готелів, хостелів або гостьових будинків. Крім того, багато платформ стягують комісію з готелів, що впливає на кінцеву вартість проживання для клієнта. У деяких випадках користувачам пропонуються додаткові функції, які ускладнюють інтерфейс і не є критично необхідними для базового процесу бронювання.

У рамках даної дипломної роботи пропонується розробити власний вебсервіс для бронювання готельних номерів, який буде орієнтований на простоту використання, зручність для адміністратора готелю, швидкодію і можливість подальшого масштабування. Такий підхід дозволить створити адаптоване рішення для малих і середніх готелів, використовуючи сучасні технології, зокрема TypeScript та Node.js, і забезпечити можливість легкого розширення функціоналу залежно від потреб користувачів.

1.3 Аналіз технологій, що використовуються у веб-сервісах бронювання

У сучасному світі веб-сервіси бронювання готелів є невід’ємною частиною індустрії гостинності, забезпечуючи зручний та ефективний спосіб для клієнтів здійснювати бронювання онлайн. Розробка таких сервісів вимагає ретельного вибору технологій, що забезпечують масштабованість, безпеку та зручність використання.

Для створення ефективного веб-сервісу бронювання готелів важливо обрати відповідний стек технологій. На фронтенді можуть використовуватись фреймворки React, Vue або Angular, які забезпечують динамічний та інтерактивний інтерфейс користувача. Однак, у випадку використання серверного рендерингу з Node.js та Express, шаблонізатор EJS (Embedded JavaScript) є популярним вибором для генерації динамічного HTML. EJS

дозволяє вбудовувати JavaScript-код безпосередньо в HTML-шаблони, що спрощує створення динамічного контенту на сервері та забезпечує швидке завантаження сторінок.

На бекенді широко використовується Node.js разом з фреймворком Express, який дозволяє створювати швидкі та масштабовані серверні додатки. Express забезпечує зручну маршрутизацію та обробку HTTP-запитів, а також підтримує використання middleware для обробки запитів та забезпечення безпеки. Для обробки запитів та управління маршрутизацією застосовуються middleware, такі як body-parser та helmet, що забезпечують безпеку та ефективність обробки даних. Аутентифікація та авторизація користувачів реалізуються за допомогою JWT або OAuth2, що гарантує безпечний доступ до ресурсу.

Зберігання даних є критичним аспектом веб-сервісу бронювання. MongoDB, як документно-орієнтована база даних, надає гнучкість у моделюванні даних, що особливо корисно для динамічних структур, таких як інформація про готелі, номери та бронювання. Використання ODM-бібліотеки Mongoose дозволяє визначати схеми даних та забезпечує валідацію, що сприяє підтримці цілісності даних.

Інфраструктура розгортання та підтримки веб-сервісу включає використання контейнеризації за допомогою Docker, що забезпечує консистентність середовища розробки та продакшну. Оркестрація контейнерів здійснюється за допомогою Docker Compose або Kubernetes, що дозволяє ефективно управляти масштабуванням та доступністю сервісу. Веб-сервер Nginx використовується як реверс-проксі та для обслуговування статичних файлів, а Certbot автоматизує отримання та оновлення SSL-сертифікатів, забезпечуючи безпечне з'єднання.

Таким чином, вибір та інтеграція сучасних технологій у веб-сервіс бронювання готелів є ключовими для забезпечення його ефективності, масштабованості та безпеки. Правильна архітектура та використання перевірених рішень дозволяють створити надійну платформу, яка відповідає очікуванням користувачів та вимогам бізнесу.

2 РОЗРОБКА СЕРВІСУ ДЛЯ БРОНЮВАННЯ ГОТЕЛЬНИХ НОМЕРІВ

2.1 Постановка задачі

У сучасному цифровому середовищі індустрія гостинності активно впроваджує інформаційні технології для оптимізації процесів обслуговування клієнтів. Одним із ключових напрямків є автоматизація процесу бронювання готельних номерів через веб-сервіси. Існуючі платформи, такі як Booking.com, Airbnb та інші, надають широкі можливості для користувачів, проте вони не завжди враховують специфічні потреби окремих готелів, особливо малих та середніх, які прагнуть мати власну систему бронювання з урахуванням індивідуальних вимог та бренду.

Метою даної дипломної роботи є розробка веб-сервісу для бронювання номерів у готелі, який забезпечить зручний інтерфейс для користувачів та ефективні інструменти управління для адміністрації готелю. Сервіс повинен підтримувати повний цикл бронювання: від пошуку вільних номерів до підтвердження бронювання та обробки платежів.

Для реалізації цього проєкту планується використання сучасного стеку технологій, зокрема Node.js як середовища виконання JavaScript на сервері, що забезпечує високу продуктивність та масштабованість; Express.js як фреймворку для побудови серверної частини веб-додатку, що спрощує розробку маршрутизації та обробки запитів; EJS (Embedded JavaScript) як шаблонізатору для генерації динамічного HTML-контенту на стороні сервера; MongoDB як документно-орієнтованої бази даних для зберігання інформації про користувачів, номери, бронювання та інші сутності.

Основні функціональні вимоги до системи включають:

- можливість реєстрації та авторизації користувачів з різними ролями (адміністратор, гість);

- інтерфейс для пошуку доступних номерів, механізм бронювання номерів з підтвердженням та можливістю скасування;
- панель адміністратора для управління номерами, перегляду та обробки бронювань;
- забезпечення безпеки даних користувачів та захист від несанкціонованого доступу.

Реалізація даного веб-сервісу сприятиме покращенню процесу бронювання для клієнтів та оптимізації внутрішніх процесів управління для адміністрації готелю. Використання сучасних технологій забезпечить масштабованість, надійність та зручність у користуванні системою.

2.2 Стек технологій для розробки

Для розробки веб-сервісу бронювання готелів було обрано сучасний і перевірений стек, який поєднує безпечні практики, гнучкість та високу продуктивність.

TypeScript використовується як основна мова програмування на боці сервера, що забезпечує статичну типізацію, покращену підтримку IDE і зниження кількості помилок на етапі розробки. Node.js служить середовищем виконання JavaScript-коду на сервері і завдяки неблокуючому I/O дозволяє обробляти велику кількість одночасних з’єднань з низькою затримкою.

Express, як мінімалістичний та гнучкий фреймворк для Node.js, надає зручні інструменти маршрутизації та підтримку middleware, що спрощує побудову RESTful API та інтеграцію з іншими компонентами системи. Для рендерингу HTML на сервері обрано шаблонізатор EJS, який дозволяє вбудовувати JavaScript-логіку безпосередньо в шаблони та забезпечує високу швидкість генерації сторінок.

MongoDB використовується як документно-орієнтоване сховище даних, що ідеально підходить для зберігання гнучких структур бронювань, номерів та

профілів користувачів. Підключення до MongoDB реалізовано через ODM-бібліотеку Mongoose, яка вводить суворі схеми, валідацію даних та middleware для обробки подій запису/читання.

Для безпеки паролів використовується bcrypt, що забезпечує надійне хешування із сольовим фактором, а cookie-parser у поєднанні з JSON Web Token (jsonwebtoken) реалізують зберігання та перевірку токенів у HTTP-куках для аутентифікації й авторизації. Для валідації вхідних даних застосовано Joi, що гарантує коректність запитів до API та унеможливлює некоректні операції з базою даних.

Для завантаження файлів (наприклад, фотографій кімнат) використовується Multer, а генерацію унікальних ідентифікаторів – бібліотека uuid. Для утилітних операцій, таких як дебаунсінг чи клонування об’єктів, застосовано Lodash, що спрощує написання чистого й зрозумілого коду.

Конфігурацію середовища реалізовано через dotenv, що дозволяє зберігати секрети та параметри підключень у файлі .env, недоступному в репозиторії.

Інфраструктура розгортання ґрунтується на Docker із Docker Compose, завдяки чому забезпечується консистентність середовища на всіх етапах: розробка, тестування та продакшен. Nginx виконує роль реверс-проксі та статичного сервера, а Certbot автоматизує отримання та оновлення SSL-сертифікатів, забезпечуючи безпечне HTTPS-з’єднання. Деплой на DigitalOcean спрощує масштабування та управління Droplet’ами, забезпечуючи високу доступність сервісу.

Збалансоване поєднання цих технологій дозволяє створити надійний, масштабований і безпечний веб-сервіс бронювання готелів із мінімальними витратами на підтримку та розвиток.

Діаграма стеку технологій на рисунку 2.1.

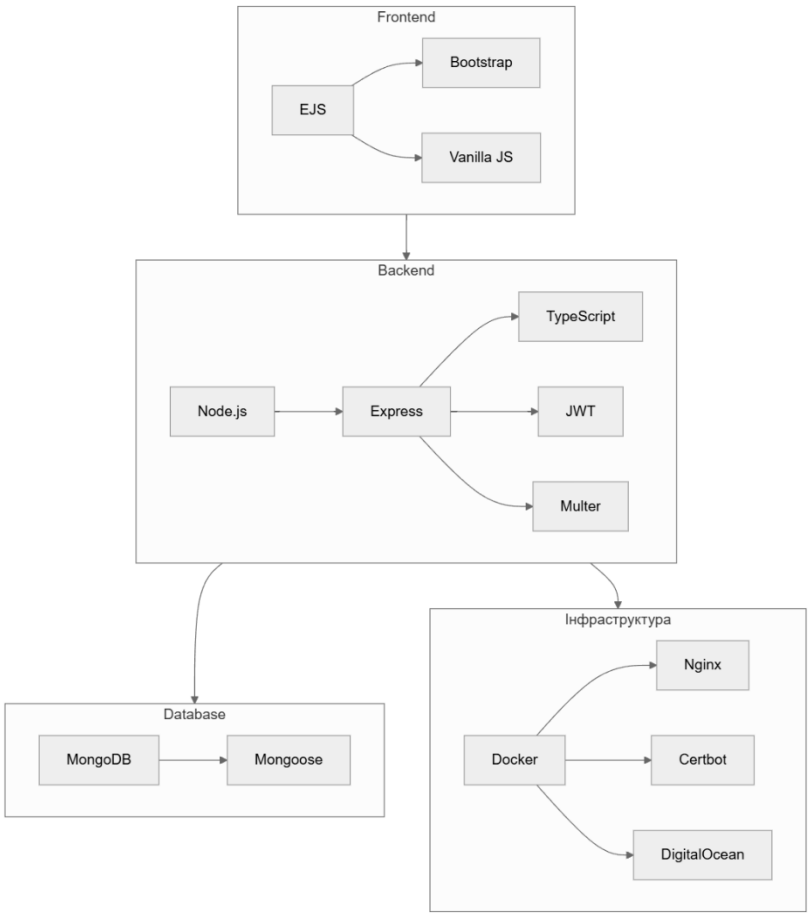


Рисунок 2.1 – Діаграма стеку технологій

2.3 Опис обраних технологій для frontend

Для клієнтської частини веб-сервісу бронювання готелів обрано поєднання серверного шаблонізатора, адаптивної стилізації та мінімальної клієнтської логіки, що забезпечує швидке завантаження сторінок, простоту підтримки та зручність для користувачів.

Express рендерить HTML-шаблони за допомогою EJS, дозволяючи вбудовувати JavaScript-логіку безпосередньо в розмітку. Це дає змогу серверу повертати вже сформовану HTML-сторінку з усіма необхідними даними – списком кімнат із фотографіями, ціною, типом і доступністю, а також формою для вибору дат. Завдяки partials (заголовок, підвал, навігаційне меню) базова структура сайту залишається модульною й легко підтримується.

Для стилізації інтерфейсу використовуються Bootstrap і кастомні CSS. Bootstrap забезпечує адаптивну верстку “з коробки” за допомогою готових компонентів (карточки для кімнат, навігаційна панель, кнопки, форми), грід-системи для розмітки та утилітних класів для відступів і вирівнювання. Це дозволяє швидко створювати сучасний і лаконічний дизайн без необхідності писати великий обсяг власного CSS.

Інтерактивність реалізована за допомогою невеликих скриптів на чистому JavaScript, що підключаються у відповідних EJS-шаблонах. Наприклад, для відображення сповіщень про успішне або невдале бронювання – динамічні модальні вікна Bootstrap. Такий підхід мінімізує складність клієнтської частини, дозволяючи уникнути важких JavaScript-фреймворків, але водночас забезпечує базову динаміку сторінок.

Загалом, поєднання EJS для серверного рендерингу, Bootstrap для адаптивного й інтерактивного дизайну та мінімального JavaScript робить фронтенд частиною простою в розумінні, легкою у підтримці та достатньо функціональною для реалізації всіх необхідних сценаріїв користувача: від перегляду списку кімнат до оформлення бронювання.

2.4 Опис обраних технологій для backend

Для побудови серверної логіки веб-сервісу бронювання готелів було обрано поєднання Node.js та TypeScript, що забезпечує високу продуктивність завдяки неблокуючому I/O і статичну типізацію для зменшення помилок на етапі розробки. Як основний фреймворк використано Express – мінімалістичний і гнучкий інструмент для маршрутизації та обробки HTTP-запитів, який підтримує велику екосистему middleware і дозволяє швидко створювати RESTful API.

Для роботи з базою даних обрана MongoDB, а доступ до неї реалізовано через Mongoose – ODM-бібліотеку, що дозволяє задавати суворі схеми, валідувати дані та використовувати middleware при операціях читання/запису.

Вхідні дані запитів перевіряються за допомогою Joi, що забезпечує гнучке описання схем валідації та інтегрується з Express як middleware. Аутентифікація й авторизація користувачів базуються на JWT (jsonwebtoken) та bcrypt: bcrypt відповідає за безпечне хешування паролів із сольовим фактором, а jsonwebtoken – за генерацію й перевірку токенів у HTTP-куках через cookie-parser.

Для завантаження файлів (наприклад, фотографій номерів) застосовано Multer – middleware на основі busboy, що спрощує обробку multipart/form-data та дозволяє налаштовувати зберігання файлів на диску або в пам’яті. Генерацію унікальних ідентифікаторів у різних частинах системи (ім’я файлу, токени відновлення) забезпечує бібліотека uuid. Для утилітарних операцій, таких як клонування об’єктів і робота з масивами, використовується Lodash, що знижує складність коду та підвищує його читабельність.

Конфігурація середовища виконання реалізована через dotenv, що дозволяє завантажувати змінні оточення з файлу .env у process.env відповідно до Twelve-Factor App методології. Усі секрети та параметри підключень (рядок до БД, секрети JWT) зберігаються поза репозиторієм, що підвищує безпеку та спрощує роботу в різних середовищах.

Цей набір технологій – TypeScript, Node.js, Express, Mongoose, Joi, bcrypt, jsonwebtoken, Multer, uuid, Lodash та dotenv – створює надійну, масштабовану та безпечну платформу для серверної частини веб-сервісу бронювання готелів.

2.5 Інші бібліотеки та інструменти

В проєкті також використовуються допоміжні бібліотеки та інструменти: cookie-parser для читання та запису HTTP-куків (зокрема JWT-токенів) в Express-додатку, що спрощує реалізацію безпечної аутентифікації та авторизації користувачів; dotenv для завантаження конфіденційних налаштувань (рядок підключення до бази даних, секрети JWT, ключі API) з файлу .env у process.env, що відповідно до методології Twelve-Factor App забезпечує чистий розподіл конфігурації та коду; Joi для декларативної валідації вхідних даних HTTP-

запитів, захищаючи сервер від некоректних чи потенційно шкідливих запитів; Multer для обробки multipart/form-data (наприклад, завантаження фотографій кімнат); uuid для генерації унікальних ідентифікаторів; Lodash для утилітарних операцій із масивами та об’єктами; Mongoose як ODM-бібліотека для MongoDB, що вводить суворі схеми, валідацію даних та middleware для обробки подій читання/запису ; а також інструменти інфраструктури: Docker із Docker Compose для контейнеризації всіх сервісів, Nginx як реверс-проксі й сервер статичних файлів, Certbot для автоматичного отримання й оновлення SSL-сертифікатів та хостинг на DigitalOcean для простого масштабування й управління продакшен-середовищем.Для побудови.

2.6 Архітектура системи

Діаграма архітектури системи на рисунку 2.2.

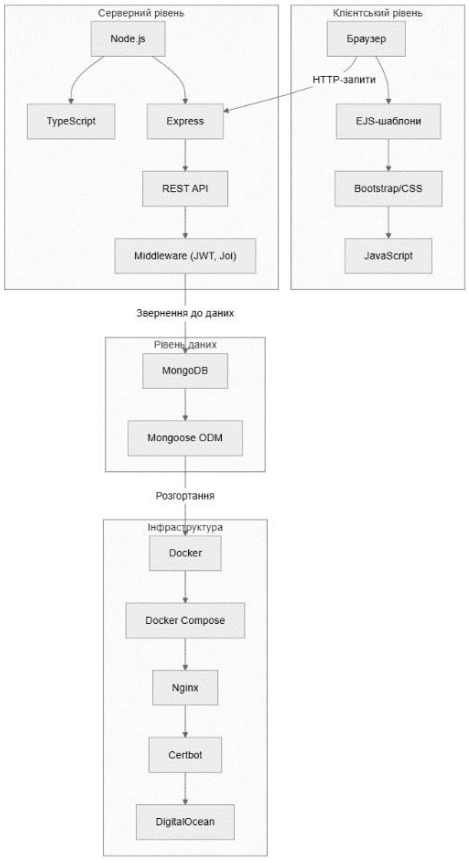


Рисунок 2.2 – Діаграма архітектури системи

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Реалізація backend

Програмну реалізацію серверної частини сайту можна розділити на дві частини: проєктування та створення структури бази даних та розробку API для клієнта. При проєктуванні бази даних виділено три основні сутності, які відображають ключові бізнес-об’єкти сервісу бронювання:

- bookings – колекція, що зберігає інформацію про кожне бронювання. Кожний документ містить унікальний id, посилання на користувача (userId) та кімнату (roomId), дати початку (startDate) і завершення бронювання (endDate), поточний status (наприклад, pending, confirmed, cancelled) і загальну вартість (totalPrice). Додатково автоматично ведуться поля createdAt і updatedAt завдяки опції timestamps;
- rooms – колекція, що описує всі наявні в готелі кімнати. У схемі передбачено id, унікальний номер кімнати (roomNumber), тип (roomType), докладний description, вартість за ніч (pricePerNight), місткість (capacity), масив шляхів до фотографій (photoPaths), а також булеве поле isAvailable, яке сигналізує про загальну готовність кімнати до заселення. Поля дати створення й оновлення також автоматично генеруються;
- users – колекція для зберігання даних про зареєстрованих користувачів. Кожний документ має свій id, ім’я (firstName), прізвище (lastName), контактний email, номер телефону (phone), захищений password і роль (role), що визначає рівень доступу (наприклад, user або admin). Як і в інших сутностях, використовуються createdAt і updatedAt.

ER-діаграма бази даних на рисунку 3.1.

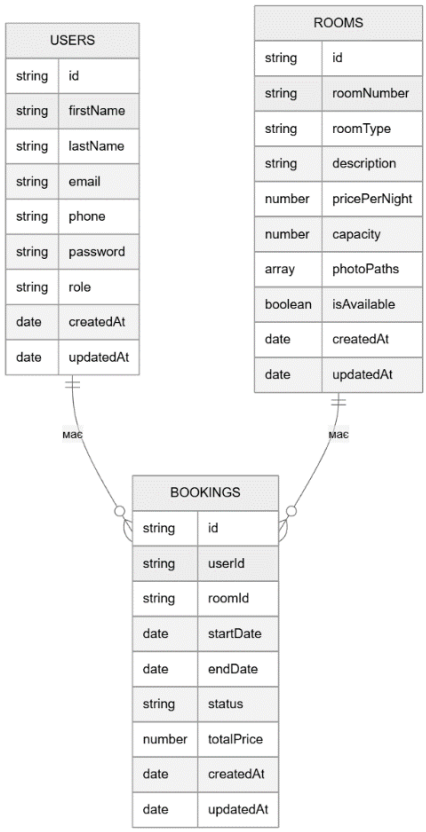


Рисунок 3.1 – ER-діаграма бази даних

В кореневому каталозі розміщено єдиний вхідний файл – server.ts, який ініціалізує Express-сервер, підключення до бази даних і основні middleware. Структура директорій backend частини представлена на рисунку 3.2, де показано організацію папок за функціональними модулями. Решта коду структурується за функціональними папками.

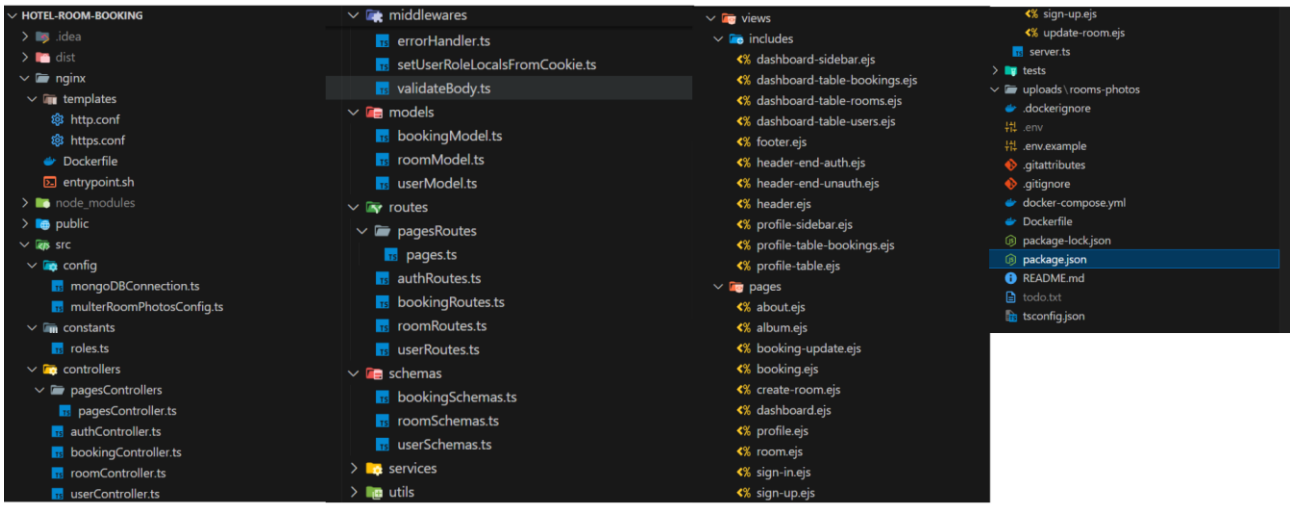


Рисунок 3.2 – Структура директорій серверної частини проєкту

config/ містить файли конфігурації сервера і зовнішніх сервісів:

– mongoDBConnection.ts – ініціалізує підключення до MongoDB через Mongoose, читаючи URI з .env;
– multerRoomPhotosConfig.ts – налаштовує Multer для завантаження фотографій кімнат.

constants/ зберігає константи, що використовуються в усьому проєкті:

– roles.ts – перелік ролей користувачів (наприклад, user, staff, admin).

controllers/ відповідає за бізнес-логіку і обробку HTTP-запитів:

– authController.ts – реєстрація, логін, вихід із сесії;
– bookingController.ts – створення, перегляд, скасування бронювань;
– roomController.ts – CRUD-операції над кімнатами;
– userController.ts – керування профілем користувача;
– pagesControllers/ містить pagesController.ts для рендерингу EJS-сторінок (головна, профіль, дашборд).

middlewares/ зберігає проміжні обробники запитів:

– authorizeRole.ts – перевірка ролі користувача перед виконанням запиту;
– errorHandler.ts – централізована обробка помилок;
– setUserRoleLocalsFromCookie.ts – занесення інформації про поточного користувача в res.locals;
– validateBody.ts – валідація тіла запиту через Joi-схеми.

models/ описує схеми Mongoose для бази даних:

– bookingModel.ts – схема бронювання;
– roomModel.ts – схема кімнати;
– userModel.ts – схема користувача.

routes/ об’єднує всі API-шляхи в роутери:

– authRoutes.ts, bookingRoutes.ts, roomRoutes.ts, userRoutes.ts – REST-інтерфейси для відповідних сутностей;
– pagesRoutes/ містить pages.ts – маршрути для відображення EJS-сторінок.

schemas/ зберігає Joi-схеми для валідації:

– bookingSchemas.ts, roomSchemas.ts, userSchemas.ts.

views/ включає EJS-шаблони і розділена на дві підпапки:

– includes/ – часткові шаблони (header.ejs, footer.ejs, бокові панелі дашборду і профілю, табличні компоненти);
– pages/ – повні сторінки (about.ejs, room.ejs, booking.ejs, дашборд, сторінки реєстрації/логіну тощо).

Повний код серверної частини представлений в додатку А.

3.2 Реалізація frontend

Клієнтська частина реалізована на основі серверного рендерингу з використанням EJS-шаблонізатора, Bootstrap для адаптивного дизайну та мінімальної клієнтської логіки на чистому JavaScript. Вхід у фронтенд забезпечує файл server.ts, де за допомогою Express налаштовано роздачу статичних ресурсів із папки public/ (CSS, JS, зображення) та встановлення EJS як view-engine.

Структура директорій frontend частини представлена на рисунку 3.3.

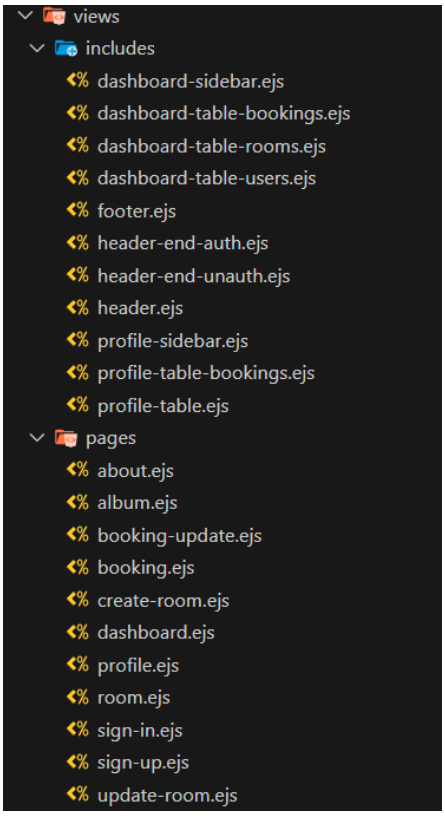


Рисунок 3.3 – Структура директорій клієнтської частини проєкту

У каталозі views/includes/ зібрані загальні фрагменти інтерфейсу:

– header.ejs та footer.ejs – спільні шапка й підвал усіх сторінок;
– dashboard-sidebar.ejs і profile-sidebar.ejs – бокові меню для адміна та користувача;
– табличні partials (dashboard-table-*.ejs, profile-table-*.ejs) – для уніфікованого відображення списків бронювань, кімнат і користувачів.

У views/pages/ розміщено основні сторінки:

– album.ejs – головна сторінка з кімнатами;
– about.ejs – загальна інформація про сервіс;
– room.ejs та booking.ejs – відображення деталей кімнати та форми бронювання;
– create-room.ejs і update-room.ejs – форми для адміна;
– sign-in.ejs, sign-up.ejs – сторінки аутентифікації;
– dashboard.ejs та profile.ejs – інтерфейси адміна й користувача відповідно.

Діаграма компонентів фронтенду на рисунку 3.4.

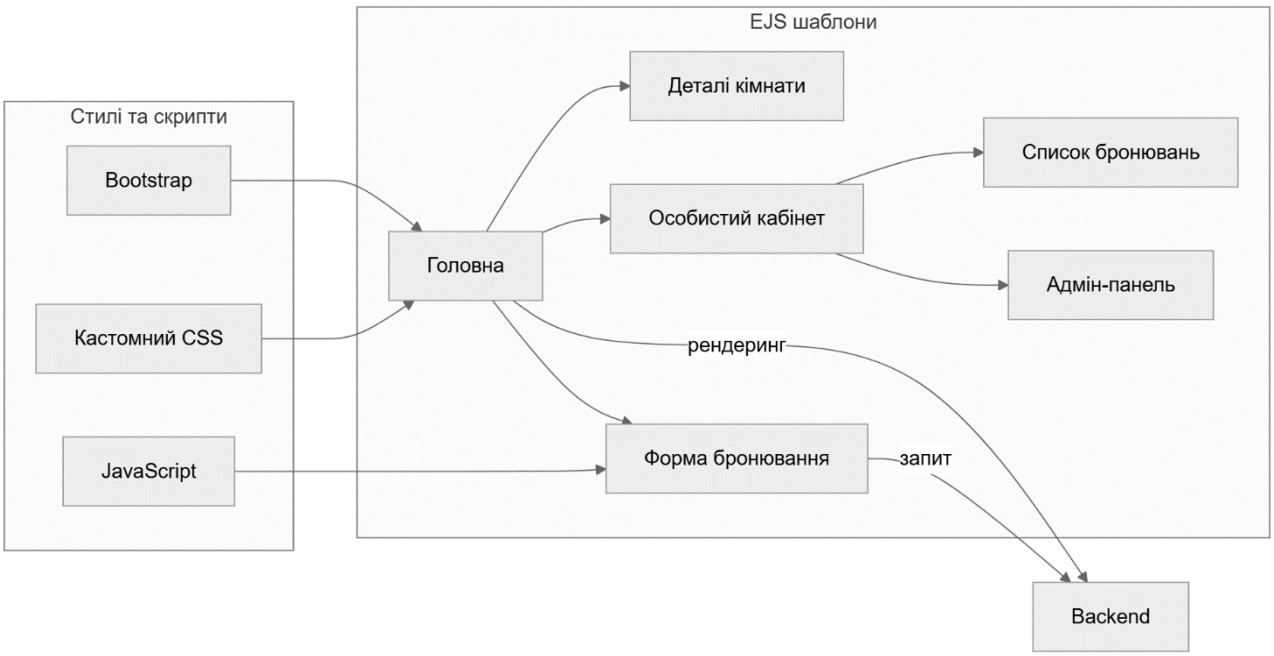


Рисунок 3.4 – Діаграма компонентів фронтенду

3.3 Опис функцій проекту

Розроблений вебсервіс для бронювання готельних номерів надає користувачам зручний інтерфейс для перегляду доступних номерів, оформлення бронювання та керування своїми замовленнями. Адміністратори готелів, у свою чергу, можуть додавати, редагувати та видаляти інформацію про номери, встановлювати ціни, переглядати список бронювань.

Ключовий функціонал проєкту включає:

– реєстрацію та авторизацію користувачів;
– перегляд списку номерів;
– детальну сторінку номера з описом, фотографіями та доступністю;
– можливість створення бронювання;
– особистий кабінет користувача з історією замовлень;
– адміністративну панель, яка дозволяє управляти списком готельних номерів, переглядати інформацію про всіх зареєстрованих користувачів, а також здійснювати керування бронюваннями;
– захист персональних даних користувачів та обмеження доступу до адміністративних функцій;

3.4 Інструкція по роботі з програмним продуктом

Після запуску вебсервісу на головній сторінці (рис. 3.5), користувач може обрати вхід до системи (рис. 3.6) або реєстрацію (рис. 3.7).

Після авторизації відкривається головна сторінка з переліком доступних номерів (рис. 3.8).

При виборі конкретного номера відкривається сторінка з детальною інформацією, де доступна кнопка для створення бронювання (рис. 3.9).

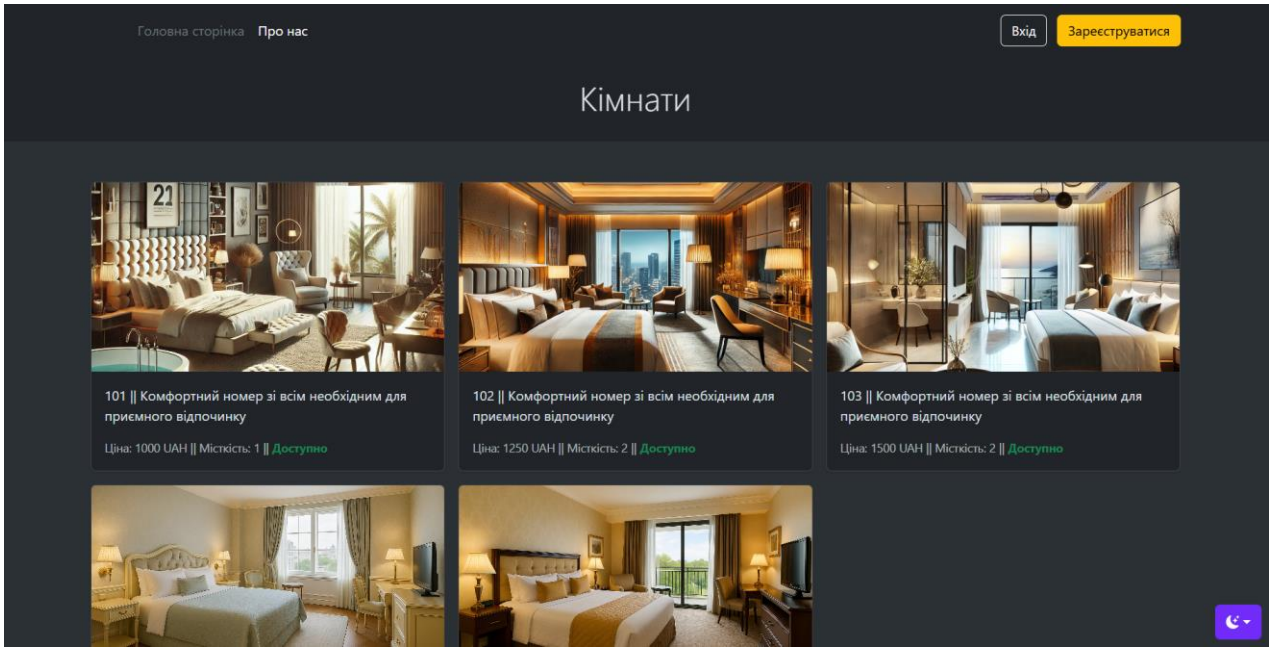


Рисунок 3.5 – Головна сторінка з кнопками «Вхід» та «Зареєструватися»

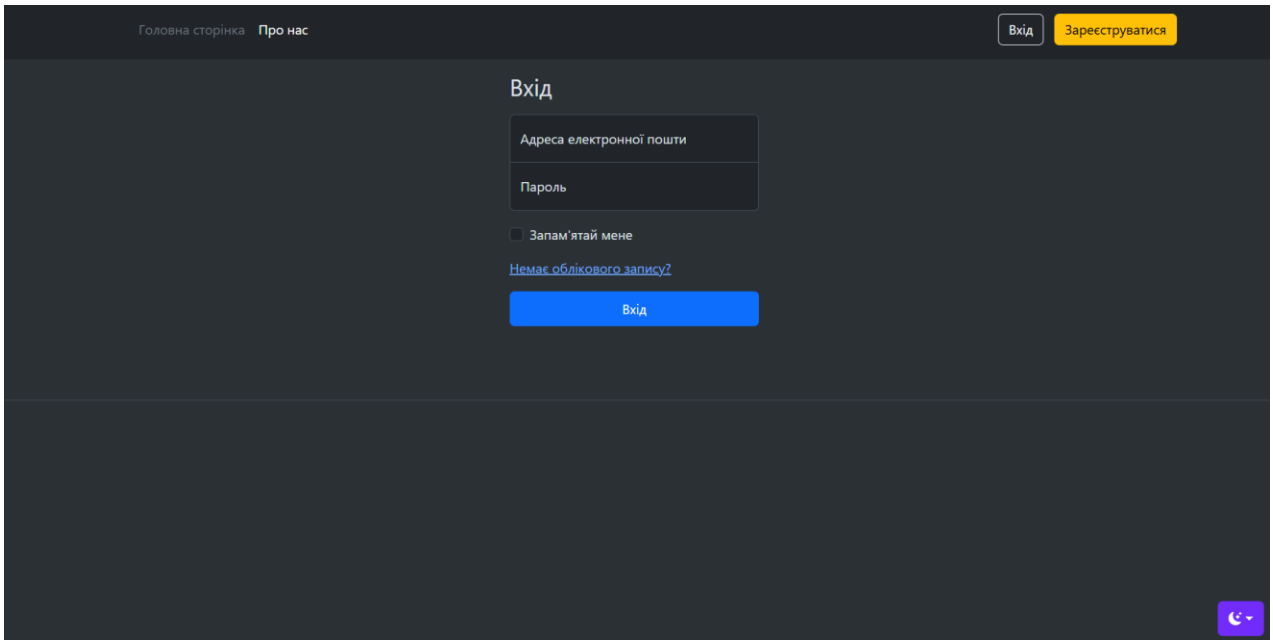


Рисунок 3.6 – Сторінка входу до системи

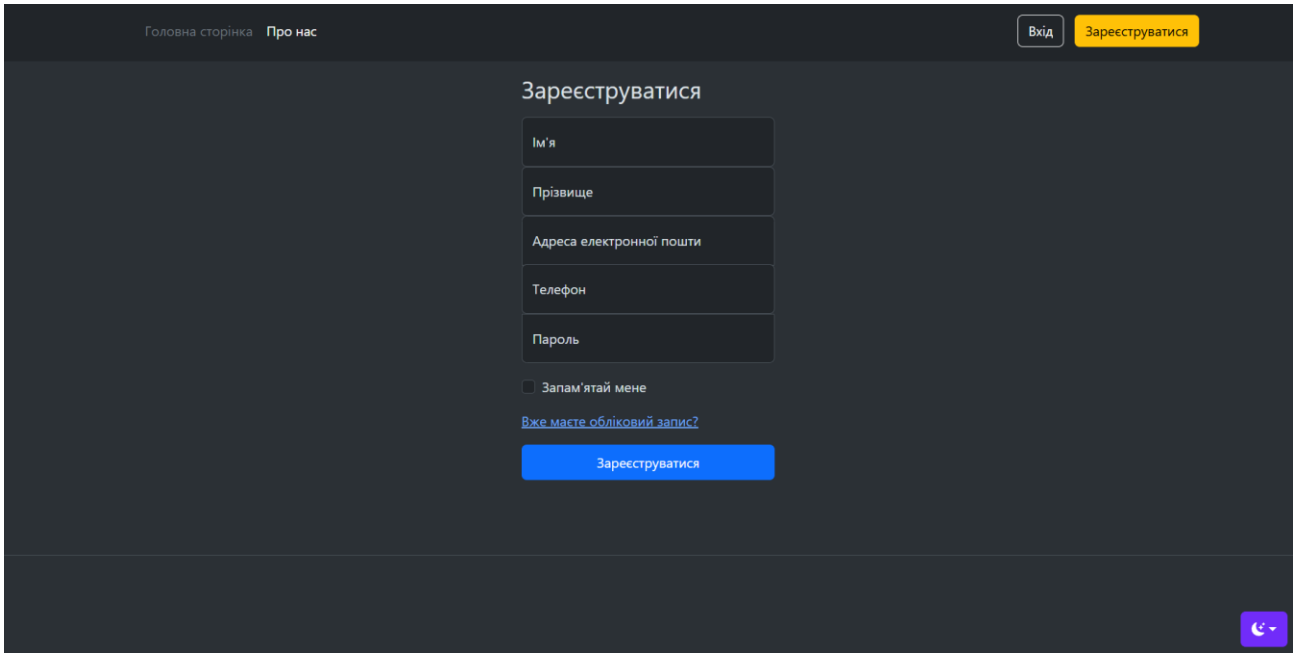


Рисунок 3.7 – Сторінка реєстрації користувача

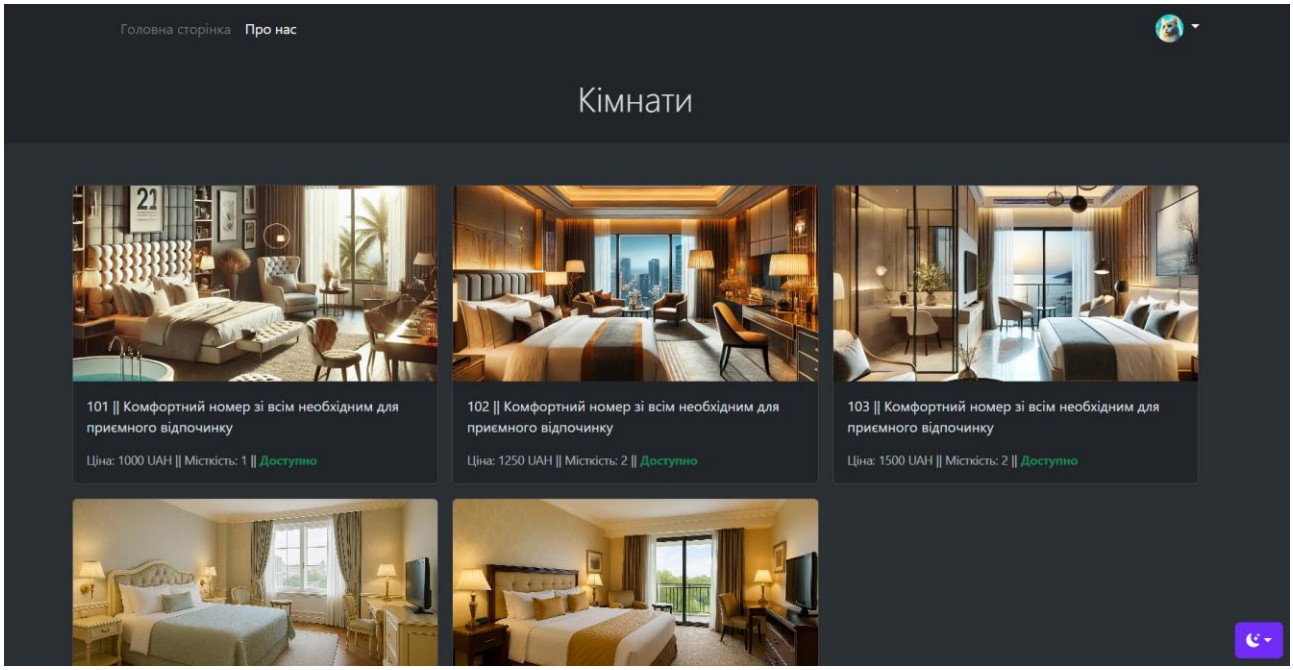


Рисунок 3.8 – Перелік доступних номерів після авторизації

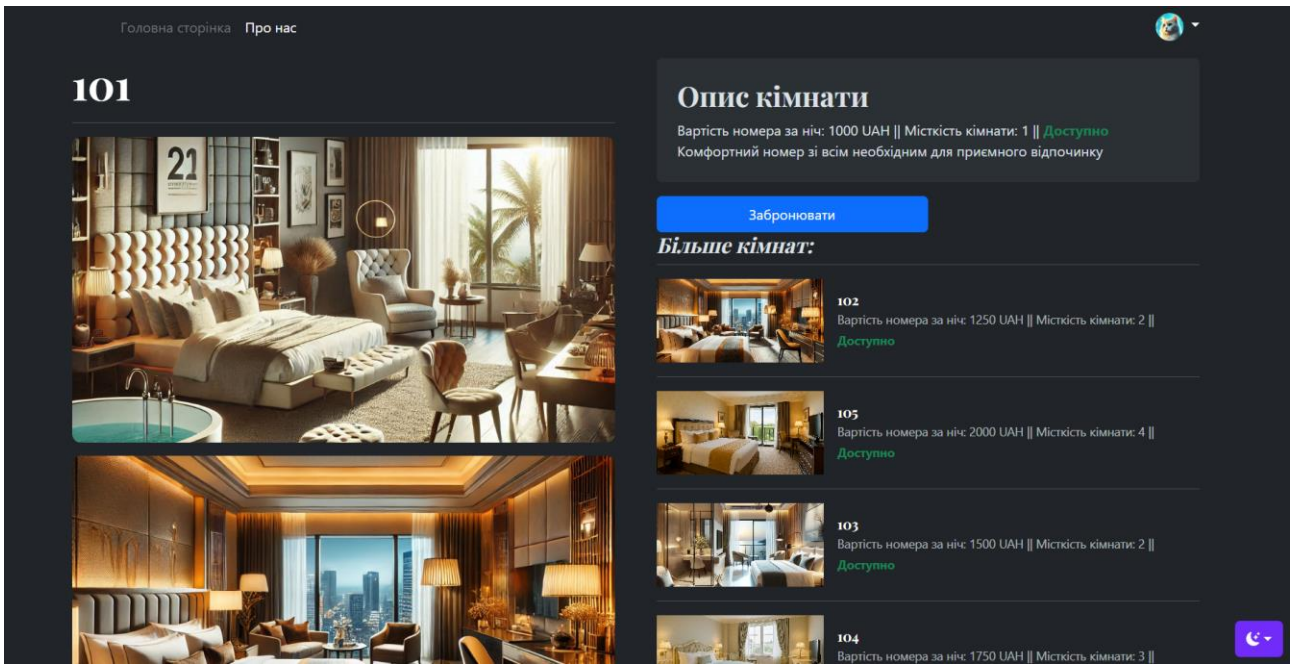


Рисунок 3.9 – Сторінка з деталями конкретного номера та кнопкою бронювання

Після натискання на кнопку відкривається форма бронювання (рис. 3.10), де користувач повинен вибрати дату заселення та дату виїзду. Після підтвердження інформації бронювання створюється, і система автоматично перенаправляє користувача до особистого кабінету зі списком його бронювань (рис. 3.11).

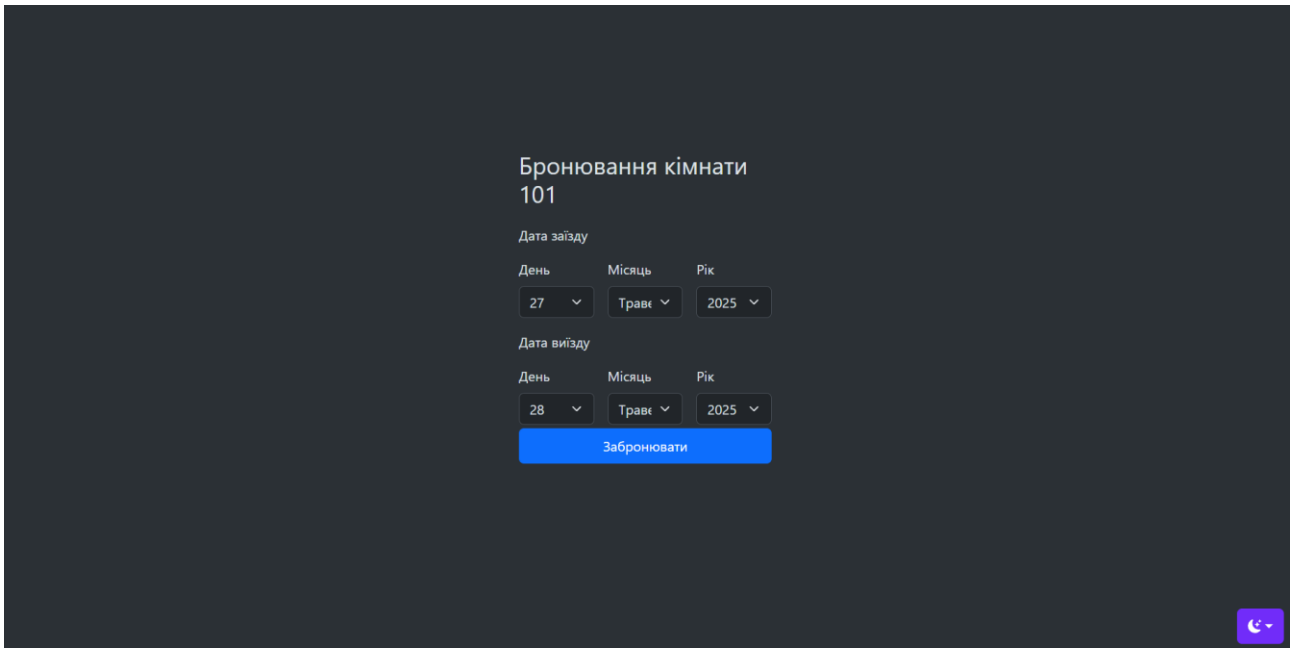


Рисунок 3.10 – Форма створення бронювання

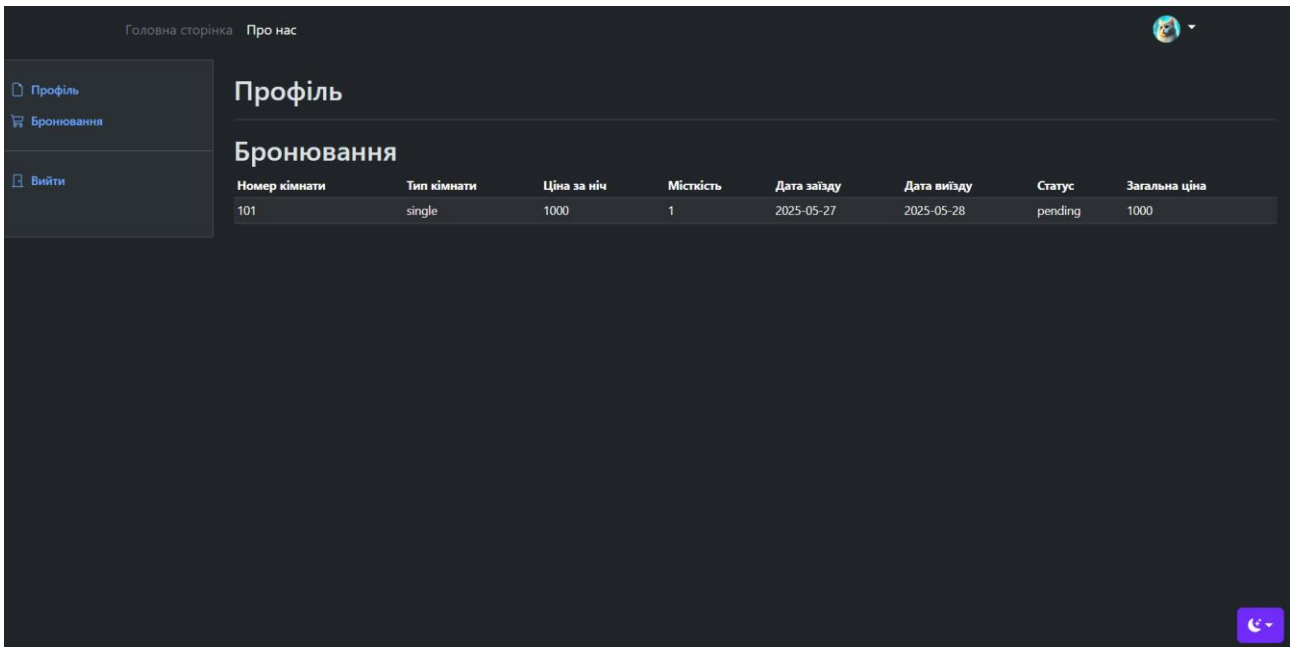


Рисунок 3.11 – Особистий кабінет користувача зі списком бронювань

Адміністратор після входу до системи отримує доступ до панелі керування (рис.3.12).

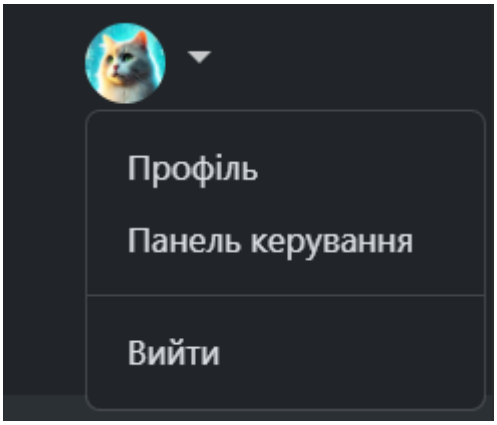


Рисунок 3.12 – Панель керування після входу адміністратора

Тут він може керувати списком готельних номерів – додавати нові, редагувати існуючі або видаляти їх (рис. 3.13).

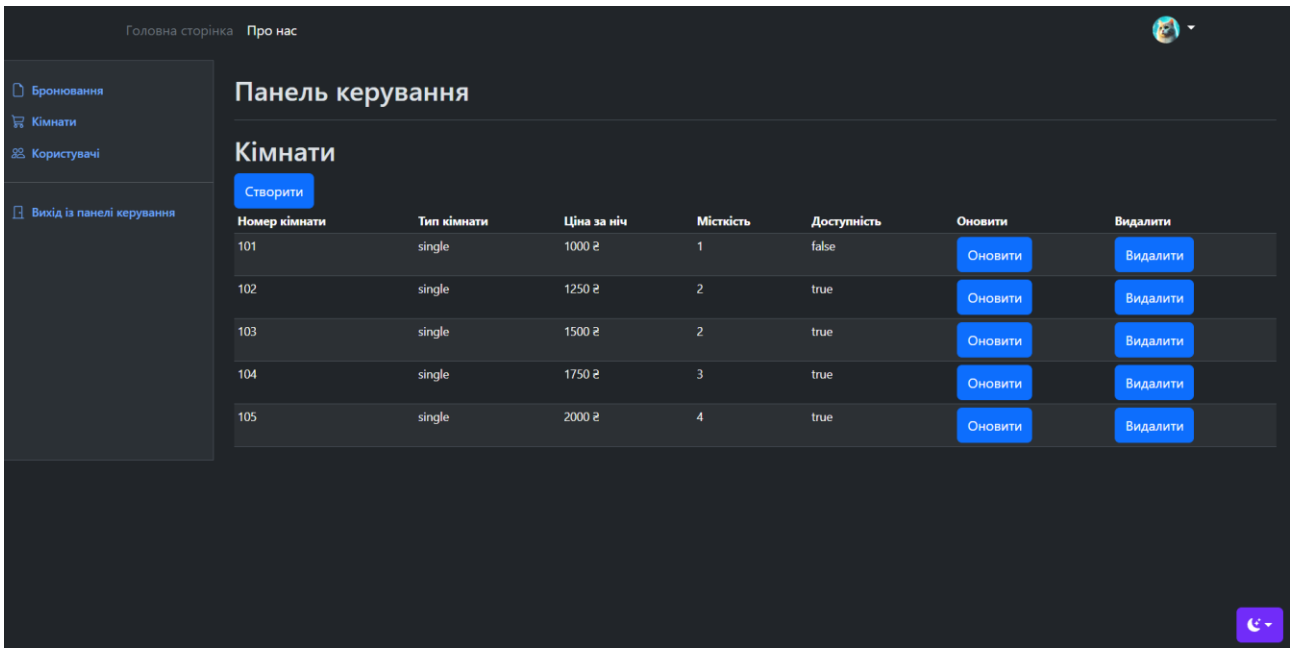


Рисунок 3.13 – Керування списком готельних номерів

Також доступні сторінки перегляду усіх бронювань (рис. 3.14) та списку зареєстрованих користувачів (рис. 3.15).

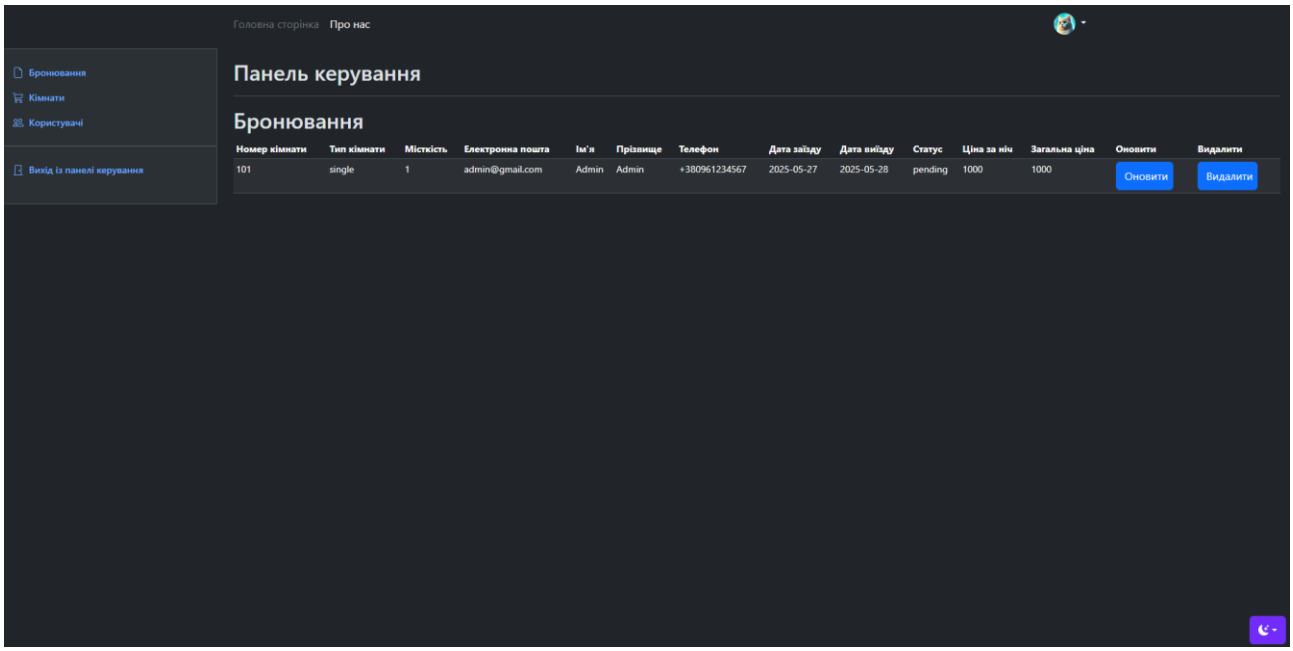


Рисунок 3.14 – Перегляд усіх бронювань адміністратором

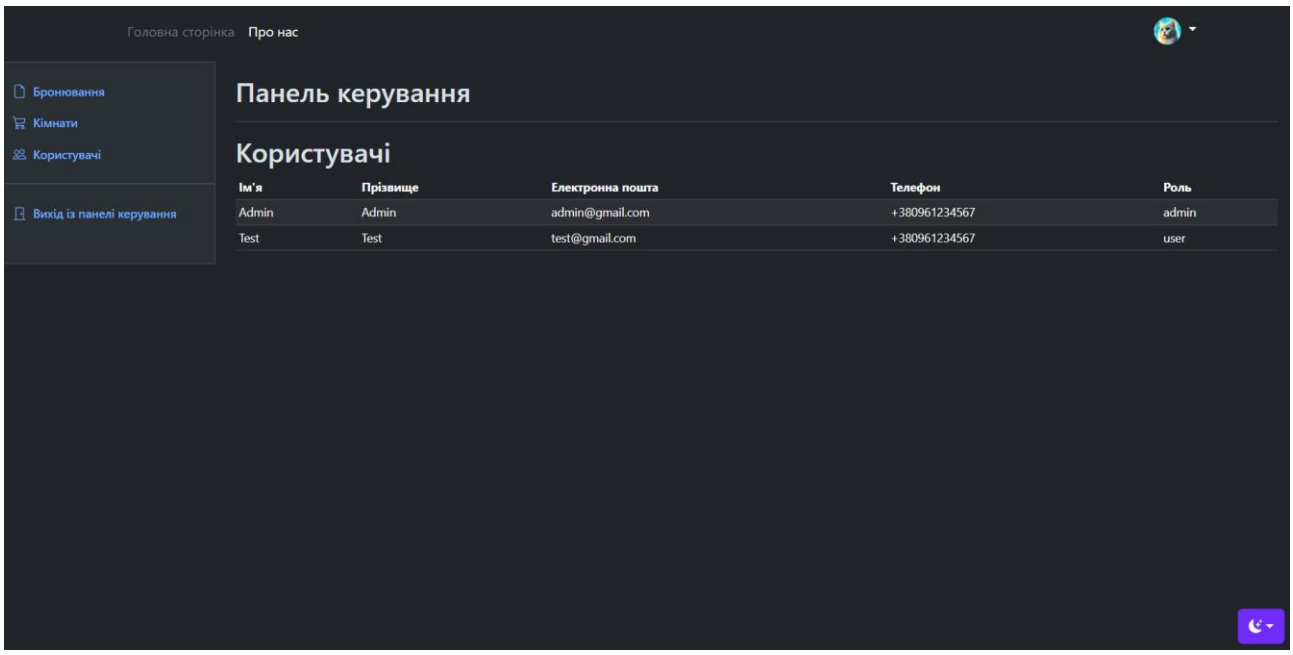


Рисунок 3.15 – Список зареєстрованих користувачів

Діаграма взаємодії користувача на рисунку 3.16.

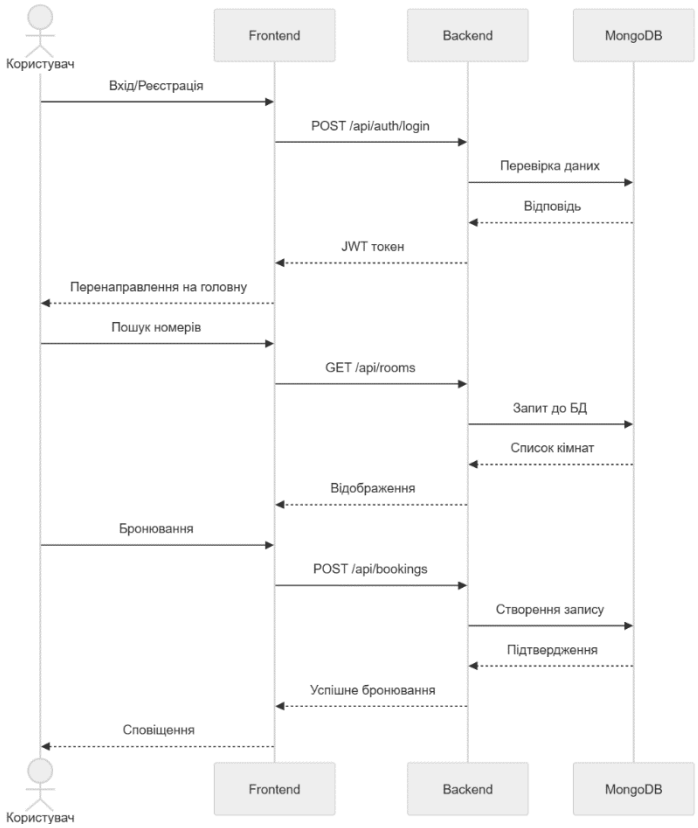


Рисунок 3.16 – Діаграма взаємодії користувача

ВИСНОВКИ

У ході виконання дипломної роботи було розроблено веб-сервіс для бронювання готельних номерів, який інтегрує сучасні технології та відповідає вимогам ефективності, безпеки та зручності. Головним результатом стало створення повноцінної системи, що автоматизує процес бронювання, забезпечує зручний інтерфейс для користувачів та надає адміністраторам інструменти для управління номерами та бронюваннями. Використання TypeScript і Node.js дозволило побудувати масштабований бекенд із статичною типізацією, що зменшило кількість критичних помилок на етапі розробки. Серверний рендеринг через EJS та адаптивний дизайн на Bootstrap забезпечили швидке завантаження сторінок і зручну взаємодію на будь-яких пристроях.

Інтеграція MongoDB як основної бази даних забезпечила гнучкість у роботі з динамічними структурами, а використання JWT та bcrypt для аутентифікації додало системі рівень безпеки, необхідний для обробки конфіденційних даних. Завдяки Docker та Nginx сервіс може бути легко розгорнутий у хмарному середовищі, що робить його доступним для малих і середніх готелів із мінімальними витратами.

Розроблений сервіс не лише спрощує процес бронювання для клієнтів, але й дозволяє готелям зменшити залежність від зовнішніх агрегаторів, збільшуючи пряму монетизацію. Перспективи розвитку включають впровадження онлайн-оплат, AI-рекомендацій та мобільного додатку, що розширить функціонал і покращить персоналізацію.

Таким чином, робота довела ефективність обраного підходу та технологій для вирішення актуальних завдань готельної індустрії. Розроблений продукт є готовим рішенням для цифрової трансформації традиційних процесів, що сприятиме підвищенню конкурентоспроможності готелів на ринку.

ПЕРЕЛІК ПОСИЛАНЬ

1. Hotel Booking Market Size, Share, Trends, Growth And Forecast. Verified Market Research. URL : https://www.verifiedmarketresearch.com/product/hotel-booking-market/ (дата звернення : 18.05.2025).

2. 87 Facts that Tell the Story of Online Travel Booking in 2025. Navan. URL : https://navan.com/blog/insights-trends/online-travel-booking-statistics (дата звернення : 18.05.2025).

3. Get Inspired with Global Travel Trends. American Express. URL : https://www.americanexpress.com/en-us/travel/discover/get-inspired/global-travel-trends (дата звернення : 18.05.2025).

4. Hotel Industry Statistics. SiteMinder. URL : https://www.siteminder.com/r/hotel-industry-statistics/ (дата звернення : 18.05.2025).

5. What Is a Hotel Reservation System? Cloudbeds. URL : https://www.cloudbeds.com/articles/hotel-reservation-system/ (дата звернення : 18.05.2025).

6. Advantages of Using an Online Booking System for Your Property. Hotelogix Blog. URL : https://blog.hotelogix.com/advantages-using-online-booking-system-property/ (дата звернення : 18.05.2025).

7. Top Hospitality Trends. Swiss Hotel Management School (SHMS). URL : https://www.shms.com/en/news/top-hospitality-trends/ (дата звернення : 18.05.2025).

8. Hotel Statistics 2024: Trends & Insights. Event Temple. URL : https://www.eventtemple.com/blog/hotel-statistics-2024-trends-insights (дата звернення : 18.05.2025).

9. Online Travel Booking Statistics & Trends. Navan. URL : https://navan.com/blog/insights-trends/online-travel-booking-statistics (дата звернення : 18.05.2025).

10. Hotel Reservation System Explained. Cloudbeds. URL : https://www.cloudbeds.com/articles/hotel-reservation-system/ (дата звернення : 18.05.2025).

11. Hotel Statistics 2024: Trends & Insights. Event Temple. URL : https://www.eventtemple.com/blog/hotel-statistics-2024-trends-insights (дата звернення : 18.05.2025).

ДОДАТОК А

Програмний код серверної частини системи

Файл server.ts

```
import express from 'express';
import dotenv from 'dotenv';
import path from 'path';
import {mongoDBConnection} from "./config/mongoDBConnection";
import userRoutes from "./routes/userRoutes";
import authRoutes from "./routes/authRoutes";
import roomRoutes from "./routes/roomRoutes";
import pages from "./routes/pagesRoutes/pages"
import bookingRoutes from "./routes/bookingRoutes";
import errorHandler from "./middlewares/errorHandler";
import cookieParser from "cookie-parser";
import methodOverride from "method-override";

dotenv.config();

const app = express();
const PORT = process.env.PORT || 3000;

// Middlewares
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use(cookieParser());
app.use(methodOverride('_method'));

app.locals.FRONTEND_URL_AND_PORT = process.env.DOMAIN_URL ||
'http://localhost:3000';

// Static files
app.use('/uploads', express.static('uploads'));
app.use('/upload', express.static('upload'))
app.use(express.static('public'));

// View engine setup
app.set('views', path.join('src', 'views'));
app.set('view engine', 'ejs');
```

```
// Pages Routes
app.use('/', pages)

// API Routes
app.use('/api/auth', authRoutes);
app.use('/api/users', userRoutes);
app.use('/api/rooms', roomRoutes);
app.use('/api/bookings', bookingRoutes);

// Error handling middleware
app.use(errorHandler);

// Connect to the database and start the server
const startServer = async () => {
   try {
      await mongoDBConnection();
      app.listen(PORT, () => {
         console.log(`Server is running on http://localhost:${PORT}`);
      });
   } catch (error) {
      console.error('Failed to start server:', error);
      process.exit(1); // Exit the process with failure
   }
};

// Start the server
startServer();
```

Файл authRoutes.ts

```
import express from 'express';
import {validateBody} from '../middlewares/validateBody';
import {registerUserSchema, loginUserSchema, updateUserProfileSchema} from
'../schemas/userSchemas';
import {authorizeRole} from '../middlewares/authorizeRole';
import {Roles} from '../constants/roles';
import {registerUser, loginUser, logoutUser, userProfile, updateUserProfile} from
'../controllers/authController';

const router = express.Router();

//@desc User registration
//@route POST /api/auth/register
```

```
//@access Public
router.post('/register', validateBody(registerUserSchema), registerUser)

//@desc User login
//@route POST /api/auth/login
//@access Public
router.post('/login', validateBody(loginUserSchema), loginUser);

//@desc User Logout
//@route POST /api/auth/logout
//@access Public
router.post('/logout', logoutUser);

//@desc Get a profile
//@route GET /api/auth/profile
//@access Admin, Staff, User
router.get('/profile', authorizeRole(Roles.ADMIN, Roles.STAFF, Roles.USER),
userProfile);

//@desc Update user profile
//@route PATCH /api/auth/profile
//@access Admin, Staff, User
router.patch('/profile', authorizeRole(Roles.ADMIN, Roles.STAFF, Roles.USER),
validateBody(updateUserProfileSchema), updateUserProfile)

export default router;
```

Файл bookingRoutes.ts

```
import express from 'express';
import { validateBody } from '../middlewares/validateBody';
import { createBookingSchema, updateBookingSchema} from
'../schemas/bookingSchemas';
import { authorizeRole } from '../middlewares/authorizeRole';
import { Roles } from '../constants/roles';
import { createBooking, getBookings, getBookingById, getMyBookings,
updateBooking, deleteBooking } from '../controllers/bookingController';

const router = express.Router();

//@desc Get bookings
//@route GET /api/bookings
//@access Admin, Staff
router.get('/', authorizeRole(Roles.ADMIN, Roles.STAFF), getBookings);
```

```
//@desc The user get his bookings
//@route GET /api/bookings/my
//@access Admin, Staff, User
router.get('/my', authorizeRole(Roles.ADMIN, Roles.STAFF, Roles.USER),
getMyBookings);

//@desc Get booking by id
//@route GET /api/bookings/:id
//@access Admin, Staff
router.get('/:id', authorizeRole(Roles.ADMIN, Roles.STAFF), getBookingById);

//@desc Create booking
//@route POST /api/bookings
//@access Admin, Staff, User
router.post('/', authorizeRole(Roles.ADMIN, Roles.STAFF, Roles.USER),
validateBody(createBookingSchema), createBooking);

//@desc Delete booking
//@route DELETE /api/bookings/:id
//@access Admin, Staff
router.delete('/:id', authorizeRole(Roles.ADMIN, Roles.STAFF), deleteBooking);

//@desc Update booking
//@route PATCH /api/bookings/:id
//@access Admin, Staff
router.patch('/:id', authorizeRole(Roles.ADMIN, Roles.STAFF),
validateBody(updateBookingSchema), updateBooking);

export default router;
```

Файл roomRoutes.ts

```
import express from 'express';
import {getRooms, getRoomById, createRoom, updateRoom, deleteRoom} from
'../controllers/roomController';
import {validateBody} from '../middlewares/validateBody';
import {createRoomSchema, updateRoomSchema} from '../schemas/roomSchemas';
import {authorizeRole} from "../middlewares/authorizeRole";
import {Roles} from "../constants/roles";
import { multerRoomPhotosConfig } from '../config/multerRoomPhotosConfig';

const router = express.Router();
```

```
//@desc Get rooms
//@route GET /api/rooms
//@access Public
router.get('/', getRooms);

//@desc Get room by id
//@route GET /api/rooms/:id
//@access Public
router.get('/:id', getRoomById);

//@desc Create a room
//@route POST /api/rooms
//@access Admin, Staff
router.post('/', authorizeRole(Roles.ADMIN, Roles.STAFF),
validateBody(createRoomSchema), multerRoomPhotosConfig.array('photos'),
createRoom);

//@desc Update a room
//@route PATCH /api/rooms/:id
//@access Admin, Staff
router.patch('/:id', authorizeRole(Roles.ADMIN, Roles.STAFF),
validateBody(updateRoomSchema), multerRoomPhotosConfig.array('photos'),
updateRoom)

//@desc Delete a room
//@route DELETE /api/rooms/:id
//@access Admin, Staff
router.delete('/:id', authorizeRole(Roles.ADMIN, Roles.STAFF), deleteRoom);

export default router;
```

Файл userRoutes.ts

```
import express from 'express';
import {getUsers, getUserById, updateUserById} from '../controllers/userController';
import {authorizeRole} from "../middlewares/authorizeRole";
import {Roles} from "../constants/roles";
import {validateBody} from '../middlewares/validateBody';
import {updateUserByIdSchema} from '../schemas/userSchemas'

const router = express.Router();

//@desc Get users
//@route GET /api/users
```

```
//@access Admin
router.get('/', authorizeRole(Roles.ADMIN), getUsers);

//@desc Get user by id
//@route GET /api/users/:id
//@access Admin
router.get('/:id', authorizeRole(Roles.ADMIN), getUserById);

//@desc Update user by id (role, etc.)
//@route PATCH /api/users/:id
//@access Admin
router.patch('/:id', authorizeRole(Roles.ADMIN),
validateBody(updateUserByIdSchema), updateUserById);

export default router;
```

Файл pages.ts

```
import express from 'express';
import { setUserRoleLocalsFromCookie } from
'../../middlewares/setUserRoleLocalsFromCookie';
import {
   loginPage,
   registerPage,
   mainPage,
   roomPage,
   aboutPage,
   profilePage,
   profileBookingsPage,
   createBooking,
   staffBookingsPage,
   staffRoomsPage,
   staffUsersPage,
   roomUpdatePage,
   roomCreatePage,
   bookingUpdatePage
} from '../../controllers/pagesControllers/pagesController';
import { authorizeRole } from '../../middlewares/authorizeRole';
import { Roles } from '../../constants/roles';

const router = express.Router();

router.get('/', setUserRoleLocalsFromCookie, mainPage);
```

```
router.get('/room/create', setUserRoleLocalsFromCookie,
authorizeRole(Roles.ADMIN, Roles.STAFF), roomCreatePage);

router.get('/room/:id', setUserRoleLocalsFromCookie, roomPage);

router.get('/room/update/:id', setUserRoleLocalsFromCookie,
authorizeRole(Roles.ADMIN, Roles.STAFF), roomUpdatePage);

router.get('/about', setUserRoleLocalsFromCookie, aboutPage);

router.get('/profile', setUserRoleLocalsFromCookie, authorizeRole(Roles.ADMIN,
Roles.STAFF, Roles.USER), profilePage);

router.get('/profile/bookings', setUserRoleLocalsFromCookie,
authorizeRole(Roles.ADMIN, Roles.STAFF, Roles.USER), profileBookingsPage)

router.get('/staff/bookings', setUserRoleLocalsFromCookie,
authorizeRole(Roles.ADMIN, Roles.STAFF), staffBookingsPage);

router.get('/staff/rooms', setUserRoleLocalsFromCookie,
authorizeRole(Roles.ADMIN, Roles.STAFF), staffRoomsPage);

router.get('/staff/users', setUserRoleLocalsFromCookie,
authorizeRole(Roles.ADMIN, Roles.STAFF), staffUsersPage);

router.get('/booking/:id', setUserRoleLocalsFromCookie,
authorizeRole(Roles.ADMIN, Roles.STAFF, Roles.USER), createBooking)

router.get('/booking/update/:id', setUserRoleLocalsFromCookie,
authorizeRole(Roles.ADMIN, Roles.STAFF), bookingUpdatePage);

router.get('/login', setUserRoleLocalsFromCookie, loginPage);

router.get('/register', setUserRoleLocalsFromCookie, registerPage);

export default router;
```

Файл authController.ts

```
import {Request, Response} from 'express';
import bcrypt from 'bcrypt';
import jwt from 'jsonwebtoken';
import {serialize} from 'cookie';
import { v4 as uuidv4 } from 'uuid';
```

```
import {User} from '../models/userModel';

//@desc User registration
const registerUser = async (req: Request, res: Response) => {
    const {firstName, lastName, email, phone, password } = req.body;
    const user = await User.findOne({email});
    if (user) {
        res.status(400).json({message: 'User already exists'});
        return;
    }

    const hashedPassword = await bcrypt.hash(password, 10);

    const newUser = await User.create({
        id: uuidv4(),
        firstName,
        lastName,
        email,
        phone,
        password: hashedPassword,
        role: 'user'
    });

    if (!newUser) {
        res.status(400).json({message: 'Failed to create user'});
        return;
    }

    const jwtSecret = process.env.JWT_SECRET || '';
    if (!jwtSecret) {
        throw new Error('JWT secret is not defined in environment variables');
    }
    const accessToken = jwt.sign(
        {
            id: newUser.id,
            role: newUser.role
        },
        jwtSecret,
        {expiresIn: '30d'}
    );

    const serializeToken = serialize('accessToken', accessToken, {
        httpOnly: true,
        secure: false,
        sameSite: 'lax',
```

```
    maxAge: 30 * 24 * 60 * 60,
    path: '/'
  });

  res.status(201).setHeader('Set-Cookie', serializeToken).redirect('/');

}

//@desc User login
const loginUser = async (req: Request, res: Response) => {
  const {email, password} = req.body;
  const user = await User.findOne({email});
  if (!user) {
    res.status(400).json({message: 'Invalid email or account does not exist'});
    return;
  }
  const isPasswordValid = await bcrypt.compare(password, user.password);
  if (!isPasswordValid) {
    res.status(400).json({message: 'Invalid password'});
    return;
  }
  const jwtSecret = process.env.JWT_SECRET || '';
  if (!jwtSecret) {
    throw new Error('JWT secret is not defined in environment variables');
  }
  const accessToken = jwt.sign(
    {
      id: user.id,
      role: user.role
    },
    jwtSecret,
    {expiresIn: '30d'}
  );

  const serializeToken = serialize('accessToken', accessToken, {
    httpOnly: true,
    secure: false,
    sameSite: 'lax',
    maxAge: 30 * 24 * 60 * 60,
    path: '/'
  });

  res.setHeader('Set-Cookie', serializeToken).redirect('/');

}
```

```
//@desc User Logout
const logoutUser = async (req: Request, res: Response) => {
   res.clearCookie('accessToken').status(200).redirect('/login');
}

//@desc Get a profile
const userProfile = async (req: Request, res: Response) => {
   // @ts-ignore
   const user = req.user;
   res.status(200).json({message: `${user.id} ${user.role} profile`});
}

//@desc Update user profile
const updateUserProfile = async (req: Request, res: Response) => {
   // @ts-ignore
   const userId = req.user.id;
   const data = req.body;
   if (data.password) {
      data.password = await bcrypt.hash(data.password, 10);
   }
   await User.updateOne({id: userId}, data);
   res.status(200).json({message: 'Profile updated successfully'});

}

export {registerUser, loginUser, logoutUser, userProfile, updateUserProfile};
```

Файл bookingController.ts

```
import {Request, Response} from 'express';
import {Booking} from '../models/bookingModel';
import {v4 as uuidv4} from 'uuid';
import { Room } from '../models/roomModel';

//@desc Get bookings
const getBookings = async (req: Request, res: Response) => {
   const bookings = await Booking.find();
   if (!bookings) {
      res.status(404).json({message: 'Bookings not found'});
      return;
   }
   res.status(200).json(bookings);
};
```

```
//@desc Get booking by id
const getBookingById = async (req: Request, res: Response) => {
    const id = req.params.id;
    const booking = await Booking.findOne({id});
    if (!booking) {
        res.status(404).json({message: 'Booking not found'});
        return;
    }
    res.status(200).json(booking);
}

//@desc The user get his bookings
const getMyBookings = async (req: Request, res: Response) => {
    // @ts-ignore
    const userId = req.user.id;
    const bookings = await Booking.find({userId});
    if (!bookings) {
        res.status(404).json({message: 'Bookings not found'});
        return;
    }
    res.status(200).json(bookings);
};

//@desc Create booking
const createBooking = async (req: Request, res: Response) => {
    const {roomId, startDate, endDate} = req.body;
    // @ts-ignore
    const userId = req.user.id;
    const status = 'pending';
    const id = uuidv4();

    const msPerDay = 1000 * 60 * 60 * 24;
    const diffInMs = new Date(endDate).getTime() - new Date(startDate).getTime();
    const diffInDays = Math.floor(diffInMs / msPerDay);

    const room = await Room.findOne({id: roomId});
    if (!room) {
        res.status(404).json({message: 'Room not found for booking'});
        return;
    }

    if (new Date(startDate) > new Date(endDate)) {
        res.status(400).json({message: 'Start date must be before end date'});
        return;
```

```
    }

    const bookings = await Booking.find({roomId});

    // Check if the new booking conflicts with any existing bookings
    const isDoubleBooking = bookings.some(booking => new Date(startDate) <= new
Date(booking.endDate) && new Date(booking.startDate) <= new Date(endDate));
    if (isDoubleBooking) {
        res.status(400).json({message: 'Room is already booked for the selected dates'});
        return;
    }

    // Check if room is available
    if (!room.isAvailable) {
        res.status(400).json({message: 'Room is not available'});
        return;
    }

    const totalPrice = room.pricePerNight * diffInDays;

    const booking = await Booking.create({
        id,
        userId,
        roomId,
        startDate,
        endDate,
        status,
        totalPrice
    });
    if (!booking) {
        res.status(500).json({message: 'Error creating booking'});
        return;
    }

    // Update room availability
    await Room.updateOne({id: roomId}, {isAvailable: false});

    res.status(201).redirect('/profile/bookings');
};

const deleteBooking = async (req: Request, res: Response) => {
    const bookingId = req.params.id;
    const booking = await Booking.deleteOne({id: bookingId});
    if (!booking) {
        res.status(404).json({message: 'Booking not found'});
```

```
        return;
    }
    res.status(200).redirect('/staff/bookings');
}

const updateBooking = async (req: Request, res: Response) => {
    const bookingId = req.params.id;
    const {status} = req.body;
    await Booking.updateOne({id: bookingId}, {status});
    res.status(200).redirect('/staff/bookings');

}

export { createBooking, getBookings, getBookingById, getMyBookings,
updateBooking, deleteBooking };
```

Файл roomController.ts

```
import {Request, Response} from 'express';
import {v4 as uuidv4} from 'uuid';
import {Room} from '../models/roomModel';

//@desc Get rooms
const getRooms = async (req: Request, res: Response) => {
    const rooms = await Room.find();
    res.status(200).json(rooms);
};

//@desc Get room by id
const getRoomById = async (req: Request, res: Response) => {
    const roomId = req.params.id;
    const room = await Room.findOne({id: roomId});
    if (!room) {
        res.status(404).json({message: 'Room not found'});
        return;
    }
    res.status(200).json(room);
};

//@desc Create a room
const createRoom = async (req: Request, res: Response) => {
    const data = req.body;
    const photos = req.files as Express.Multer.File[];
    const photoPaths = photos.map((file) => file.path);
```

```
    data.photoPaths = photoPaths;
    data.id = uuidv4();
    const room = await Room.create(data);
    if (!room) {
        res.status(500).json({message: 'Error creating room'});
        return;
    }
    res.status(201).redirect('/staff/rooms');
};

//@desc Update a room
const updateRoom = async (req: Request, res: Response) => {
    const roomId = req.params.id;
    const data = req.body;
    const photos = req.files as Express.Multer.File[];
    if (photos.length > 0) {
        data.photoPaths = photos.map((file) => file.path);
    }
    await Room.updateOne({id: roomId}, data);
    res.status(200).redirect('/staff/rooms');
};

//@desc Delete a room
const deleteRoom = async (req: Request, res: Response) => {
    const roomId = req.params.id;
    await Room.deleteOne({id: roomId});
    res.status(200).redirect('/staff/rooms');
};

export {getRooms, getRoomById, createRoom, updateRoom, deleteRoom};
```

Файл userController.ts

```
import {Request, Response} from 'express';
import bcrypt from 'bcrypt';
import {User} from '../models/userModel';

//@desc Get users
const getUsers = async (req: Request, res: Response) => {
    const users = await User.find().select('-password');
    res.status(200).json(users);
}

//@desc Get user by id
```

```
const getUserById = async (req: Request, res: Response) => {
    const userId = req.params.id;
    const user = await User.findOne({id: userId}).select('-password');
    if (!user) {
        res.status(404).json({message: 'User not found'});
        return;
    }
    res.status(200).json(user);
}

//@desc Update user by id (role, etc.)
const updateUserById = async (req: Request, res: Response) => {
    const userId = req.params.id;
    const data = req.body;
    if (data.password) {
        data.password = await bcrypt.hash(data.password, 10);
    }
    await User.updateOne({id: userId}, data);
    res.status(200).json({message: 'User updated successfully'});
}

export {getUsers, getUserById, updateUserById};
```

Файл pagesController.ts

```
import { Request, Response } from 'express';
import { Room } from '../../models/roomModel';
import lodash from 'lodash';
import { User } from '../../models/userModel';
import { Booking } from '../../models/bookingModel';



const mainPage = async (req: Request, res: Response) => {
    const rooms = await Room.find();
    res.render('pages/album', { rooms });
}

const roomPage = async (req: Request, res: Response) => {
    const roomId = req.params.id;
    const room = await Room.findOne({ id: roomId });
    const rooms = await Room.find();
    //@ts-ignore
```

```
    const roomRecommended = lodash.shuffle(rooms.filter((roomF) => room.id !==
roomF.id)).slice(0, 4);
    res.render('pages/room', { room, rooms: roomRecommended });
}

const roomUpdatePage = async (req: Request, res: Response) => {
    const roomId = req.params.id;
    const room = await Room.findOne({id: roomId});
    res.render('pages/update-room', {room});
}

const roomCreatePage = async (req: Request, res: Response) => {
    res.render('pages/create-room');
}

const aboutPage = async (req: Request, res: Response) => {
    res.render('pages/about');
}

const profilePage = async (req: Request, res: Response) => {
    // @ts-ignore
    const userId = req.user.id;
    const user = await User.findOne({ id: userId });
    res.render('pages/profile', { table: 'profile', profile: user });
}

const profileBookingsPage = async (req: Request, res: Response) => {
    // @ts-ignore
    const userId = req.user.id;
    const getBookings = await Booking.find({ userId });
    const getRooms = await Room.find();

    const bookings = getBookings.map((booking) => {
        const room = getRooms.find((room) => room.id === booking.roomId);
        const startDate = new Date(booking.startDate).toISOString().split('T')[0];
        const endDate = new Date(booking.endDate).toISOString().split('T')[0];
        return {
            roomNumber: room?.roomNumber,
            roomType: room?.roomType,
            pricePerNight: room?.pricePerNight,
            capacity: room?.capacity,
            checkIn: startDate,
            checkOut: endDate,
            status: booking?.status,
            totalPrice: booking?.totalPrice
```

```
        };
    });

    res.render('pages/profile', { table: 'bookings', bookings })
}

const staffBookingsPage = async (req: Request, res: Response) => {
    const getBookings = await Booking.find();
    const getRooms = await Room.find();
    const getUsers = await User.find();

    const bookings = getBookings.map((booking) => {
        const room = getRooms.find((room) => room.id === booking.roomId);
        const user = getUsers.find((user) => user.id === booking.userId);
        const startDate = new Date(booking.startDate).toISOString().split('T')[0];
        const endDate = new Date(booking.endDate).toISOString().split('T')[0];
        return {
            id: booking?.id,
            roomNumber: room?.roomNumber,
            roomType: room?.roomType,
            capacity: room?.capacity,
            email: user?.email,
            firstName: user?.firstName,
            lastName: user?.lastName,
            phone: user?.phone,
            checkIn: startDate,
            checkOut: endDate,
            status: booking?.status,
            pricePerNight: room?.pricePerNight,
            totalPrice: booking?.totalPrice
        };
    });

    res.render('pages/dashboard', { table: 'bookings', bookings });
}

const staffRoomsPage = async (req: Request, res: Response) => {
    const rooms = await Room.find();

    res.render('pages/dashboard', { table: 'rooms', rooms })
}

const staffUsersPage = async (req: Request, res: Response) => {
    const users = await User.find();
```

```
    res.render('pages/dashboard', { table: 'users', users })
}

const createBooking = async (req: Request, res: Response) => {
    const roomId = req.params.id;
    const room = await Room.findOne({ id: roomId });

    function addDays(date: Date, days: number): Date {
        const result = new Date(date);
        result.setDate(result.getDate() + days);
        return result;
    }

    const nowInUkraine = new Date().toLocaleDateString('en-GB', {
        timeZone: 'Europe/Kyiv',
        day: '2-digit',
        month: 'long',
        year: 'numeric',
    });

    const nextDayInUkraine = addDays(new Date(), 1).toLocaleDateString('en-GB', {
        timeZone: 'Europe/Kyiv',
        day: '2-digit',
        month: 'long',
        year: 'numeric',
    });

    const [dayCurrent, monthCurrent, yearCurrent] = nowInUkraine.split(' ');

    const [dayNext, monthNext, yearNext] = nextDayInUkraine.split(' ');

    res.render('pages/booking', {
        room,
        currentDate: { day: dayCurrent, month: monthCurrent, year: yearCurrent },
        nextDate: { day: dayNext, month: monthNext, year: yearNext }
    });
}

const bookingUpdatePage = async (req: Request, res: Response) => {
    const bookingId = req.params.id;
    const booking = await Booking.findOne({id: bookingId});
    res.render('pages/booking-update', {booking});
};
```

```
const loginPage = async (req: Request, res: Response) => {
   res.render('pages/sign-in');
};

const registerPage = async (req: Request, res: Response) => {
   res.render('pages/sign-up');
};

export {
   loginPage,
   registerPage,
   mainPage,
   roomPage,
   aboutPage,
   profilePage,
   profileBookingsPage,
   createBooking,
   staffBookingsPage,
   staffRoomsPage,
   staffUsersPage,
   roomUpdatePage,
   roomCreatePage,
   bookingUpdatePage
};
```

Файл mongoDBConnection.ts

```
import mongoose from "mongoose";

const mongoDBConnection = async () => {
   const mongoURI: string | undefined = process.env.MONGODB_URI;
   if (!mongoURI) {
      throw new Error("MongoDB URI is not defined in environment variables");
   }
   const connect = await mongoose.connect(mongoURI);
   console.log(`MongoDB connected: ${connect.connection.host}`);
}

export {mongoDBConnection};
```

Файл multerRoomPhotosConfig.ts

```
import fs from 'fs';
import path from 'path';
import multer, {Multer} from 'multer';

const uploadDir = path.join(process.cwd(), 'uploads', 'rooms-photos');
if (!fs.existsSync(uploadDir)) {
   fs.mkdirSync(uploadDir, { recursive: true });
}

const storage = multer.diskStorage({
   destination: (req, file, cb) => {
      cb(null, 'uploads/rooms-photos');
   },
   filename: (req, file, cb) => {
      cb(null, `${Date.now()}-${file.originalname}`);
   }
});

const fileFilter = (req: any, file: any, cb: any) => {
   const allowedMimeTypes = ['image/jpeg', 'image/png', 'image/jpg'];

   if (!allowedMimeTypes.includes(file.mimetype)) {
      cb(new Error('Invalid file type. Only JPEG, PNG and JPG are allowed.'));
      return;
   }

   if (file.size > 10 * 1024 * 1024) {
      cb(new Error('File size exceeds 10MB limit.'));
      return;
   }

   cb(null, true);
};

const multerRoomPhotosConfig: Multer = multer({
   storage,
   fileFilter,
});

export {multerRoomPhotosConfig};
```

Файл roles.ts

```
const Roles = {
```

```
    ADMIN: 'admin',
    STAFF: 'staff',
    USER: 'user'
}

export {Roles}
```

Файл authorizeRole.ts

```
import {Request, Response, NextFunction} from 'express';
import jwt from 'jsonwebtoken';


const authorizeRole = (...requiredRole: string[]) =>
    (req: Request, res: Response, next: NextFunction) => {
        const jwtSecret = process.env.JWT_SECRET || '';
        if (!jwtSecret) {
            throw new Error('JWT secret is not defined in environment variables');
        }

        const accessToken = req.cookies.accessToken;
        if (!accessToken) {
            res.status(401).json({message: 'Unauthorized'});
            return;
        }

        const payload = jwt.verify(accessToken, jwtSecret) as { id: string, role: string };

        if (!requiredRole.includes(payload.role)) {
            res.status(403).json({message: 'Access denied: insufficient role'});
            return;
        }

        // @ts-ignore
        req.user = {id: payload.id, role: payload.role};
        next();
    };

export {authorizeRole};
```

Файл errorHandler.ts

```
import express from "express";
```

```
const errorHandler = (err: any, req: express.Request, res: express.Response, next:
express.NextFunction) => {
   res.status(500).json({ message: 'Internal Server Error', error: err.message });
}

export default errorHandler;
```

Файл setUserRoleLocalsFromCookie.ts

```
import {Request, Response, NextFunction} from 'express';
import jwt from 'jsonwebtoken';

const setUserRoleLocalsFromCookie = (req: Request, res: Response, next:
NextFunction) => {
   const jwtSecret = process.env.JWT_SECRET || '';
   if (!jwtSecret) {
      throw new Error('JWT secret is not defined in environment variables');
   }

   const accessToken = req.cookies.accessToken;
   if (!accessToken) {
      res.locals.role = null;
      next();
      return;
   }

   const payload = jwt.verify(accessToken, jwtSecret) as { id: string, role: string };

   res.locals.role = payload.role;
   next();
};

export {setUserRoleLocalsFromCookie};
```

Файл validateBody.ts

```
import {Request, Response, NextFunction} from 'express';
import {ObjectSchema, ValidationErrorItem} from 'joi';

const validateBody = (schema: ObjectSchema) => (req: Request, res: Response, next:
NextFunction) => {
   const { error } = schema.validate(req.body, { abortEarly: false });
```

```
    if (error) {
        res.status(400).json({
            error: error.details.map((err: ValidationErrorItem) => err.message),
        });
        return;
    }
    next();
};

export {validateBody};
```

Файл bookingModel.ts

```
import mongoose from 'mongoose';

const bookingSchema = new mongoose.Schema({
        id: {
            type: String,
            required: [true, "ID is required"],
            unique: [true, "ID must be unique"],
        },
        userId: {
            type: String,
            required: [true, "User ID is required"],
        },
        roomId: {
            type: String,
            required: [true, "Room ID is required"],
        },
        startDate: {
            type: Date,
            required: [true, "Start date is required"],
        },
        endDate: {
            type: Date,
            required: [true, "End date is required"],
        },
        status: {
            type: String,
            required: [true, "Status is required"],
        },
        totalPrice: {
            type: Number,
```

```
        required: [true, "Total price is required"],
    },
  },
  {
    timestamps: true,
  }
);

const Booking = mongoose.model('Booking', bookingSchema);

export {Booking};
```

Файл roomModel.ts

```
import mongoose from 'mongoose';

const roomSchema = new mongoose.Schema({
    id: {
      type: String,
      required: [true, "ID is required"],
      unique: [true, "ID must be unique"],
    },
    roomNumber: {
      type: String,
      required: [true, "Room number is required"],
      unique: [true, "Room number must be unique"],
    },
    roomType: {
      type: String,
      required: [true, "Room type is required"],
    },
    description: {
      type: String,
      required: [true, "Description is required"],
    },
    pricePerNight: {
      type: Number,
      required: [true, "Price per night is required"],
    },
    capacity: {
      type: Number,
      required: [true, "Capacity is required"],
    },
    photoPaths: {
```

```
            type: [String],
            required: [true, "Photos url are required"],
            validate: {
                validator: function (arr: string[]) {
                    return Array.isArray(arr) && arr.length > 0;
                },
                message: "At least one photo url is required",
            },
        },
        isAvailable: {
            type: Boolean,
            required: [true, "Is available is required"],
        },
    },
    {
        timestamps: true,
    }
);

const Room = mongoose.model('Room', roomSchema);

export {Room};
```

Файл userModel.ts

```
import mongoose from "mongoose";

const userSchema = new mongoose.Schema({
        id: {
            type: String,
            required: [true, "ID is required"],
            unique: [true, "ID must be unique"],
        },
        firstName: {
            type: String,
            required: [true, "Name is required"],
        },
        lastName: {
            type: String,
            required: [true, "Last name is required"],
        },
        email: {
            type: String,
            required: [true, "Email is required"],
```

```
            unique: [true, "Email must be unique"],
        },
        phone: {
            type: String,
            required: [true, "Phone number is required"],
        },
        password: {
            type: String,
            required: [true, "Password is required"],
        },
        role: {
            type: String,
            required: [true, "Role is required"],
        }
    },
    {
        timestamps: true,
    }
);

const User = mongoose.model('User', userSchema);

export {User}
```

Файл bookingSchemas.ts

```
import Joi, {ObjectSchema} from "joi";

const createBookingSchema: ObjectSchema = Joi.object({
    roomId: Joi.string().max(100).required(),
    startDate: Joi.date().required(),
    endDate: Joi.date().required(),
});

const updateBookingSchema: ObjectSchema = Joi.object({
    status: Joi.string().valid("pending", "confirmed", "cancelled").required(),
});


export {createBookingSchema, updateBookingSchema};
```

Файл roomSchemas.ts

```
import Joi, {ObjectSchema} from "joi";

const createRoomSchema: ObjectSchema = Joi.object({
   roomNumber: Joi.string().max(6).required(),
   roomType: Joi.string().valid('single', 'double', 'twin', 'triple', 'quad').required(),
   description: Joi.string().max(1000).required(),
   pricePerNight: Joi.number().positive().max(1000000).required(),
   capacity: Joi.number().positive().max(10).required(),
   isAvailable: Joi.boolean().required(),
});

const updateRoomSchema: ObjectSchema = Joi.object({
   roomNumber: Joi.string().max(6),
   roomType: Joi.string().valid('single', 'double', 'twin', 'triple', 'quad'),
   description: Joi.string().max(1000),
   pricePerNight: Joi.number().positive().max(1000000),
   capacity: Joi.number().positive().max(10),
   isAvailable: Joi.boolean(),
})

export {createRoomSchema, updateRoomSchema};
```

Файл userSchemas.ts

```
import Joi, {ObjectSchema} from "joi";

const registerUserSchema: ObjectSchema = Joi.object({
   firstName: Joi.string().max(100).required(),
   lastName: Joi.string().max(100).required(),
   email: Joi.string().email().max(100).required(),
   phone: Joi.string().max(100).required(),
   password: Joi.string().min(8).max(100).required(),
});

const loginUserSchema: ObjectSchema = Joi.object({
   email: Joi.string().email().required(),
   password: Joi.string().min(8).required(),
});

const updateUserByIdSchema: ObjectSchema = Joi.object({
   firstName: Joi.string().max(100),
   lastName: Joi.string().max(100),
   email: Joi.string().email().max(100),
   phone: Joi.string().max(100),
```

```
    password: Joi.string().min(8).max(100),
    role: Joi.string().valid('admin', 'staff', 'user'),
});

const updateUserProfileSchema: ObjectSchema = Joi.object({
    firstName: Joi.string().max(100),
    lastName: Joi.string().max(100),
    email: Joi.string().email().max(100),
    phone: Joi.string().max(100),
    password: Joi.string().min(8).max(100),
});

export {registerUserSchema, loginUserSchema, updateUserByIdSchema,
updateUserProfileSchema};
```

Файл .dockerignore

```
# Node modules
node_modules

# Build output
dist

# Docker files
Dockerfile
.dockerignore

# IDE files
.idea

# Git files
.git
.gitignore
.github

# Environment files
.env
.env.*

# Tests
tests

# Other
todo.txt
```

Файл .env.example

```
PORT=3000 # The port on which the server will run
DOMAIN=it-is-hotel-room-booking.online # The domain name for the application
DOMAIN_URL=http://localhost:3000 # The full domain name for the application
EMAIL_FOR_CERTBOT=email@gmail.com # The email address for Let's Encrypt certificate notifications
MONGODB_URI=mongodb://root:password@mongo:27017 # MongoDB connection string
MONGO_INITDB_ROOT_USERNAME=root # MongoDB root username
MONGO_INITDB_ROOT_PASSWORD=password # MongoDB root password
JWT_SECRET=your_jwt_secret # Secret key for JWT signing
```

Файл .gitattributes

```
# Force LF for all text files
* text=auto eol=lf

# Enforce LF for specific known file types
*.sh text eol=lf
*.js text eol=lf
*.ts text eol=lf
*.json text eol=lf
*.md text eol=lf
```

Файл .gitignore

```
# Node modules
node_modules

# Build output
/dist

# IDE files
.idea

# Environment files
.env

# Other
todo.txt
```

Файл docker-compose.yml

```
services:
  app:
    build: .
    container_name: app
#    undocument if you want to access the app from outside on port 3000
    ports:
      - "3000:3000"
    env_file:
      - .env
    volumes:
      - uploads:/app/uploads
    depends_on:
      - mongo
    networks:
      - app-network

  nginx:
    build: ./nginx
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - certbot_letsencrypt:/etc/letsencrypt
      - certbot_www:/var/www/certbot
    depends_on:
      - app
    networks:
      - app-network

  certbot:
    image: certbot/certbot:latest
    volumes:
      - certbot_letsencrypt:/etc/letsencrypt
      - certbot_www:/var/www/certbot
    command: certonly --webroot --webroot-path=/var/www/certbot --email
${EMAIL_FOR_CERTBOT} --agree-tos --no-eff-email -d ${DOMAIN} --keep-
until-expiring

  mongo:
    image: mongo
    container_name: mongo
```

```
#    undocument if you want to access the database from outside on port 27018
    ports:
      - "27018:27017"
    environment:
      MONGO_INITDB_ROOT_USERNAME:
${MONGO_INITDB_ROOT_USERNAME}
      MONGO_INITDB_ROOT_PASSWORD:
${MONGO_INITDB_ROOT_PASSWORD}
    volumes:
      - mongo_data:/data/db
    networks:
      - app-network

networks:
  app-network:
    driver: bridge

volumes:
  certbot_letsencrypt:
  certbot_www:
  mongo_data:
  uploads:
```

Файл Dockerfile

```
# Dockerfile
FROM node:24-alpine

WORKDIR /app

COPY package*.json ./
RUN npm install

COPY . .

RUN npm run build

EXPOSE 3000

CMD ["node", "dist/server.js"]
```

Файл package.json

```
{
 "name": "hotel-room-booking",
 "version": "1.0.0",
 "description": "init",
 "main": "index.js",
 "scripts": {
  "dev": "docker compose up app --build --force-recreate",
  "build": "tsc",
  "start": "node dist/server.js"
 },
 "author": "",
 "license": "ISC",
 "dependencies": {
  "axios": "^1.9.0",
  "bcrypt": "^5.1.1",
  "cookie": "^1.0.2",
  "cookie-parser": "^1.4.7",
  "dotenv": "^16.5.0",
  "ejs": "^3.1.10",
  "express": "^5.1.0",
  "hotel-room-booking": "file:",
  "joi": "^17.13.3",
  "jsonwebtoken": "^9.0.2",
  "lodash": "^4.17.21",
  "method-override": "^3.0.0",
  "mongoose": "^8.14.1",
  "multer": "^1.4.5-lts.2",
  "uuid": "^11.1.0"
 },
 "devDependencies": {
  "@types/bcrypt": "^5.0.2",
  "@types/cookie-parser": "^1.4.8",
  "@types/ejs": "^3.1.5",
  "@types/express": "^5.0.1",
  "@types/jsonwebtoken": "^9.0.9",
  "@types/lodash": "^4.17.16",
  "@types/method-override": "^3.0.0",
  "@types/multer": "^1.4.12",
  "@types/node": "^22.14.1",
  "ts-node": "^10.9.2",
  "ts-node-dev": "^2.0.0",
  "typescript": "^5.8.3"
 }
}
```

Файл README.md

1. Create env based on env.example and fill in the fields
2. Change server_name in nginx/templates/http.conf and nginx/templates/https.conf on your domain
3. Change the path to your CERT_FILE and KEY_FILE in nginx/entrypoint.sh to match your domain
4. Copy project on server
5. run command: docker compose up nginx --build -d
6. if (!certificate) {
   1. run command: docker compose up certbot --build
   2. run command: docker compose down nginx
   3. run command: docker compose up nginx --build -d
   }

Файл http.conf

```
server {
    listen 80;
    server_name it-is-hotel-room-booking.online;

    client_max_body_size 10M;

    # This is for processing Certbot requests
    location /.well-known/acme-challenge/ {
        root /var/www/certbot;
    }

    location / {
            # TEMPORARILY serve your app via HTTP (no redirect)
            proxy_pass http://app:3000;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

Файл https.conf

```
server {
    listen 80;
```

```
    server_name it-is-hotel-room-booking.online;

    client_max_body_size 10M;

    location /.well-known/acme-challenge/ {
        root /var/www/certbot;
    }

    location / {
        return 301 https://$host$request_uri;
    }
}

server {
    listen 443 ssl;
    server_name it-is-hotel-room-booking.online;

    client_max_body_size 10M;

    ssl_certificate /etc/letsencrypt/live/it-is-hotel-room-booking.online/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/it-is-hotel-room-
booking.online/privkey.pem;

    location / {
        proxy_pass http://app:3000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

Файл nginx/Dockerfile

```
FROM nginx:latest

# Copy entrypoint.sh to the container
COPY entrypoint.sh /entrypoint.sh

# We grant execution rights
RUN chmod +x /entrypoint.sh

# Copy configs to container
COPY ./templates /etc/nginx/templates
```

```
# Specify the script as entrypoint
ENTRYPOINT ["/entrypoint.sh"]
```

Файл entrypoint.sh

```
#!/bin/bash

CERT_FILE="/etc/letsencrypt/live/it-is-hotel-room-booking.online/fullchain.pem"
KEY_FILE="/etc/letsencrypt/live/it-is-hotel-room-booking.online/privkey.pem"
NGINX_CONF_DIR="/etc/nginx/conf.d"

if [[ ! -f "$CERT_FILE" || ! -f "$KEY_FILE" ]]; then
    echo "Certificate not found, switching to HTTP..."
    cp /etc/nginx/templates/http.conf "$NGINX_CONF_DIR/default.conf"
else
    echo "Certificate found, enable HTTPS..."
    cp /etc/nginx/templates/https.conf "$NGINX_CONF_DIR/default.conf"
fi

exec nginx -g "daemon off;"
```