МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Центр заочної, дистанційної та вечірньої форм навчання
Кафедра інформаційних технологій

«До захисту допущено»
В.о. завідувача кафедри

_____Світлана ВАЩЕНКО

____ _____ 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 «Комп'ютерні науки»

освітньо-наукової програми «Інформаційні технології проектування»

на тему: Web-орієнтована система аналізу та бронювання номерів в готелі

Здобувача групи ІТ.мз-22с. Гаджибейлі Ізмаїла Фуад Огли
(шифр групи) (прізвище, ім'я, по батькові)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Гаджибейлі Ізмаїл Фуад Огли
(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник доцент кафедри інформаційних технологій,
доцент, к.т.н. Анна НЕНЯ _____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Суми – 2024

Сумський державний університет
Центр заочної, дистанційної та вечірньої форм навчання
Кафедра інформаційних технологій
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
В.о. зав. кафедри ІТ

_____
С. М. Ващенко
«___» _____ 2023 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра студентові
Гаджибейлі Ізмаїл Фуад Огли
(прізвище, ім'я, по батькові)

1 Тема проекту Web-орієнтована система аналізу та бронювання номерів в готелі затверджена наказом по університету від «05» вересня 2023 р. № 0465-VI

2 Термін здачі студентом закінченого проекту «10» січня 2024 р.

3 Вхідні дані до проекту результати обговорення теми з керівником

4 Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) аналіз предметної області, постановка задачі, методи дослідження, проектування web-орієнтованої системи аналізу та бронювання номерів в готелі, проектування та розробка web-системи

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) актуальність проблеми, мета проетку, постановка задачі, функціональні вимоги до проекту, контекстна діаграма IDEF0, декомпозиція IDEF0, схема web-орієнтованої системи, діаграма варіантів використання, фізична модель даних, засоби реалізації, практична реалізація, висновки

6. Консультанти випускної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

Дата видачі завдання _____.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів випускної проекту	Термін виконання етапів проекту	Примітка
1	Ініціювання	до 23.09.2023	
2	Планування	до 04.10.2023	
3	Реалізація	до 25.11.2023	
4	Завершення	до 16.12.2023	
5	Оформлення документації	до 07.01.2024	

Магістрант _____ Гаджибейлі І. Ф. Огли

Керівник роботи _____ к.т.н., доц. Неня А.В.

АНОТАЦІЯ

Записка: 88 сторінок, 58 рисунків, 4 додатки, 33 використаних джерел та літератури.

Обґрунтування актуальності теми роботи: В умовах активного розвитку онлайн-технологій та постійного зростання віртуальної аудиторії, можливість швидкого та зручного бронювання номерів через веб-систему стає важливим критерієм для клієнтів. Користувач хоче мати можливість легко знаходити, порівнювати та бронювати номери без зайвих труднощів.

Мета роботи: розробка web-орієнтованої системи аналізу та бронювання номерів в готелі.

Методи дослідження: Використання та аналіз сучасних веб-технологій, дослідження кращих практик у сфері управління готельним бізнесом.

Результат: Розроблена система замовлення послуг, спрямована на оптимізацію процесу пошуку номерів та виконання бронювання. Це дозволить автоматизувати та полегшити процеси прийому та виконання замовлень.

Ключові слова: web-система, система аналізу, MERN, MongoDB , React.js, готельний бізнес.

ЗМІСТ

ВСТУП

Актуальність. У сучасному взаємопов’язаному світі веб-системи відіграють ключову роль у формуванні нашого цифрового світогляду. Вони стали незамінним інструментом для бізнесу, приватних осіб та суспільства в цілому. Від свого скромного початку до динамічної галузі, що розвивається сьогодні, веб-системи зробили революцію в тому, як ми спілкуємося, взаємодіємо та ведемо бізнес.

Веб-система дає можливість приватним особам і компаніям створити помітну присутність в Інтернеті, забезпечуючи глобальний зв’язок і доступ до інформації. Веб-сайти слугують віртуальними вітринами, що демонструють товари та послуги світовій аудиторії. Вони сприяють електронній комерції, дозволяючи здійснювати безперешкодні транзакції та відкриваючи нові шляхи для процвітання бізнесу. Крім того, веб-розробка уможливлює ефективну комунікацію та співпрацю, надаючи людям можливість обмінюватися ідеями, знаннями та налагоджувати зв’язки через кордони.

Впровадження веб-технологій впливає на розвиток суспільства та підходів до ведення бізнесу, забезпечуючи доступність інформації та сприяючи інноваціям. Оскільки веб-технології продовжують розвиватися, веб-розробники будуть в центрі трансформації цифрового світу. Майбутнє обіцяє захоплюючі перспективи, зумовлені новими технологіями та невпинним прагненням створювати винятковий веб-досвід. В епоху цифрових технологій веб-розробка залишається каталізатором прогресу, розширюючи можливості людей, бізнесу та суспільства.

Розвиток Інтернету демократизував доступ до інформації, зруйнувавши географічні бар’єри сприяючи своїй легкодоступності. Веб-системи стали потужним інструментом для просування послуги бронювання готелів, дозволяючи людям обирати послугу під власні смаки. Веб-розробка трансформувала сферу замовлення послуг: системи аналізу та резервації

надають доступні та гнучкі можливості для вибору необхідної послуги користувачу. Крім того, веб-розробка сприяла прогресу в системі аналізу інформації, управлінні в галузі адміністрації сайту, підвищуючи ефективність і якість життя людей.

Мета. Розробка веб-орієнтованої системи аналізу та бронювання номерів в готелі. Основною метою є створення системи, за допомогою якої користувач може переглянути доступні готелі, скористатись системою підбору готелю і номерів, додати власний готель та виконати замовлення, обравши номер готелю для власних потреб.

Мета декомпозується на наступні задачі:

- виконання детального аналізу предметної області та огляд останніх публікацій;
- аналіз аналогів та критеріїв системи для замовлення готелю, які будуть використовуватись при розробці веб-орієнтованої системи;
- розробка алгоритму пошуку та резервації доступних готелів;
- розробка системи, що використовує стек технологій та сховище даних;
- проведення тестування програмного продукту.

Об’єкт дослідження. Процес аналізу доступних апартаментів та резервації у web-системі аналізу та бронювання номерів в готелі.

Предмет дослідження. Веб-орієнтована система аналізу та бронювання номерів в готелі з використанням стеку MERN для створення та розгортання унікальних SPA застосунків.

Практична новизна. Розробка веб-орієнтованої системи аналізу та бронювання номерів в готелі з використанням стеку MERN надає системі можливості зберігати ресурси системи. Даний стек дозволяє використовувати окремі компоненти або комбінацію всіх чотирьох (MongoDB, Express.js, React.js, Node.js), в результаті чого можна отримати рішення, адаптоване до конкретних потреб. Така гнучкість відкриває безмежні можливості для

розширення і зростання, навіть без модифікації, що призводить до підвищення продуктивності та ефективності.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд останніх досліджень і публікацій

На даний момент, публікації та дослідження в сфері розробки веб-систем надають чітке розуміння, як підходи та аспекти вдосконалення користувацького досвіду та ефективності додатків впливають на повсякденне життя користувача.

В публікації Smith J. «React Native Mobile App Development: Best Practices for Performance Optimization.» описується процес оптимізації процесу розробки web-сервісів, використовуючи технологію React.js. Автор надав акцент на головній особливості даної технології – здатність тестувати додаток. Іншими словами, зберігаючи можливість тестувати додаток, розробники отримують шанс проаналізувати його роботу на різних типах пристроїв. Відстеження даних і швидкість налагодження - це ефективні підходи в розробці додатків, що дозволяють виконувати належне тестування в рамках вашої команди розробників [1].

В статті Johnson, A. «Responsive Web Design: Techniques for Cross-Browser Compatibility» автор акцентував увагу на можливості повторного використання компонентів у процесі розробки [2]. На думку автора, ця перевага дозволяє починати цикл розробки з невеликих частин і поступово переходити до більших, зрештою отримуючи основу додатку на основі компонентів. Завдяки можливості повторного використання цих компонентів команда розробників може заощадити час і гроші, зосередившись на інших завданнях. Таким чином, можна отримати більш оптимізований додаток, заощаджуючи ресурси та максимально використовуючи можливості фреймворку.

На думку Brown, M., при розробці веб-системи з використанням React.js, в першу чергу необхідно звернути увагу на найважливіші компоненти, які будуть в ролі фундаменту додатку. В статті «Enhancing User Experience through

Progressive Web Applications» автор розглядає Virtual DOM як ще один приклад багатьох переваг використання React.js для розробки додатків [3] . На його думку, розробник може прибрати важкі базові процеси під час роботи над додатком, залишивши лише обов'язкові. Такі рішення в кінцевому підсумку призводять до звуження обсягу робіт, а також підвищують продуктивність вашої команди розробників. Саме тоді React.js показує свій справжній потенціал.

Загалом, переваги React.js для розробки веб-систем дуже широкі, починаючи від потужної спільноти та можливостей тестування і закінчуючи доступним навчальним курсом та підвищеною продуктивністю. Якщо порівнювати цей фреймворк з його аналогами, такими як Vue.js або Angular.js, React.js виділяється досить яскраво [4].

В індустрії веб-розробки кількість технологій, фреймворків та інструментів стрімко зростає.При виборі конкретного фреймворку або бібліотеки JavaScript має безліч варіантів для фронтенд-розробки, таких як VueJS, TezJS та Svelte. Однак Angular і React міцно закріпилися на вершині списку, оскільки багато розробників вважають їх найкращими фреймворками для фронтенд-розробки завдяки своїй популярності [5]. Детальне порівняння даних технологій описано в таблиці нижче.

Таблиця 1.1 – Порівняльна таблиця технологій React.js та Angular.js

Критерій	React.js	Angular.js
Компанія-розробник	Meta	Google
Тип	Бібліотека	Фреймворк
Мови	JavaScript, JSX	TypeScript

Продовження таблиці 1.1 – Порівняння таблиця технологій React.js та Angular.js

DOM	Virtual DOM	Використовує реальний DOM
Інтеграція з іншими технологіями	Легко інтегрується з іншими бібліотеками та фреймворками	Складна інтеграція з не-Angular бібліотеками
Інтеграція в існуючі проекти	Легко інтегрується в існуючі проекти	Важко інтегрується з існуючими проектами
Компоненти	Класові та функціональні компоненти	Класові компоненти
Документація	Інтуїтивно зрозуміла документація	Інтуїтивно зрозуміла документація
Розмір бандла	Може бути меншим, в залежності від використання внутрішніх бібліотек	Більший розмір, через вбудовану функціональність
Інтеграція з IDE	Реалізовано через розширення IDE (React Developer Tools)	Інтеграція з IDE вбудована

Джерело: розроблено автором

Враховуючи всі переваги кожної технології, які були описані вище, було вирішено надати перевагу React.js, головна перевага якого інтеграція з іншими технологіями навіть в середині процесу розробки. [6] При розробці web-системи для резервації готелю велике значення має обробка та повернення запитів між клієнтом та сервером.

1.2 Порівняння аналогів web-системи

Web-орієнтований сервіс «Booking.com» реалізований за допомогою технології React.js, що надає даній системі швидкість в обробці даних та виводі запитів користувачі. Даний сервіс має інтуїтивно зрозумілий інтерфейс та приємний дизайн з вдало підібраними кольорами. На рисунку 1.1 представлено вигляд головної сторінки сервісу Booking.com. З головних переваг даного сервісу можна відзначити широкий асортимент готелів, хостелів та інших видів проживання по всьому світу. Прозора система відгуків надає користувачу чітке розуміння про популярність обраного готелю. [7]

Незважаючи на більшість плюсів сервісу, дана система має свої недоліки. У деяких випадках, вартість резервації готелю може бути вищою, ніж зазначена на сайті через комісійні внески. У деяких випадках користувачу буде важко знайти потрібні апартаменти через обмежені можливості фільтрації результатів пошуку.

При розробці веб-орієнтованої системи аналізу та бронювання номерів в готелі необхідно врахувати всі недоліки сервісу Booking.com та модернізувати вже існуючі технології пошуку готелю.

Наступним популярним онлайн-сервісом для аналізу та бронювання номерів в готелі є Trivago. Даний сервіс надає користувачу можливість за допомогою системи підбору номерів обрати потрібний з декількох сервісів одразу. Фільтрація даних допомагає ознайомитися з доступними послугами готелів. За допомогою інтуїтивно зрозумілого інтерфейсу користувач зможе знайти потрібну сторінку та ознайомитися з можливістю сервісу щодо підбору номеру. Також система надає можливість оцінювати обраний номер готелю задля рекомендації іншим користувачам.

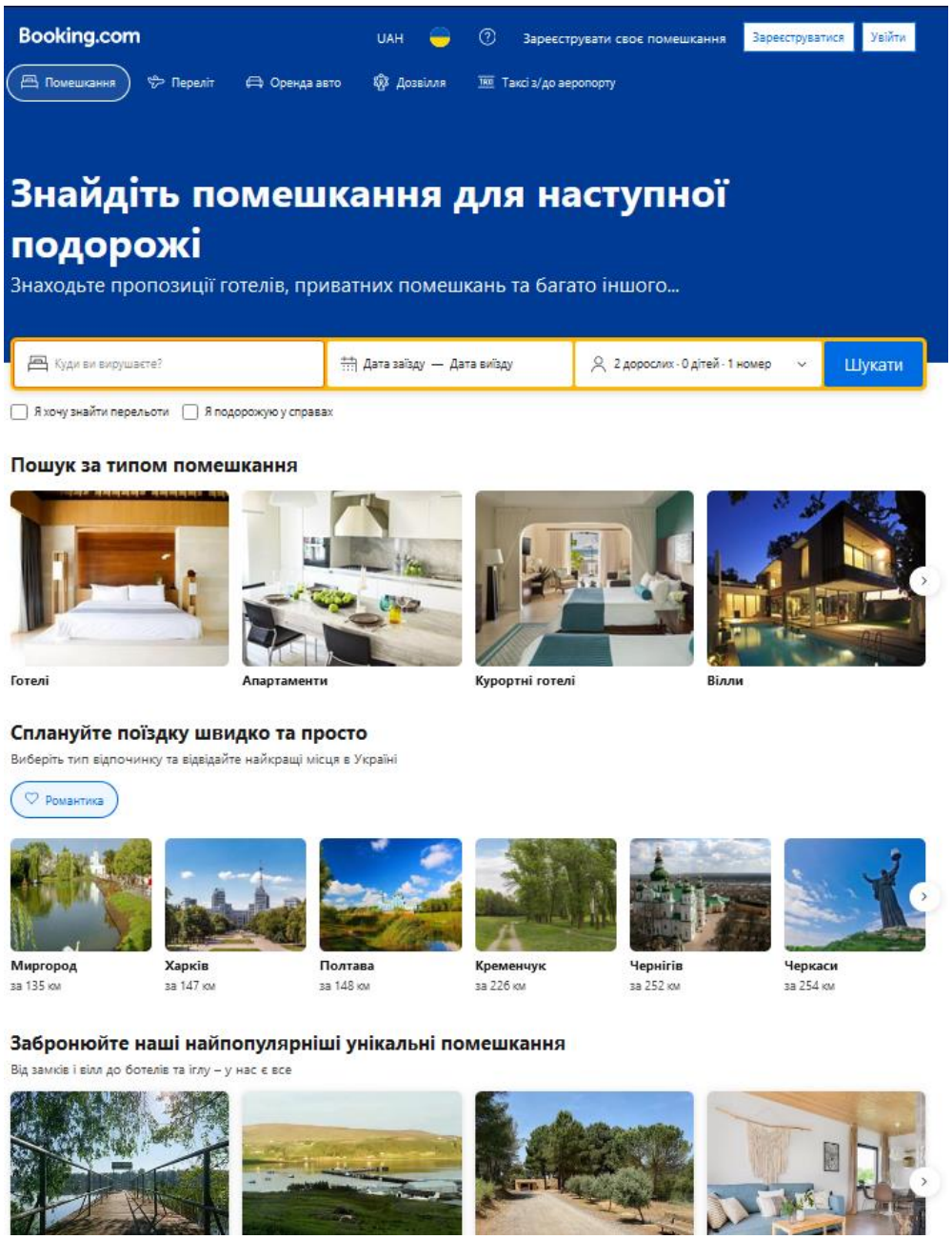

Рисунок 1.1 – Вигляд головної сторінки сервісу Booking.com

Джерело: розроблено автором

З мінусів сервісу можна відзначити повільну швидкість обробки даних та некоректний вивід інформації по запиту. На рисунку 1.2 представлено вигляд головної сторінки сервісу trivago.com.

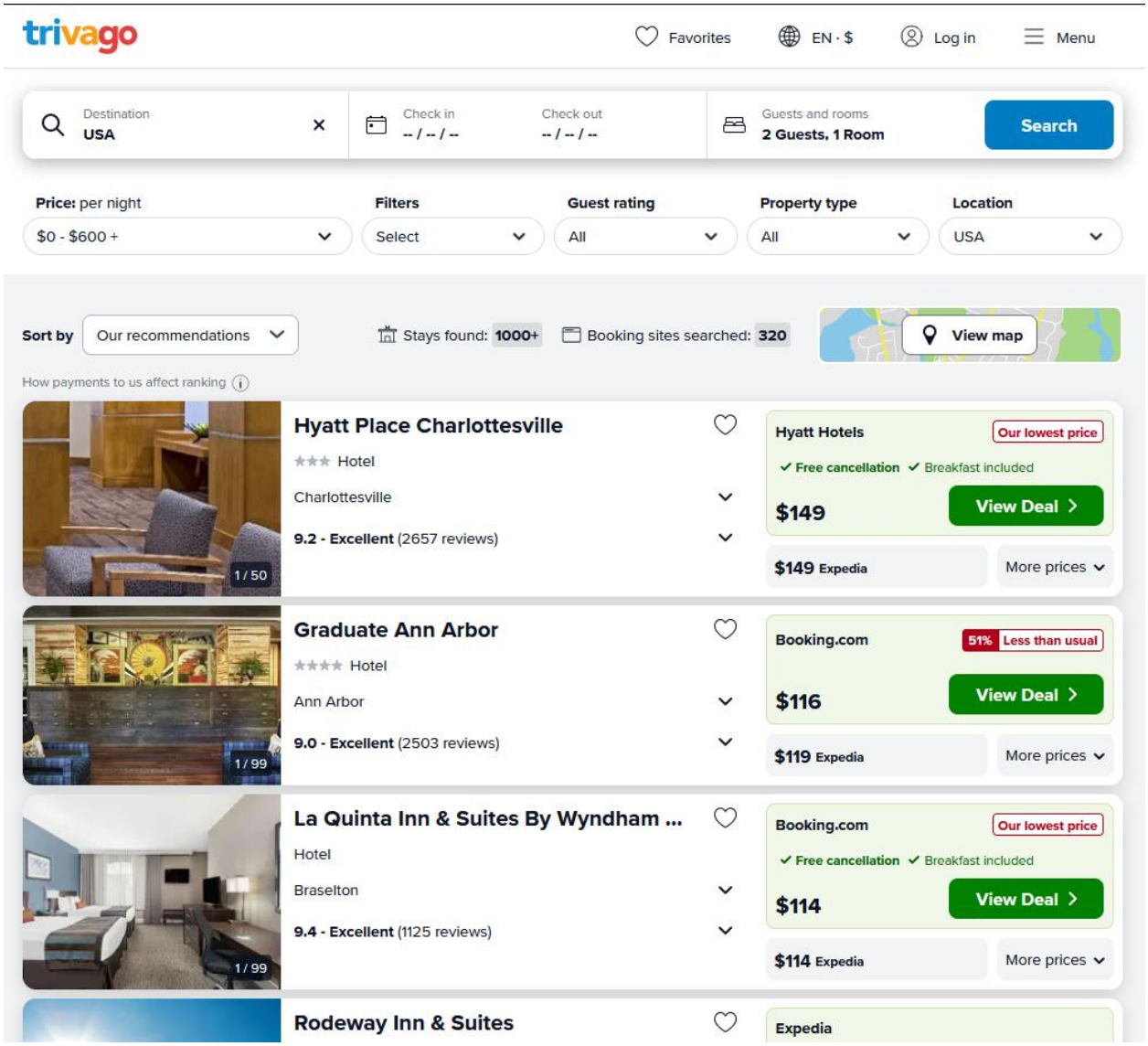

Рисунок 1.2 – Вигляд головної сторінки сервісу trivago.com

Джерело: розроблено автором

2 ПОСТАНОВКА ЗАДАЧІ

2.1 Мета та задачі дослідження

Мета: розробити веб-орієнтовану систему аналізу та бронювання номерів в готелі.

Функціональні вимоги:

- система повинна здійснювати аналіз існуючих даних для відповіді на запит користувача до серверу;
- забезпечувати аналіз показників кількості замовлень через особисту сторінку;
- система повинна визначати фактори, що впливають на популярність, такі як кількість готелів, задоволеність користувачів, та пропозиції готелів;
- надання можливості інтеграції з web-додатком для ілюстрації роботи системи.

Нефункціональні вимоги:

- забезпечення конфіденційності та захисту особистих даних користувачів, включаючи фінансову інформацію;
- готовність до масштабування для обробки великої кількості даних та запитів користувачів;
- забезпечувати стабільну та надійну роботу системи під час великого обсягу бронювань та запитів;
- можливість інтеграції системи з різними готельними платформами та іншими додатками для покращення функціоналу.

Для досягнення мети необхідно виконати наступні задачі:

- провести аналіз існуючих технологій для реалізації системи;
- проаналізувати існуючі показники популярності систем-аналогів;

- проаналізувати слабкі сторони сервісів-аналогів та визначити способи їх покращення;
- розробити веб-додаток для демонстрації роботи системи.

Гіпотеза: система аналізу та бронювання номерів в готелі надасть користувачу спрощений спосіб пошуку конкретного готелю за наданими критеріями, а також покращить процес оформлення замовлення.

2.2 Методи дослідження

Для більш детального аналізу сфери резервації готелю буде використано систему GoogleTrends, яка по заданим запитам надає інформацію про кількість запитів та подає всю інформацію в вигляді діаграми [8]. При запиті «Hotel Reservation», система видала інфографіку на якій можна побачити, що популярність даних сервісів та систем припадає на свята або відпустки. На рисунку 2.1 представлено графік популярності запиту «Hotel Reservation».

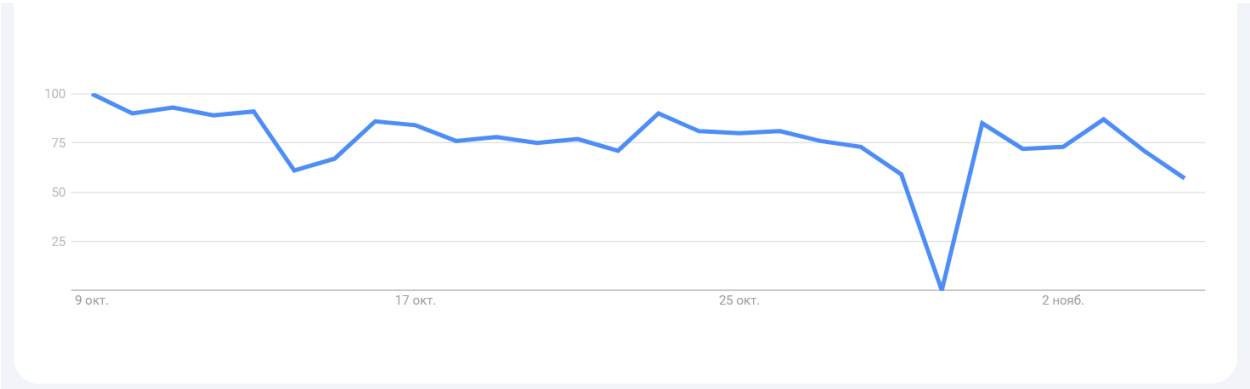

Рисунок 2.1 – Графік популярності запиту «Hotel Reservation»
Джерело: [8]

Якщо проаналізувати даний запит в Україні то результат не надасть конкретної інформації на даний момент через повномасштабне вторгнення.

Але якщо звернути увагу на дані 2021 року то можна побачити, що дані запити частіше були задані користувачами в пошуковій системі. На рисунку 2.2 представлений графік популярності запиту в 2021 році.

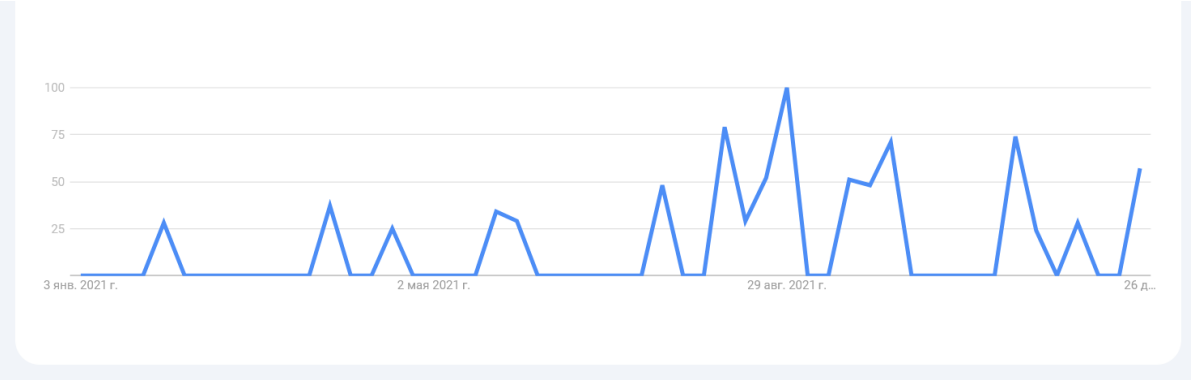

Рисунок 2.2 – Графік популярності запиту «Hotel Reservation» в Україні за 2021 рік

Джерело: [8]

Проаналізовані дані надають чітке розуміння в потребі даної системи та в потребах користувачів щодо користування даних web-системи. На рисунку 2.3 представлено графік запитів по пошуку популярного сервісу для резервації готелів Booking.com.

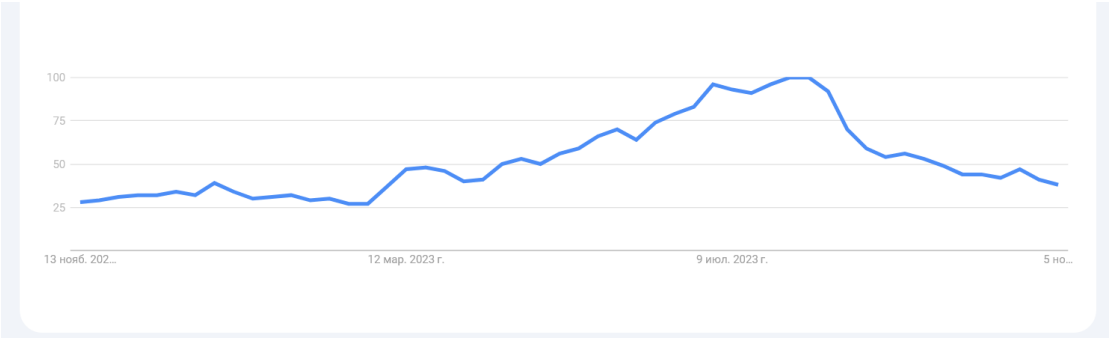

Рисунок 2.3 – Графік популярності запиту «Booking.com»
в Україні за 2023 рік
Джерело: [8]

2.3 Методи реалізації

Для розробки веб-орієнтованої системи аналізу та бронювання номерів в готелі було прийнято рішення використати стек технологій, які мають найкращу взаємодію з React.js – MERN.

Стек MERN складається з чотирьох технологій: MongoDB, ExpressJS, ReactJS та Node.js. MongoDB – це документно-орієнтована СУБД, яка дозволяє зберігати дані в JSON-подібному форматі [9].

ExpressJS – це мінімальний і гнучкий фреймворк для веб-додатків на node.js, який надає надійний набір функцій для веб- та мобільних додатків за допомогою проміжного програмного забезпечення. Він допомагає розробникам легко обробляти HTTP-запити та відповіді сервера, а також маршрутизувати запити до файлів або файлових контролерів у додатку. [10]

Стек MERN розділений на два компоненти: бек-енд і фронтенд. Крім того, вся система баз даних ізольована від решти.

Вся система, включаючи фронтенд, бекенд і базу даних, використовує REST API, який діє як "проміжне програмне забезпечення" і може бути

повторно використаний для будь-якого іншого додатку: мобільного програмного забезпечення тощо.

REST API дозволяє з'єднувати додатки один з одним та заснований на HTTP та імітує стилі веб-спілкування, що робить дану технологію дуже вигідною для використання в стеку MERN [11].

За допомогою REST API розробники мають можливість, наприклад, розробляти нові інтерфейси і створювати нові додатки, які можна дуже легко пов'язати з однією і тією ж системою [12].

Node.js дуже швидкий просто тому, що це асинхронна мова. Її продуктивність вражає у порівнянні з середніми показниками інших мов.

Швидкий бекенд означає, що користувачі додатку отримають доступ до своїх даних набагато швидше, а це дуже вигідно для заохочування користувачів. Це особливо актуально, якщо врахувати, що сьогодні середній прийнятний час очікування користувача на завантаження веб-сторінки становить менше 1 секунди.

До того ж, збільшення швидкості дозволить зменшити витрати на експлуатацію сервера [13].

На рисунку 2.5 представлено приклад роботи стеку MERN в клієнт-серверній архітектурі.

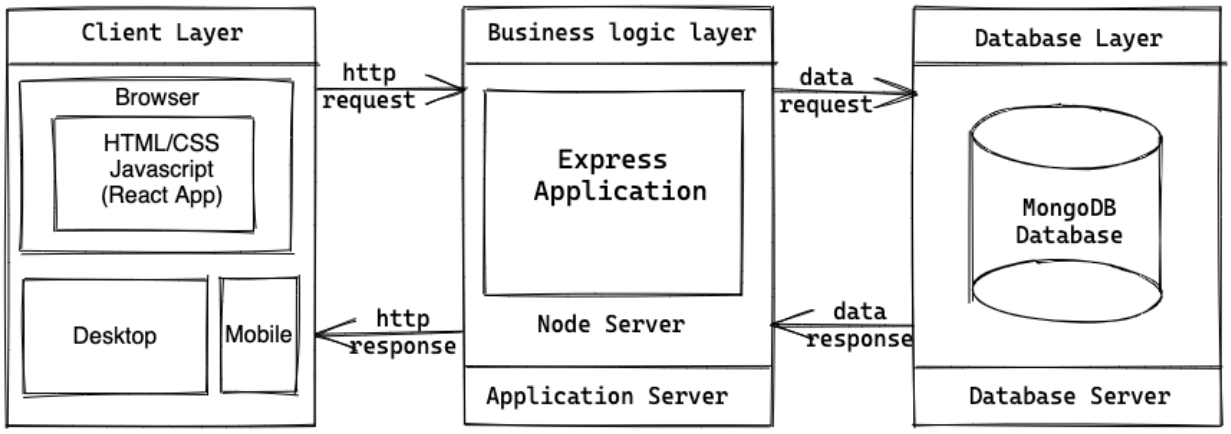

Рисунок 2.5 – Схематична ілюстрація роботи стеку MERN в клієнт-серверній архітектурі

Джерело: [13]

Одна з найвагоміших переваг стеку MERN – це модульність. Можна використовувати окремі компоненти ізольовано або об’єднати всі чотири інгредієнти в єдине ціле, яке є більшим, ніж його частини. Ця модульність робить даний стек технологій дуже гнучким і допомагає створити індивідуальне рішення, пристосоване до потреби створення web-орієнтованої системи аналізу та бронювання номерів в готелі.

3 ПРОЕКТУВАННЯ

3.1 Структурно-функціональне моделювання

IDEF0 – методологія функціонального моделювання і графічна нотація, призначена для формалізації та опису бізнес-процесів. Відмінною особливістю IDEF0 є її акцент на співпідпорядкованість об’єктів [14].

Функціональна модель IDEF0 являє собою набір блоків, кожен з яких являє собою "чорну шухляду" зі входами та виходами, управлінням і механізмами, які деталізуються (декомпозуються) до необхідного рівня. Найважливіша функція розташована у верхньому лівому кутку. А з’єднуються функції між собою за допомогою стрілок і описів функціональних блоків. При цьому кожен вид стрілки або активності має власне значення. Ця модель дає змогу описати всі основні види процесів, як адміністративні, так і організаційні [15]. Стрілки можуть бути:

Вхідні – такі, що вводять, які ставлять певне завдання.

Вихідні –такі, що виводять результат діяльності.

Контроль (зверху вниз) – механізми управління (положення, інструкції тощо).

Механізми (від низу до верху) – що використовується для того, щоб виконати необхідну роботу.

На рисунку 3.1 представлено контекстну діаграму в нотації IDEF0 для веб-орієнтованої системи аналізу та бронювання номерів в готелі.

Рисунок 3.1 – Контекстна діаграма в нотації IDEF0 для веб-орієнтованої системи аналізу та бронювання номерів в готелі.

Джерело: розроблено автором

Функціональна декомпозиція – це метод аналізу, у великому процесі, яке за допомогою декомпозиції розбиває цей процес на менші, легші для розуміння одиниці.

У бізнесі функціональна декомпозиція використовується для полегшення розуміння та управління великими і складними процесами. Функціональна декомпозиція допомагає вирішувати проблеми і сприяє розвитку бізнес-операцій, комп’ютерного програмування, машинного навчання та багатьох інших галузей [16].

На рисунку 3.2 представлено декомпозицію IDEF0-діаграми для web-орієнтованої системи аналізу та бронювання номерів в готелі.

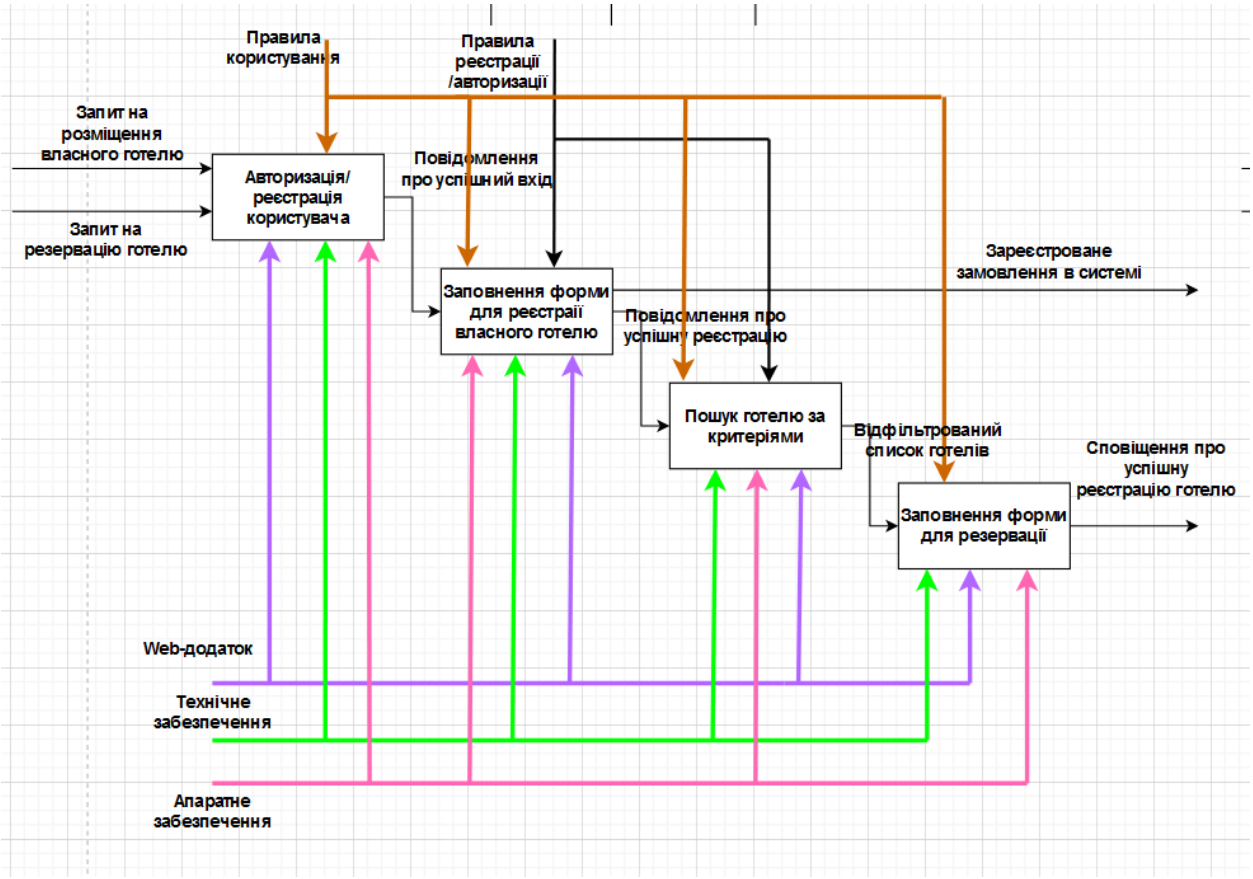

Рисунок 3.2 – Декомпозиція IDEF0-діаграми для веб-орієнтованої системи аналізу та бронювання номерів в готелі

Джерело: розроблено автором

3.2 Моделювання варіантів використання

Діаграма варіантів використання UML моделює дії, що відбуваються в системі. Це тип динамічної діаграми (на відміну від статичної), оскільки вона показує, як поведінка користувача або системи впливає на можливі взаємодії або зміни в процесі [17].

Найбільша перевага діаграми варіантів використання полягає в тому, що вона допомагає розробникам програмного забезпечення та компаніям проектувати процеси з точки зору користувача. В результаті система функціонує більш ефективно і служить цілям користувача.

Діаграма варіантів використання веб-орієнтованої системи аналізу та бронювання номерів в готелі представлена на рисунку 3.3.

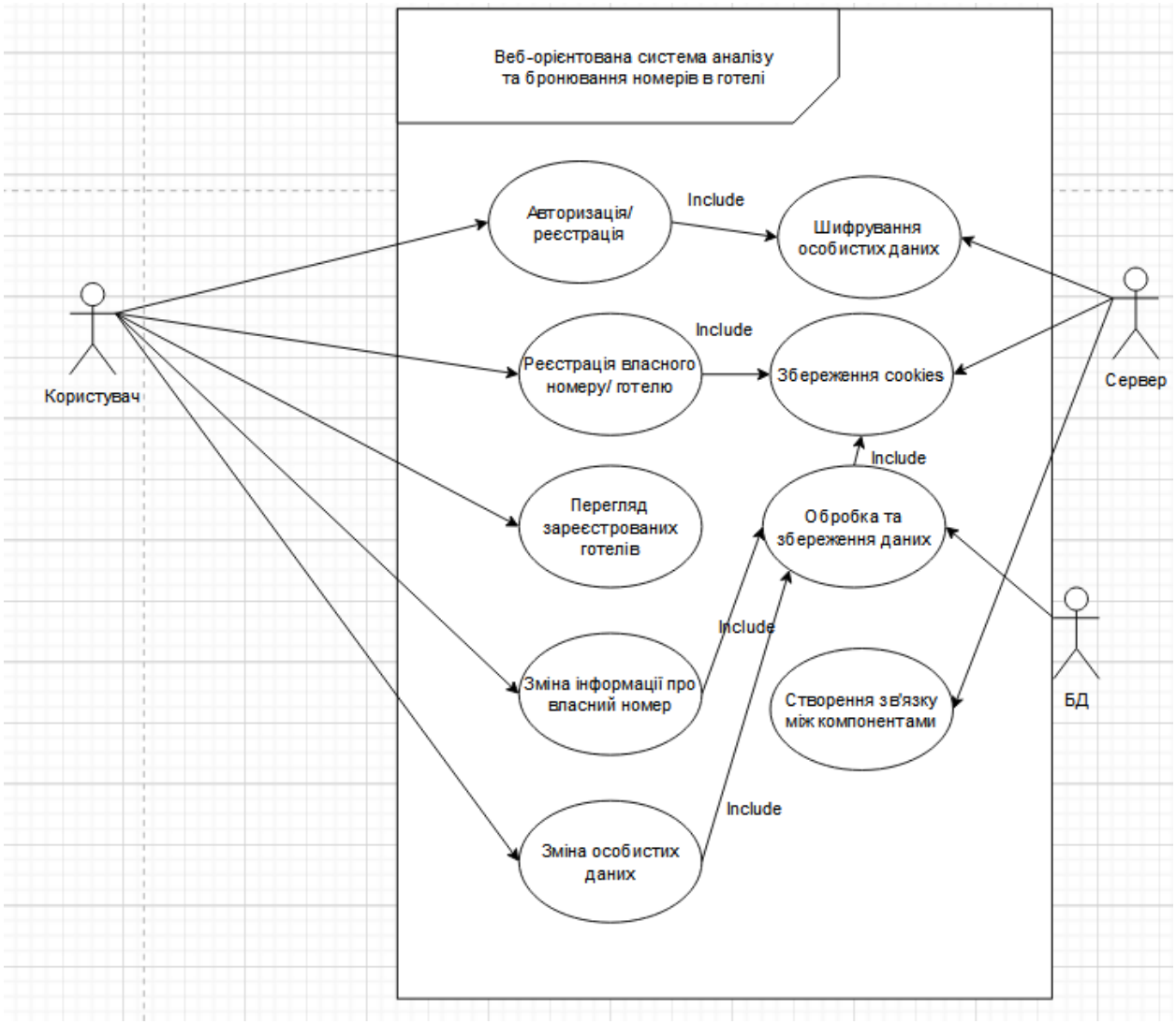

Рисунок 3.3 – Діаграма варіантів використання веб-орієнтованої системи

Джерело: розроблено автором

3.3 Проектування моделі бази даних

Проектування бази даних – це процес створення структурованого плану бази даних, який описує спосіб організації даних та спосіб доступу до них. Даний етап включає розробку схеми бази даних, яка визначає структуру та

взаємозв’язки між елементами даних, а також визначає найбільш підходящі типи даних та механізми зберігання даних.

База даних документів містить інформацію, отриману або збережену у вигляді документа або, іншими словами, напівструктуровану базу даних. Оскільки вони не є реляційними, їх часто називають NoSQL-даними [18].

База даних документів отримує і накопичує дані у вигляді пар ключ-значення, але тут значення називаються документами. Документ може бути представлений як складна структура даних. Документ тут може бути у вигляді тексту, масивів, рядків, JSON, XML або будь-якого іншого подібного формату. Використання вкладених документів також дуже поширене. Це дуже ефективно, оскільки більшість створених даних зазвичай мають форму JSON і є неструктурованими [19].

На рисунку 3.4 представлений документ збереження готелю.

Рисунок 3.4 – Документ збереження готелю веб -орієнтованої системи аналізу та бронювання номерів в готелі

Джерело: розроблено автором

Даний документ являє собою набір пар ключ-значення. В даному документі _id це ключ, а ObjectId – значення. На рисунках 3.5 – 3.6 представлено документи збереження кімнати та користувача.

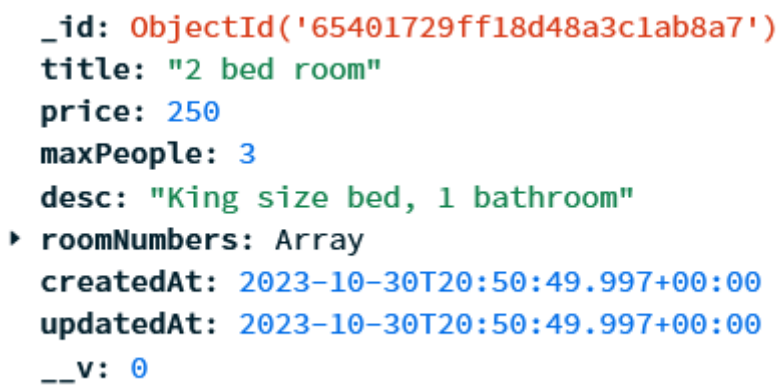

Рисунок 3.5 – Документ збереження кімнати веб-орієнтованої системи аналізу та бронювання номерів в готелі.

Джерело: розроблено автором

Рисунок 3.5 - Документ збереження користувача веб-орієнтованої системи аналізу та бронювання номерів в готелі.

Джерело: розроблено автором

Для даної системи було використано хмарне середовище MongoDB. Це несистемна база даних, яка зберігає дані у вигляді документів у форматі BSON (бінарний JSON). MongoDB надає гнучку структуру даних, не обмежуючи їх схемою, що відрізняє його від традиційних реляційних баз даних [20].

4. РЕАЛІЗАЦІЯ ВЕБ-ОРІЄНТОВАНОЇ СИСТЕМИ

4.1 Архітектура веб-системи

Стек MERN поєднує в собі MongoDB, Express, React та Node.js, щоб створити повноцінний фреймворк JavaScript для створення веб-додатків. Кожен компонент відіграє певну роль у процесі розробки.

Архітектура web-орієнтованої система має наступний вигляд:

- Користувач взаємодіє з інтерфейсом веб-додатку, який побудований за допомогою React-компонентів.
- React фіксує взаємодію користувача та оновлює стан додатку, якщо це необхідно.
- Якщо взаємодія вимагає отримання або оновлення даних, React надсилає API-запит на сервер Express.
- Express обробляє API-запит і зв’язується з MongoDB для виконання необхідних операцій з даними.
- MongoDB зберігає, витягує або оновлює дані за потреби, і результат надсилається назад до Express.
- Express надсилає відповідь назад до React, який відповідно оновлює інтерфейс користувача.

Така взаємодія між фронтендом (React) та бекендом (Express, MongoDB та Node.js) робить стек MERN потужним та ефективним фреймворком для створення web-орієнтованої системи.

На рисунку 4.1 представлено вигляд схеми архітектури web-орієнтованої системи аналізу та бронювання номерів в готелі.

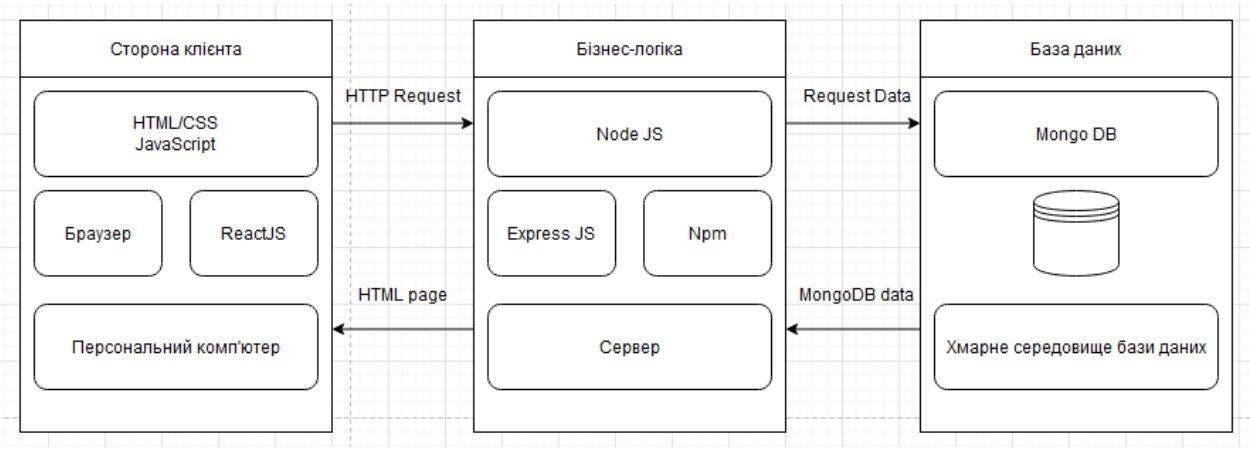

Рисунок 4.1 – Схема архітектури веб-орієнтованої системи аналізу та бронювання номерів в готелі

Джерело: побудовано автором

4.2 Створення БД та інтеграція в веб-орієнтовану систему

Для інтеграції бази даних було використано хмарне середовище MongoDB Atlas [21]. Це мультихмарна платформа даних для розробників. В її основі лежить наша повністю керована хмарна база даних для сучасних додатків. Atlas – це найкращий спосіб працювати з MongoDB, провідною нереляційною базою даних.

Модель документів MongoDB надає найшвидший спосіб впроваджувати інновації, оскільки документи відображаються безпосередньо на об’єкти у компонентах.

Для початку необхідно перейти на сайт https://cloud.mongodb.com та виконати авторизацію. Після того, як користувач виконав авторизацію то необхідно натиснути на кнопку «New Project», яка знаходиться в вкладці «Project». На рисунку 4.2 представлено початкову сторінку створення бази даних.

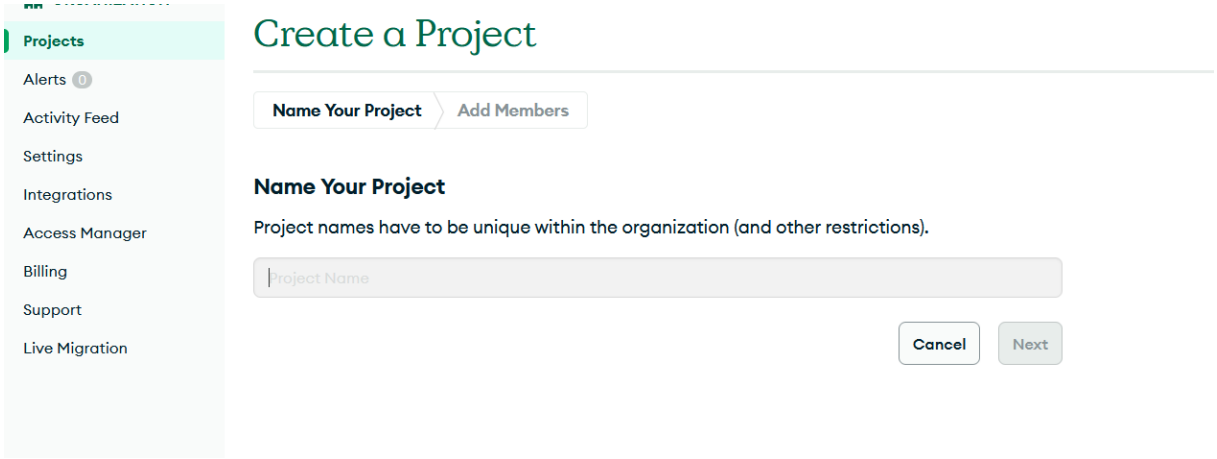

Рисунок 4.2 – Початкова сторінка створення бази даних

Джерело: побудовано автором

Після виконання операції створення бази даних необхідно перейти на сторінку створеної БД та обрати опцію розгортання з вибором регіону та налаштуванням серверу. Для економії витрат на розробку web-системи було обрано безкоштовний план використання хмарного середовища. Для безперебійного зв’язку було обрано регіон Стокгольм та провайдер від компанії AWS.

На рисунку 4.3 представлено результат налаштування серверу.

Наступним етапом буде створення адміністратора БД. Для цього користувач повинен ввести унікальний логін та пароль за яким буде надаватися доступ до системи та БД. Якщо адміністраторів декілька то це передбачено системою та надано можливість додати співучасників до адміністрування.

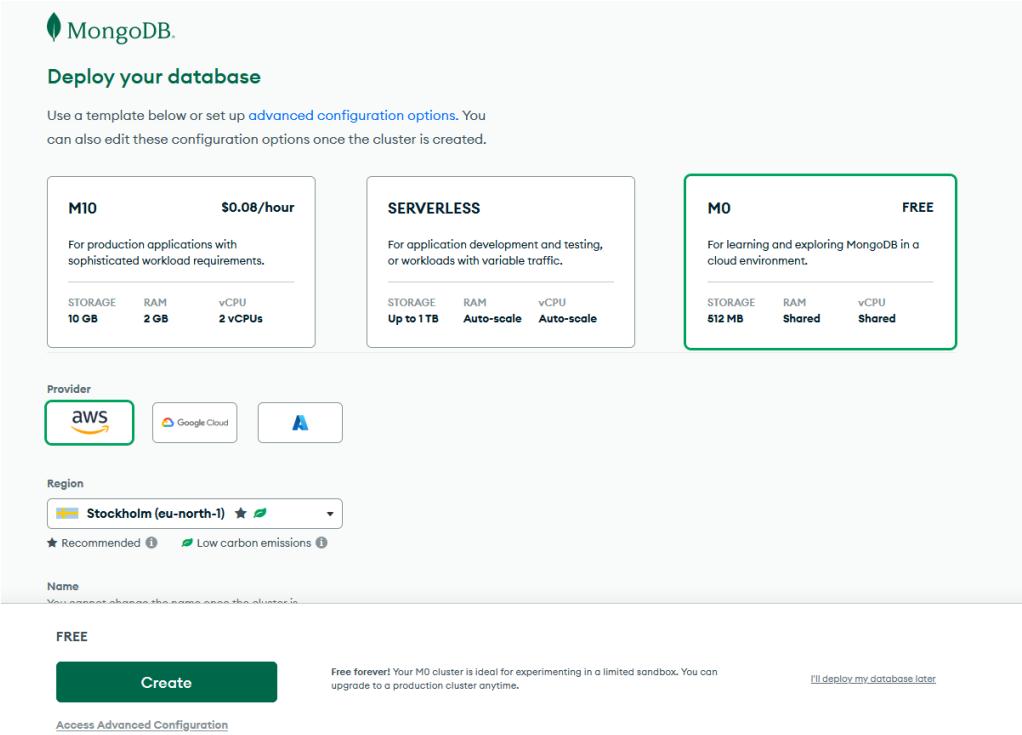

Рисунок 4.3 – Результат налаштування серверу

Джерело: побудовано автором

На рисунку 4.4 представлено процес створення користувача.

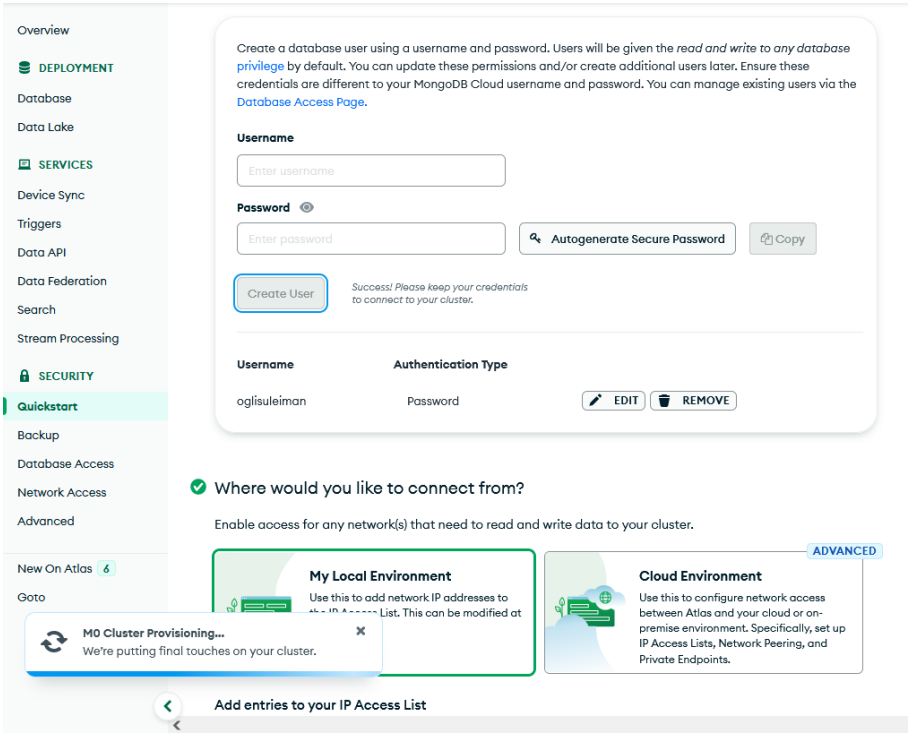

Рисунок 4.4 – Процес створення користувача

Джерело: побудовано автором

Після успішного виконання вищеописаних операцій, адміністратор отримує унікальний ключ, який необхідний для інтеграції БД в веб-систему. Для самого підключення необхідно створити .env файл, в якому буде міститися посилання на створену БД в середовищі Atlas.

На рисунках 4.5 – 4.6 представлено процес успішного підключення БД до веб-системи та наповнення файлу .env.

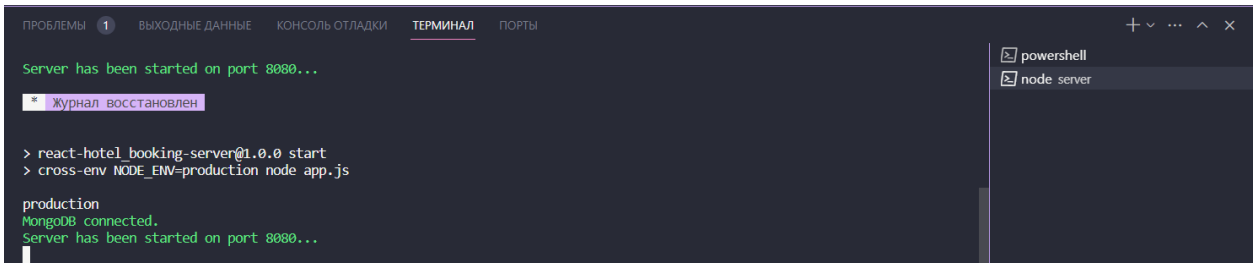

Рисунок 4.5 – Результат успішного підключення БД до web-системи

Джерело: побудовано автором

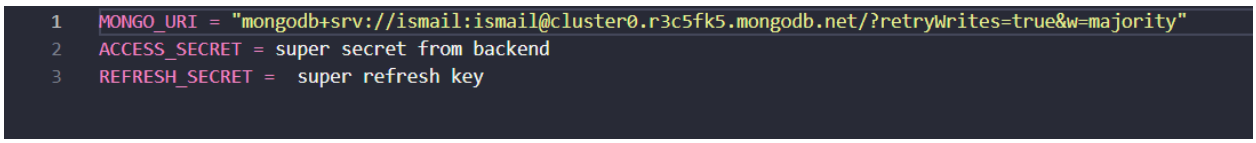

Рисунок 4.6 – Результат створення файлу .env

Джерело: побудовано автором

Також для підключення БД до веб-системи було використано бібліотеку mongoose. Це інструмент об’єктного моделювання, призначений для роботи в асинхронному середовищі. Mongoose підтримує Node.js та Deno (альфа-версія). Для установки цієї та інших бібліотек використовувався пакетний менеджер npm [22].

npm – найбільший у світі реєстр програмного забезпечення. Розробники відкритого програмного забезпечення з усіх континентів використовують npm для обміну та запозичення пакунків, а також багато організацій використовують npm для управління приватними розробками.

npm складається з трьох окремих компонентів:

– веб-сайту

– інтерфейс командного рядка (CLI)

– реєстр

На рисунку 4.7 представлено результат підключення пакетів для роботи з базою даних в файлі server.js. Повний лістинг коду представлено в додатку Б.

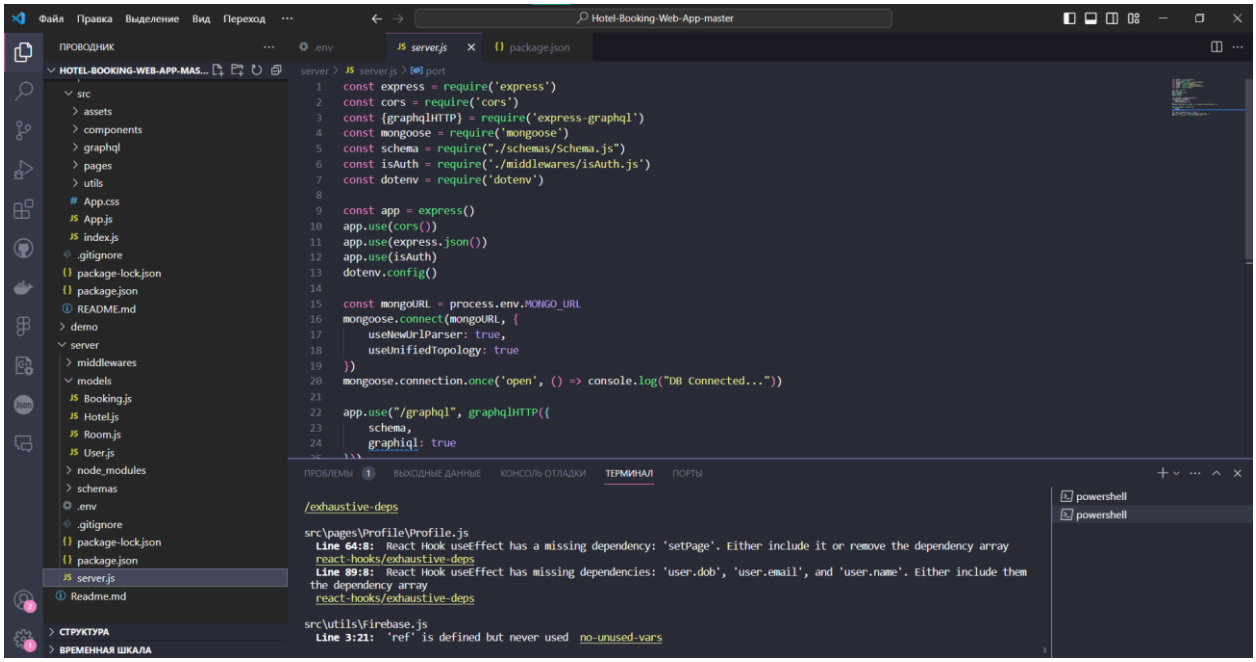

Рисунок 4.7 – Результат підключення пакетів для роботи з базою даних

Джерело: побудовано автором

Наступним етапом було створення моделей для збереження даних. На рисунках 4.8 – 4.11 представлено результати створення моделей для бази даних.

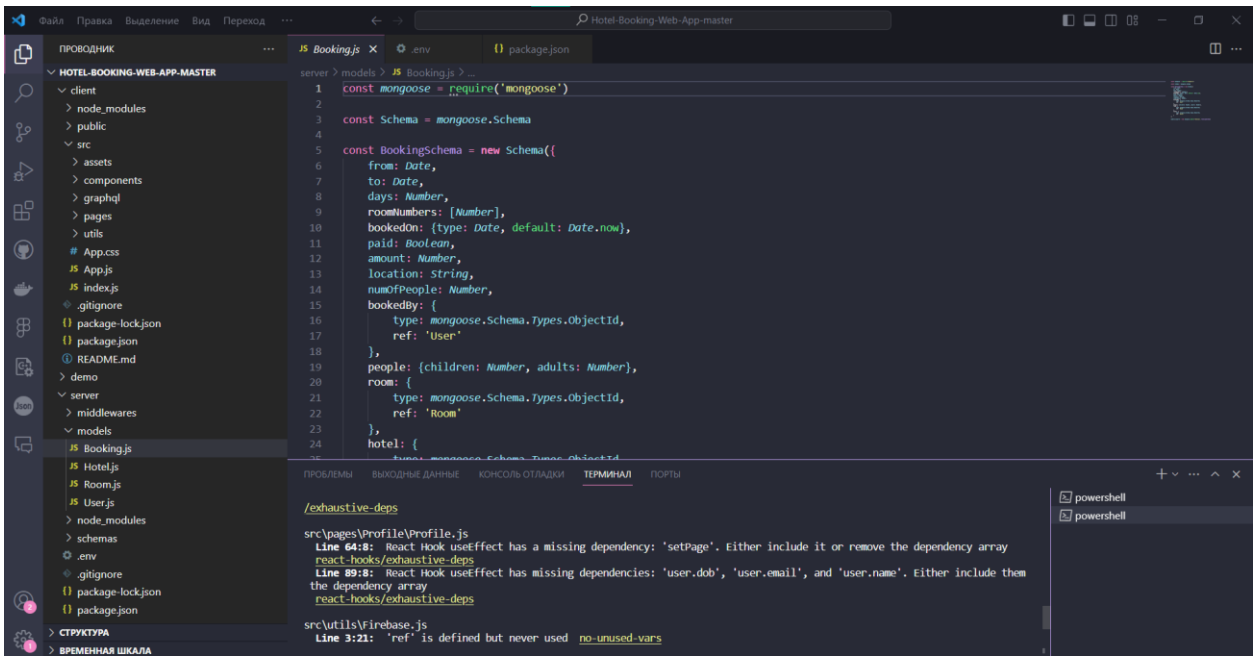

Рисунок 4.8 – Результат створення моделі Booking

Джерело: побудовано автором

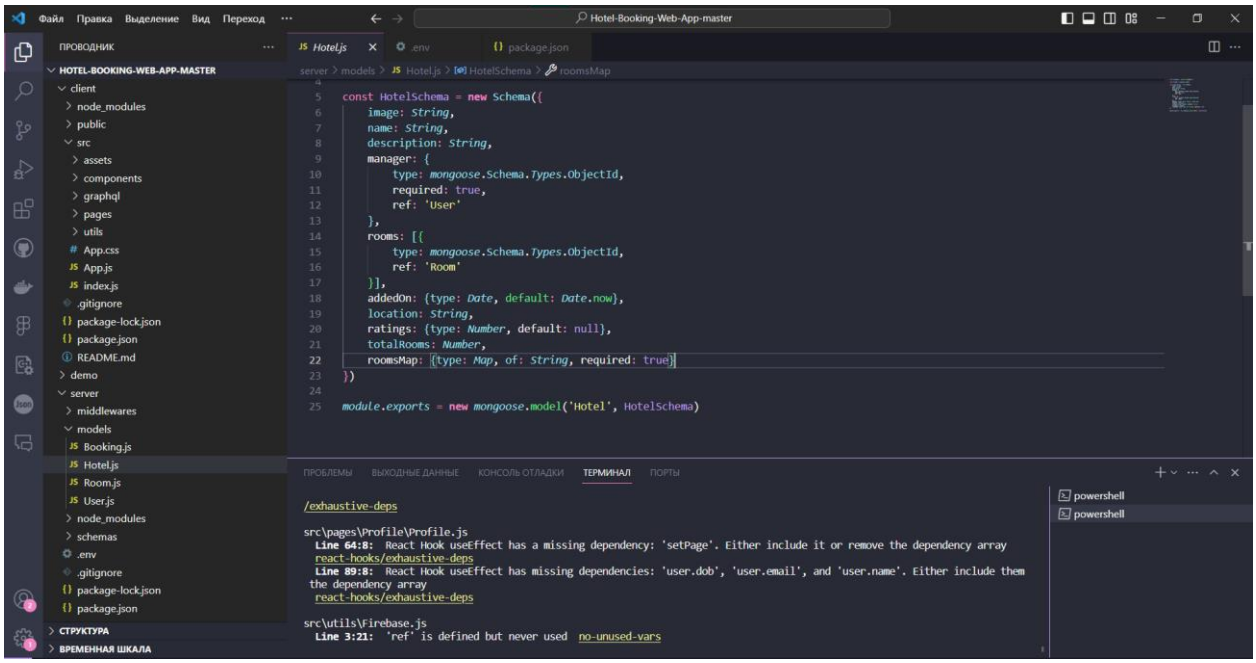

Рисунок 4.9 – Результат створення моделі Hotel

Джерело: побудовано автором

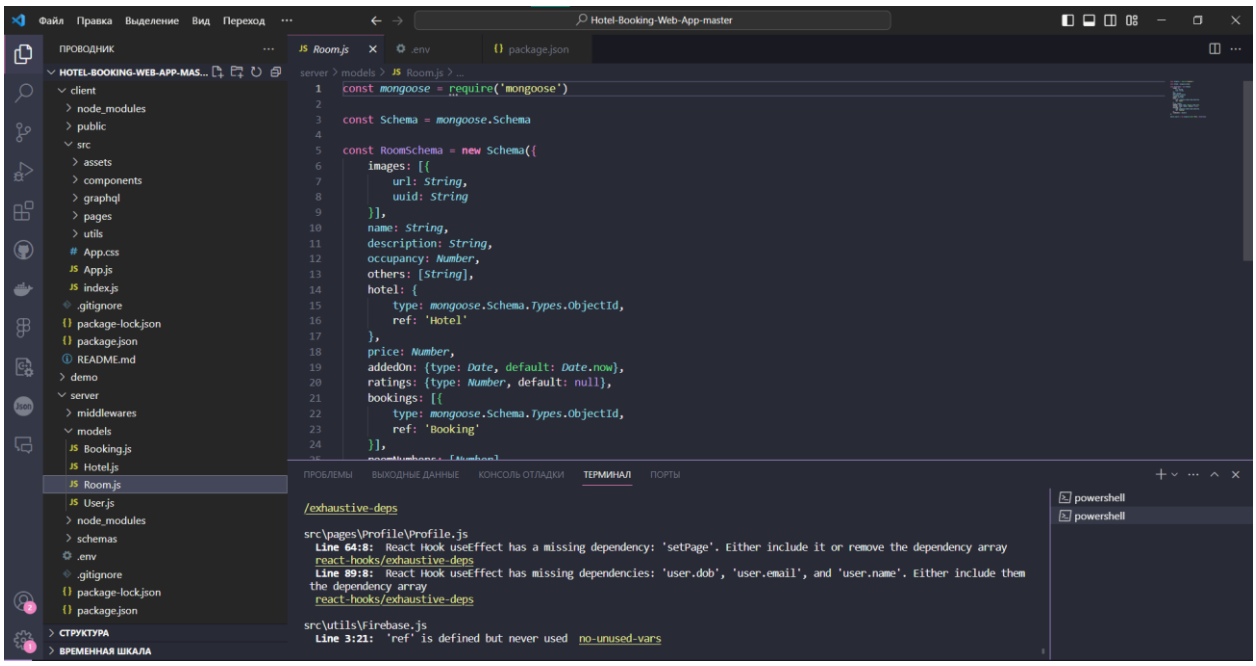

Рисунок 4.10 – Результат створення моделі Room

Джерело: побудовано автором

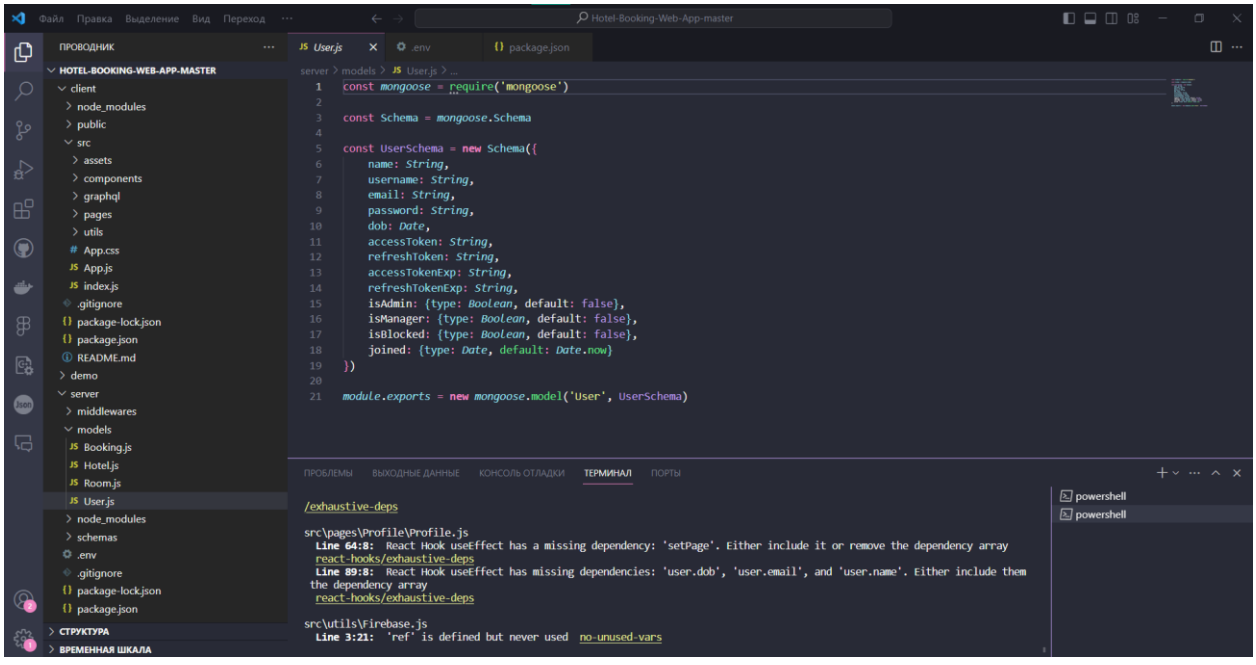

Рисунок 4.11 – Результат створення моделі User

Джерело: побудовано автором

4.3 Створення технології аналізу та бронювання номерів

Для створення технології аналізу доступних номерів за обраними критеріями були використані бібліотеки npm та мутації. На рисунку 4.12 представлено результати установки пакетів npm в файлі package.json.

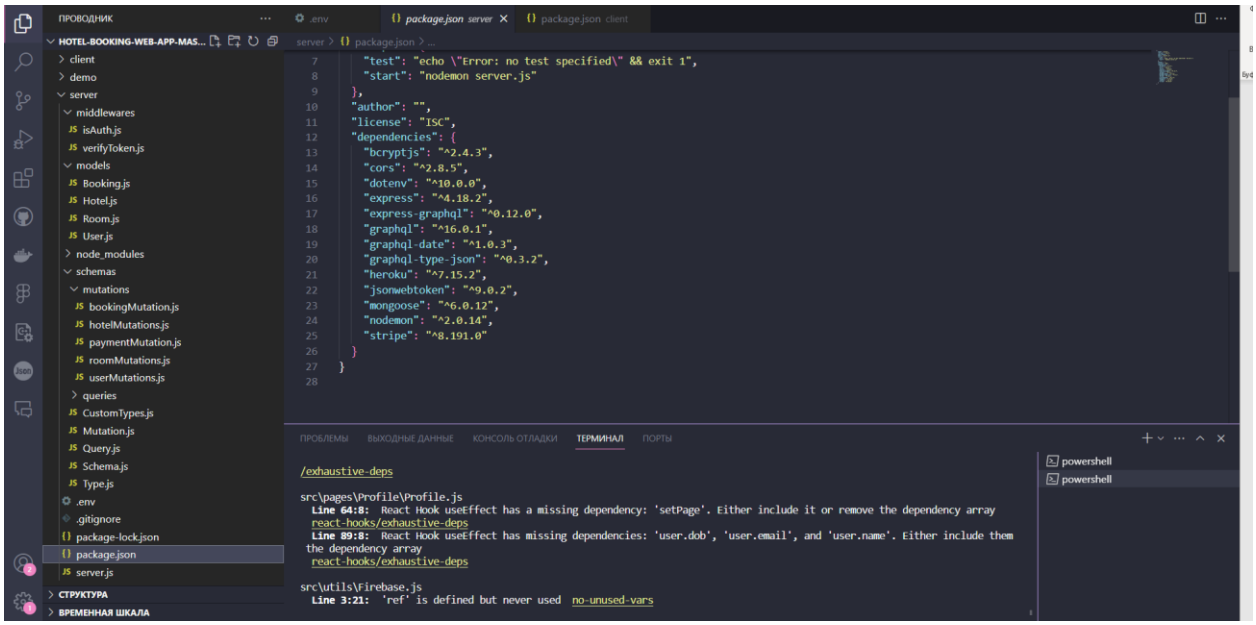

Рисунок 4.12 - Результати установки пакетів npm в файлі package.json

Джерело: побудовано автором

Мутації – це дії, які користувач може виконувати у вашому додатку. Тобто це простий хук з двома параметрами:

- Функція для обробки запиту
- Параметри для обробки, в даному випадку - хуки успіху і помилки, а також для налаштування мутації (повторна спроба, затримка повторної спроби і так далі).

Результат має три основні об’єкти:

- mutate: це дія для запуску мутації у вашому коді;
- isLoading: цей прапорець вказує, чи триває мутація;
- error: вказує на помилки, якщо в запиті виникла помилка;

Використання мутації у системі аналізу доступних номерів надає можливість спростити управління станами цих запитів.

Також важливим елементом при створенні мутацій є QueryClient.

Використовуючи QueryClient, можна зробити недійсним вже наданий запит і сказати реагуючому запиту, щоб він знову запитав дані, оскільки є впевненість, що ці дані ще не дійсні після мутації [23].

Для цього потрібно використати хук useQueryClient для отримання queryClient і за допомогою методу invalidateQueries можна зробити недійсним кеш запиту react для певного запиту або декількох запитів.

На рисунках 4.13 – 4.17 представлено результати створення мутацій для системи аналізу та бронювання доступних номерів. Повний лістинг файлів мутації представлено в додатку В.

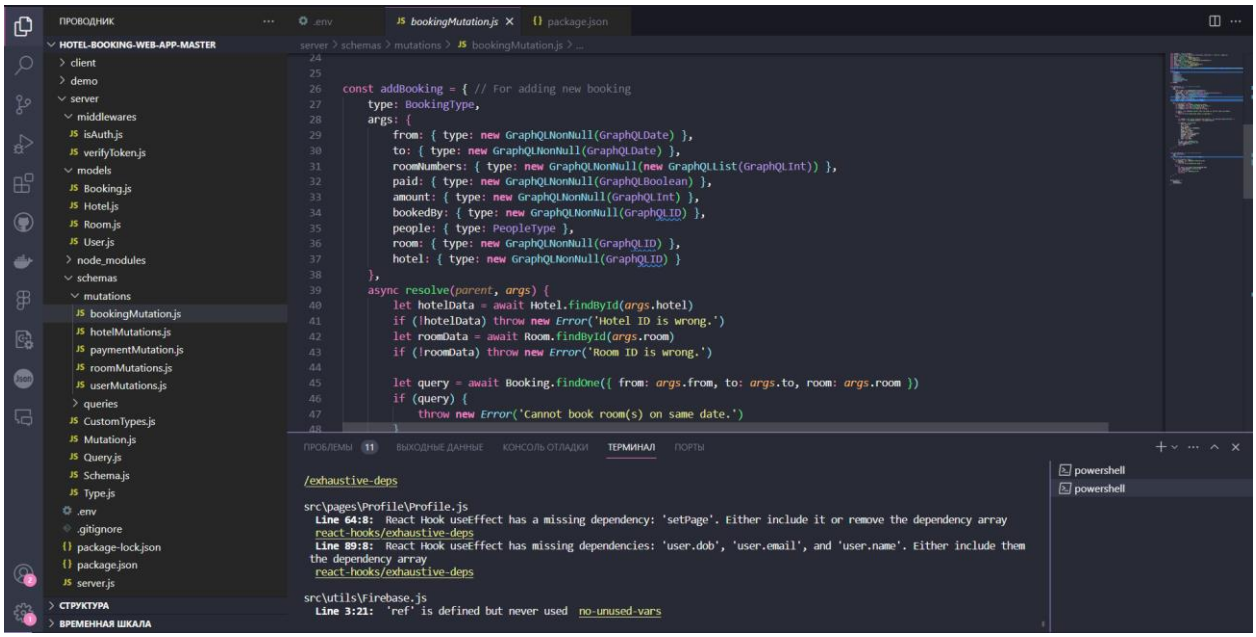

Рисунок 4.13 – Результат створення мутації bookingMutation

Джерело: побудовано автором

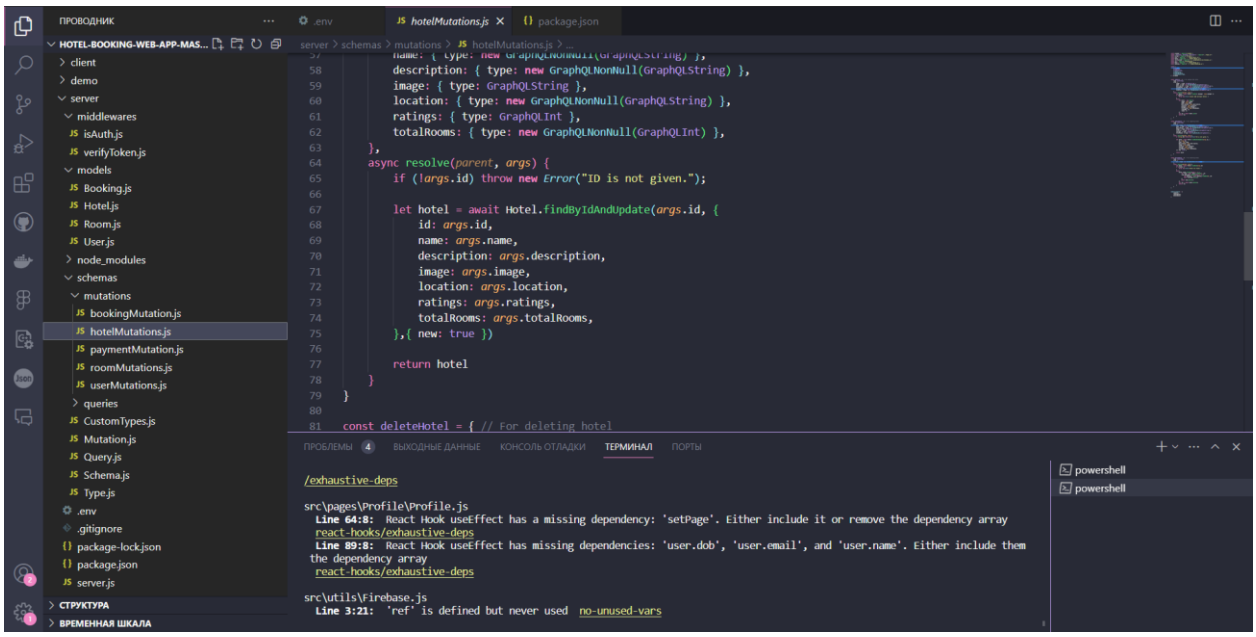

Рисунок 4.14 – Результат створення мутації hotelMutation

Джерело: побудовано автором

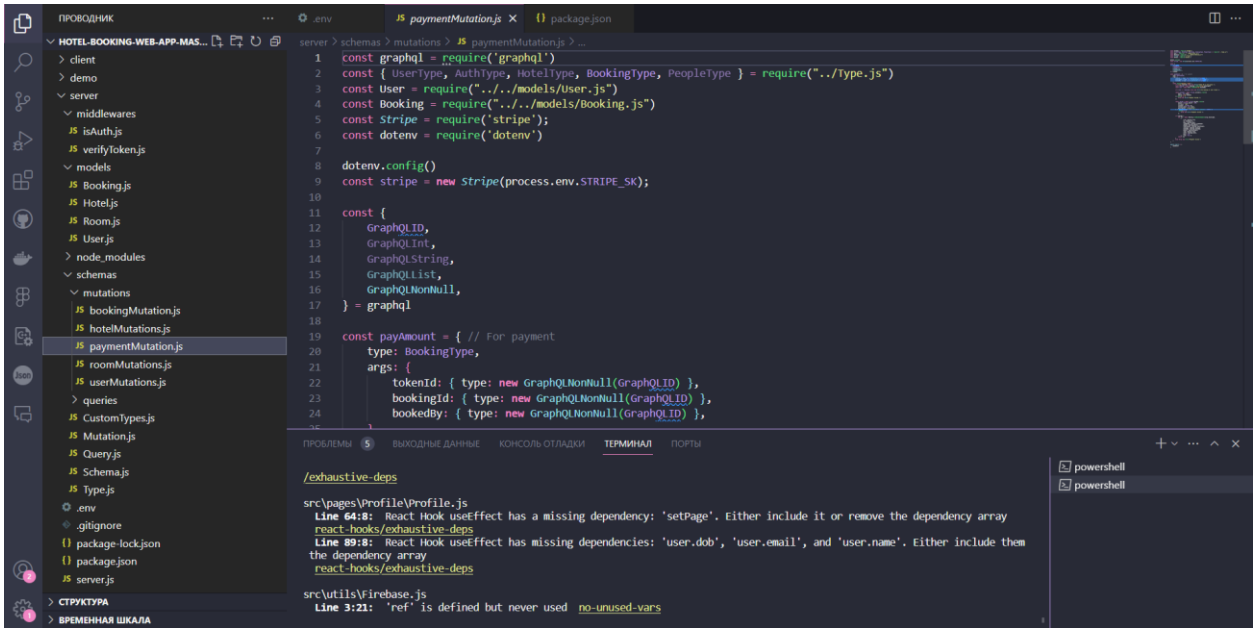

Рисунок 4.15 – Результат створення мутації paymentMutation

Джерело: побудовано автором

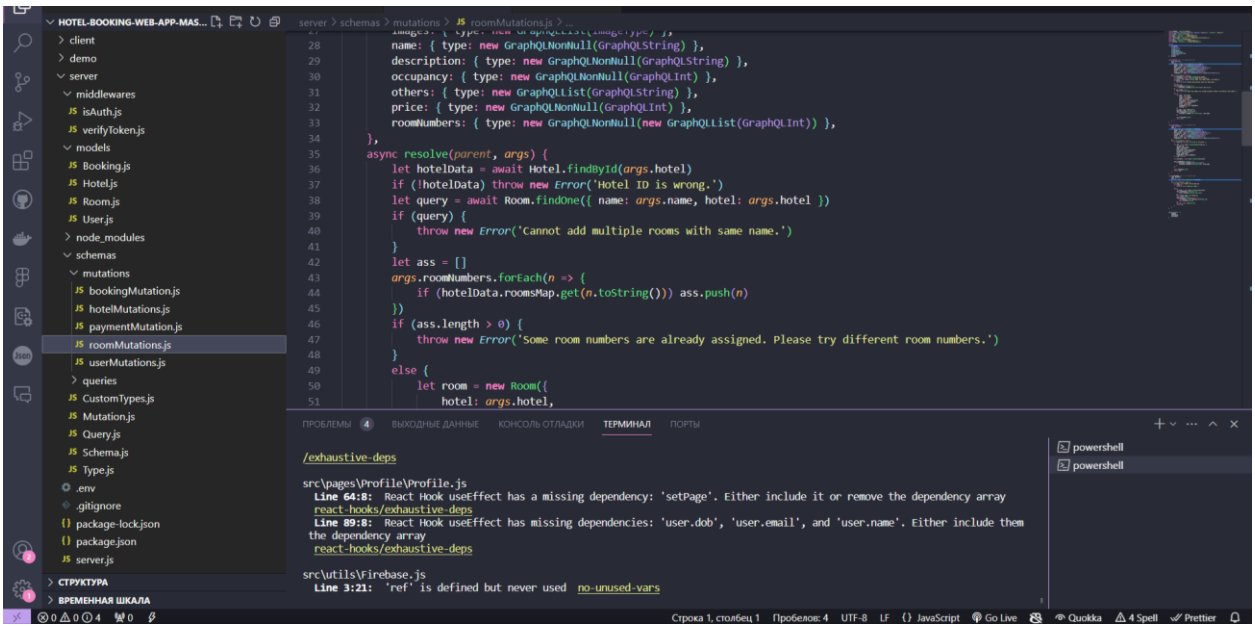

Рисунок 4.16 – Результат створення мутації roomMutation

Джерело: побудовано автором

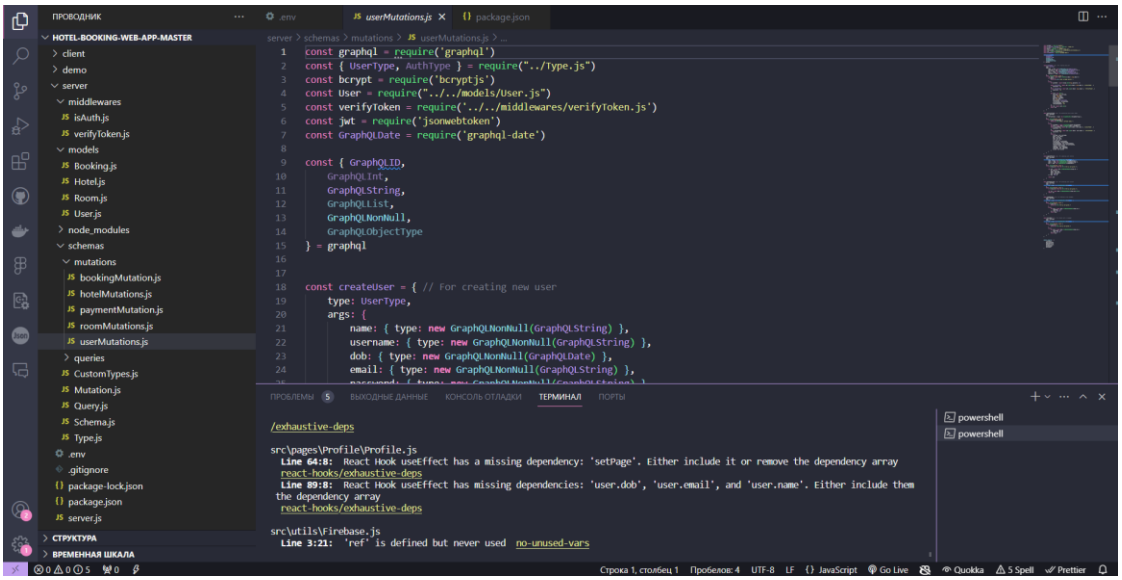

Рисунок 4.17 – Результат створення мутації hotelMutation

Джерело: побудовано автором

4.4 Реалізація клієнтської частини web-системи

Для створення клієнтської частини web-системи було використано пакетний менеджер npm для створення компонентів та маршрутизаторів. Для системи хешування особистих даних та генерації унікального ключа доступу було використано технології jsonwebtoken.

На рисунку 4.18 представлено результат додавання пакетів для створення клієнтської частини в файл package.json.

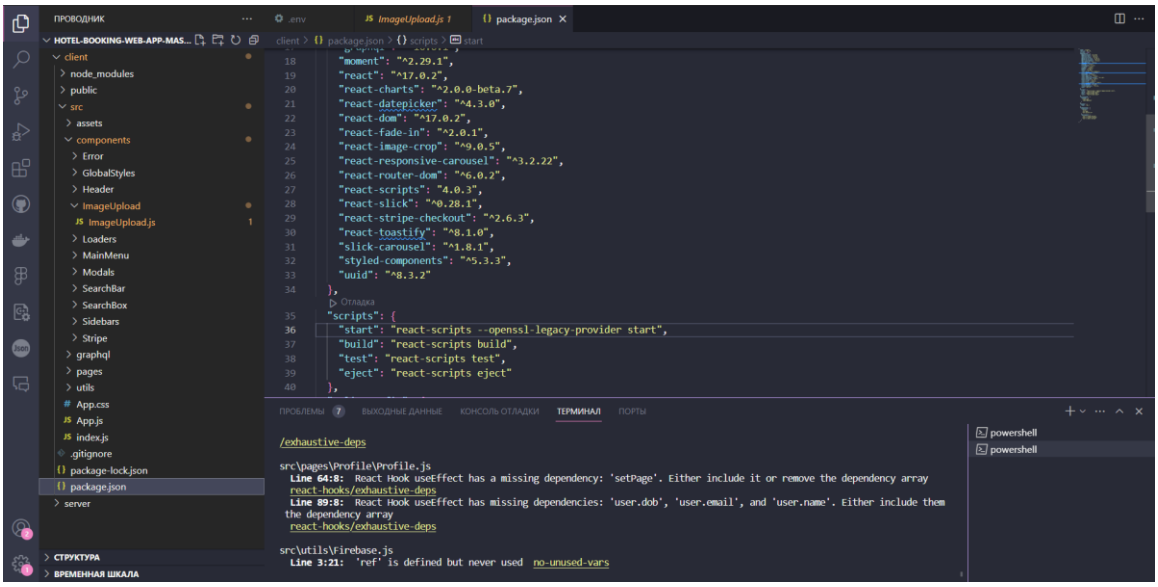

Рисунок 4.18 – Результат додавання пакетів для створення клієнтської частини

Джерело: побудовано автором

Також було створено систему завантаження фото для подачі візуальної інформації про готель або номер. На рисунку 4.19 представлено результат створення файлу ImageUpload.js.

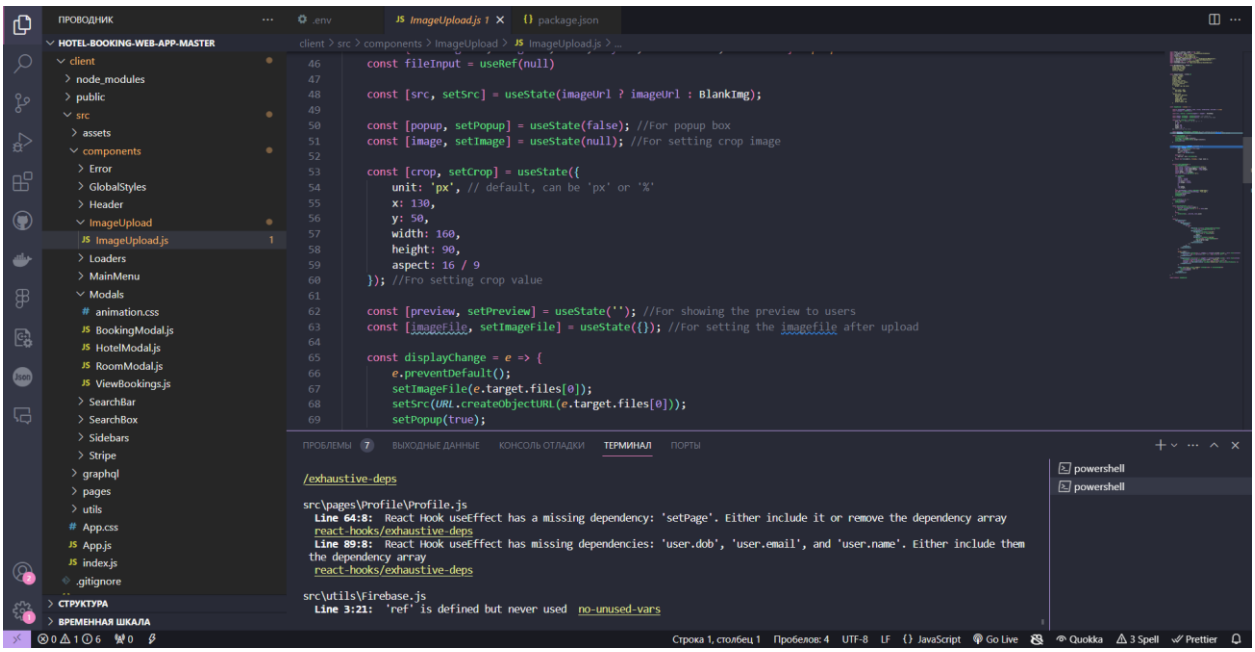

Рисунок 4.19 – Результат створення файлу ImageUpload.js

Джерело: побудовано автором

Для відображення інформації про доступні номери та готелі було створено модальні компоненти які виводили інформацію з серверу з інтуїтивно зрозумілим інтерфейсом. На рисунках 4.20 – 4.23 представлено результати створення модальних вікон. Лістинг коду BookingModal.js та HotelModal.js представлено в додатку Г. Перелік всіх створених компонентів для інтуїтивно зрозумілого інтерфейсу представлено на рисунку 4.24.

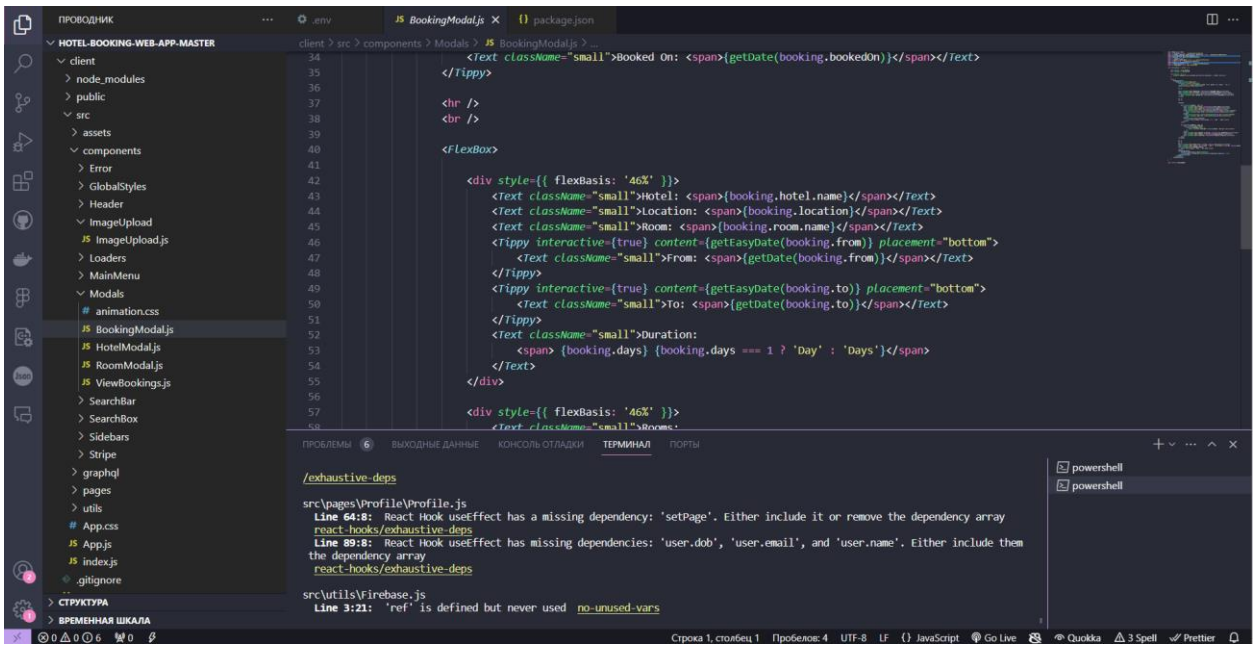

Рисунок 4.20 – Результат створення модального компоненту BookingModal.js

Джерело: побудовано автором

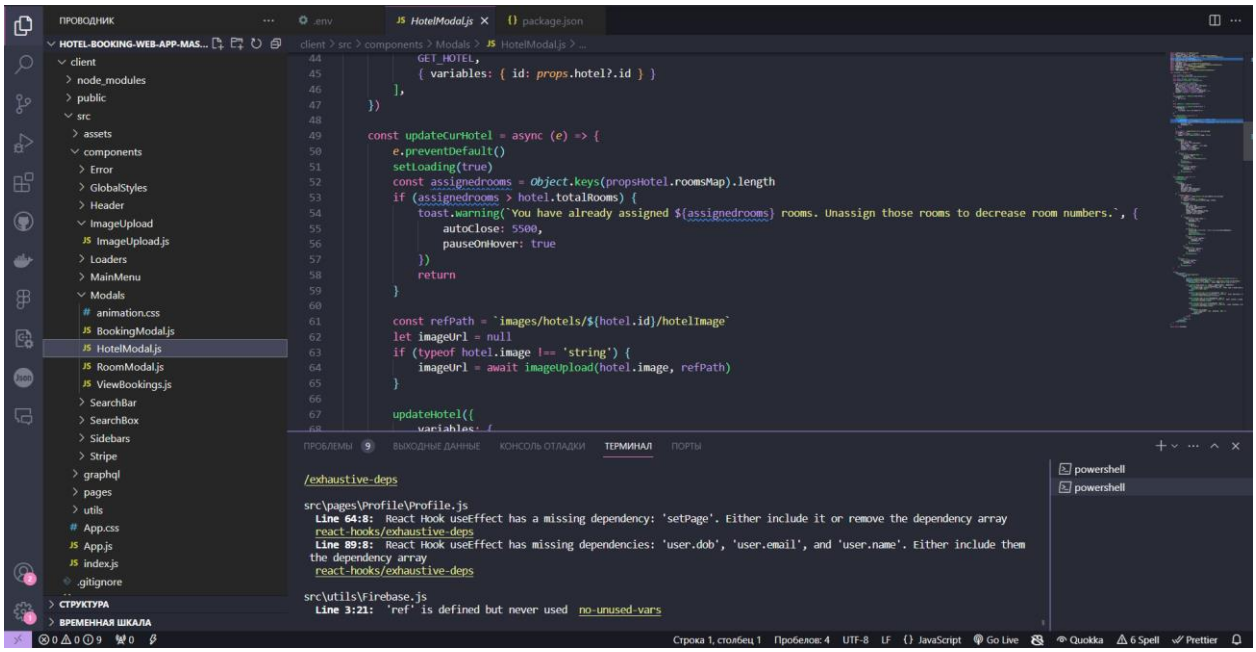

Рисунок 4.21 – Результат створення модального компоненту HotelModal.js

Джерело: побудовано автором

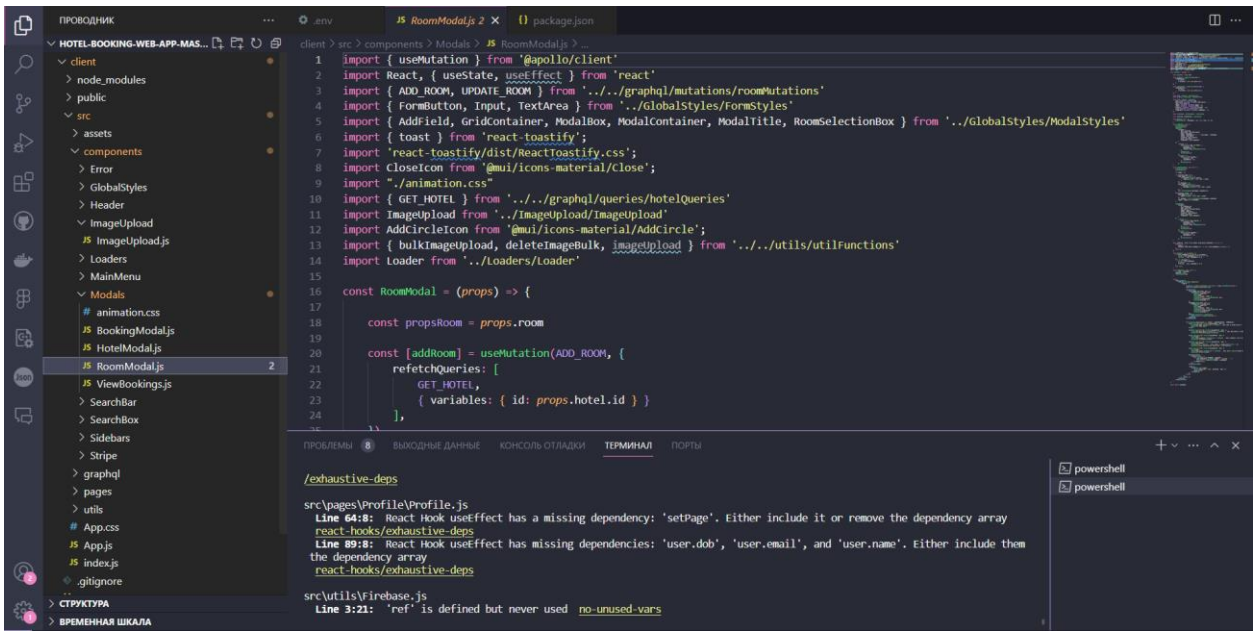

Рисунок 4.22 – Результат створення модального компоненту RoomModal.js

Джерело: побудовано автором

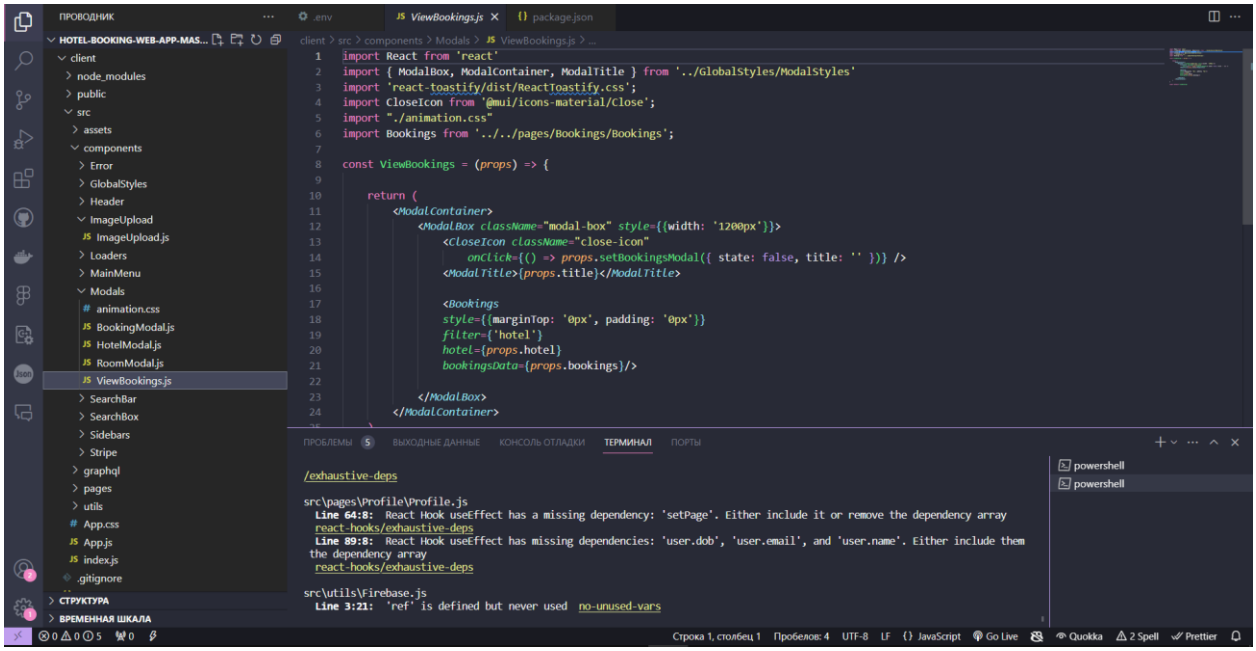

Рисунок 4.23 – Результат створення модального компоненту ViewModal.js

Джерело: побудовано автором

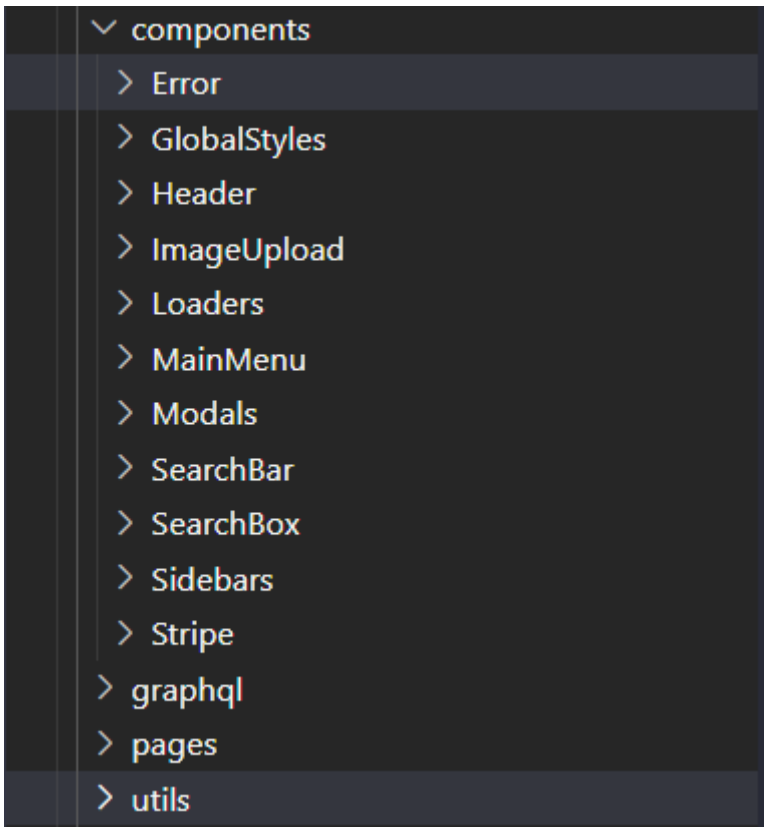

Рисунок 4.24 – Результат створення всіх компонентів

Джерело: побудовано автором

4.5 Настанови з використання

Для початку роботи з web-системою необхідно авторизуватися в останній. Якщо користувач не має облікового запису в системі, йому необхідно виконати реєстрацію особистої сторінки. На рисунках 4.25 – 4.26 представлено вигляд сторінок авторизації та реєстрації.

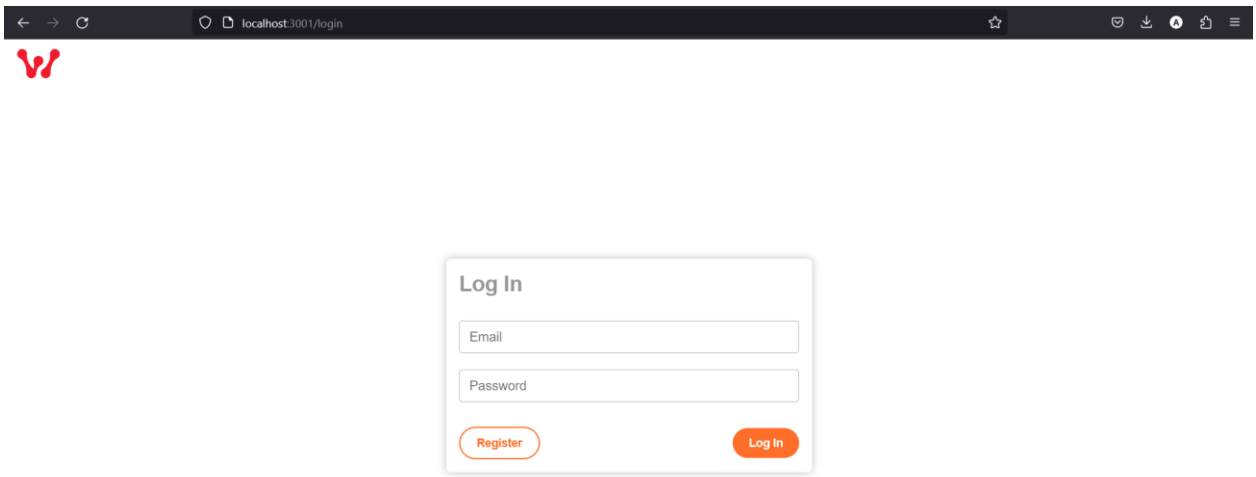

Рисунок 4.25 – Вигляд сторінки авторизації

Джерело: побудовано автором

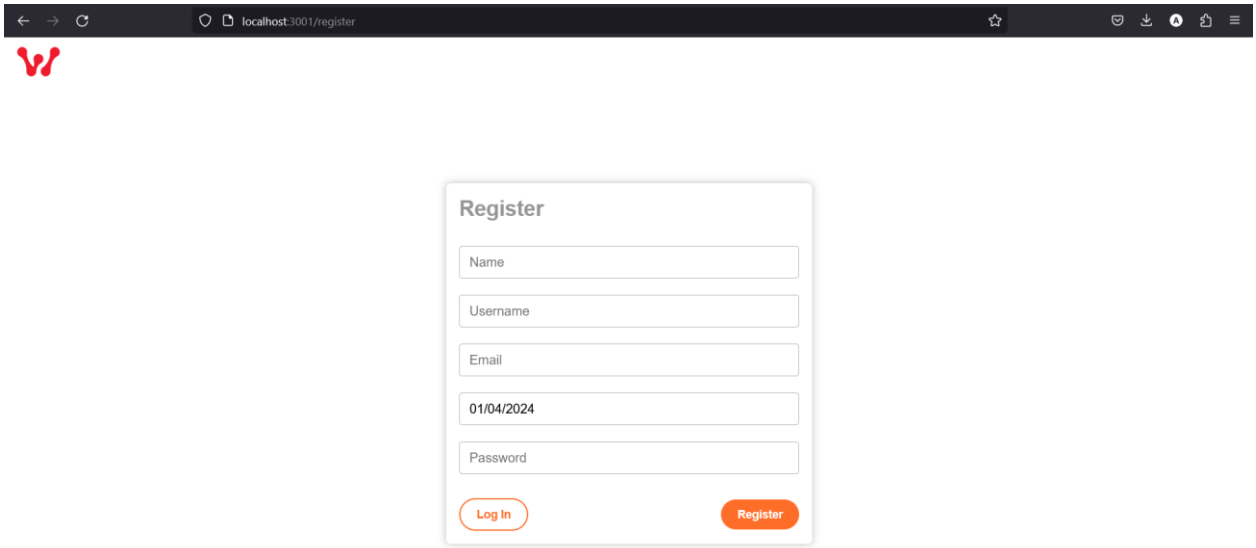

Рисунок 4.26 – Вигляд сторінки реєстрації

Джерело: побудовано автором

Після виконання авторизації чи реєстрації користувач потрапляє на головну сторінку, з якої одразу можна перейти до пошуку готелю. Форма пошуку містить поле для вводу назви локації, де повинен знаходитися готель, дату початку та кінця бронювання, а також кількість дітей з дорослими для заселення. На рисунку 4.27 представлено вигляд головної сторінки web-системи.

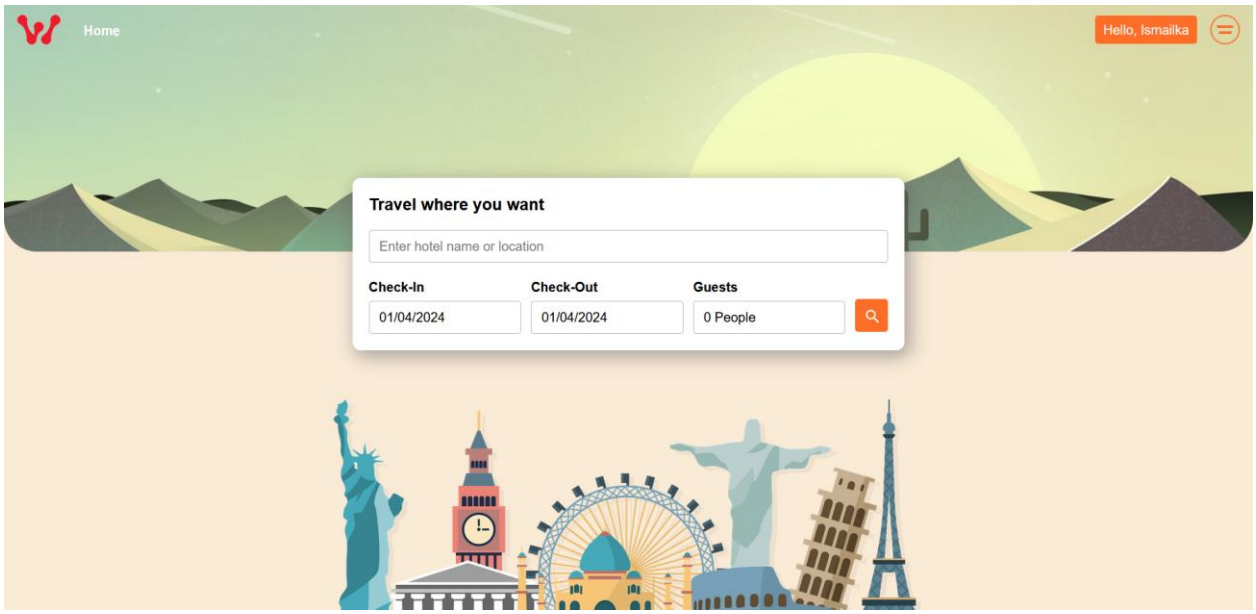

Рисунок 4.27 – Вигляд головної сторінки веб-системи

Джерело: побудовано автором

Після заповнення форми, користувач отримує всі доступні готелі з інформацією про доступні номери та ціну за бронювання. На рисунку 4.28 представлено вигляд результату пошуку готелів за назвою міста «Madrid».

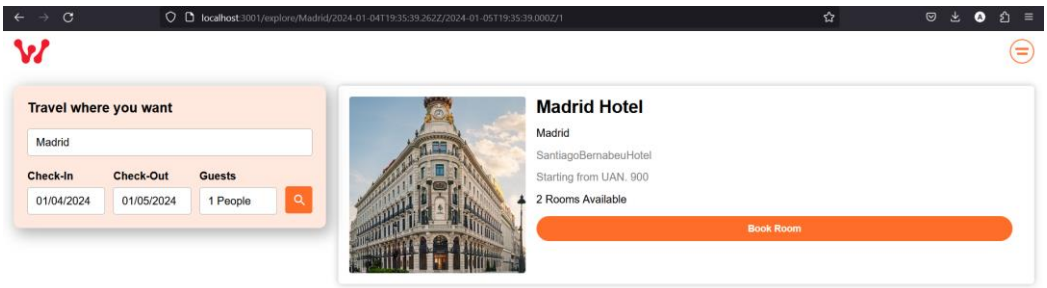

Рисунок 4.28 – Результат пошуку готелів

Джерело: побудовано автором

Для отримання більш детальної інформації про готель, користувачу необхідно натиснути на кнопку «Book Room». На сторінці готелю користувач матиме змогу отримати інформацію про назву готелю, його місцезнаходження, ціну за ніч, та контакти менеджера готелю (рис 4.29).

Якщо користувачу сподобався обраний готель, він також має змогу оформити бронювання кімнати з вибором власних вподобань (рис. 4.30).

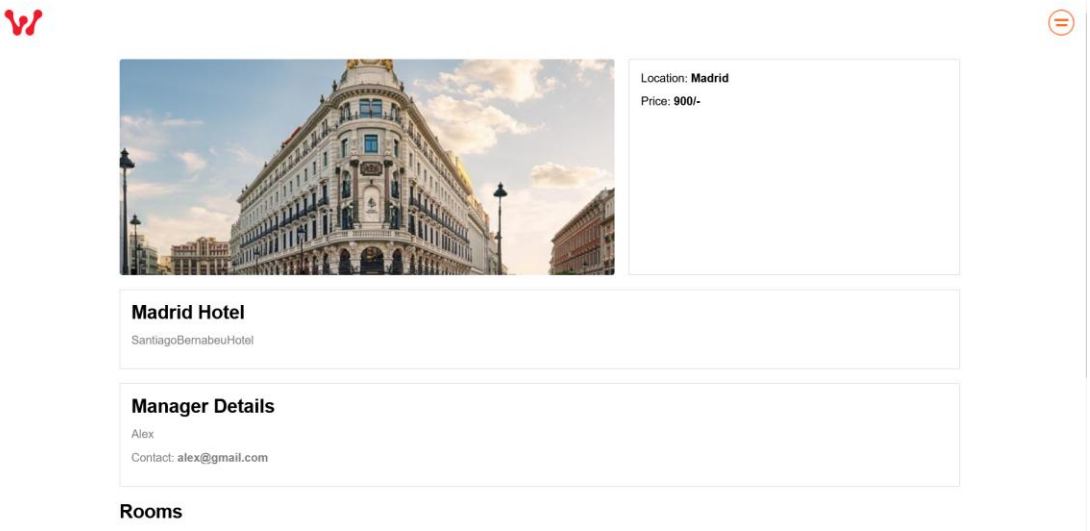

Рисунок 4.29 – Вигляд сторінки з інформацією про готель

Джерело: побудовано автором

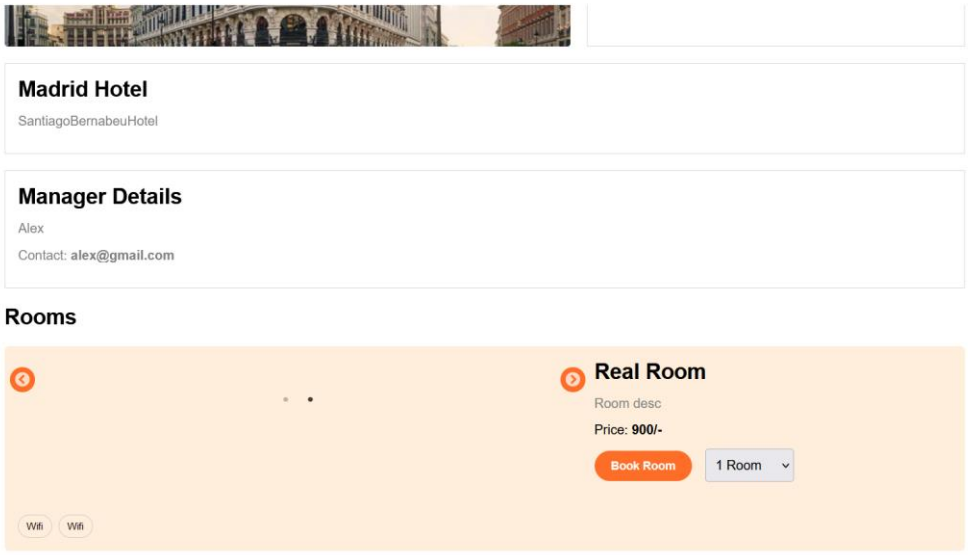

Рисунок 4.30 – Секція з бронюванням номеру обраного готелю

Джерело: побудовано автором

Після того, як користувач натиснув на кнопку «Book Room», системою буде виконано переадресацію на сторінку з підтвердженням замовлення, на якій користувач зможе перевірити правильність власного замовлення та обрати тип оплати (рис. 4.31).

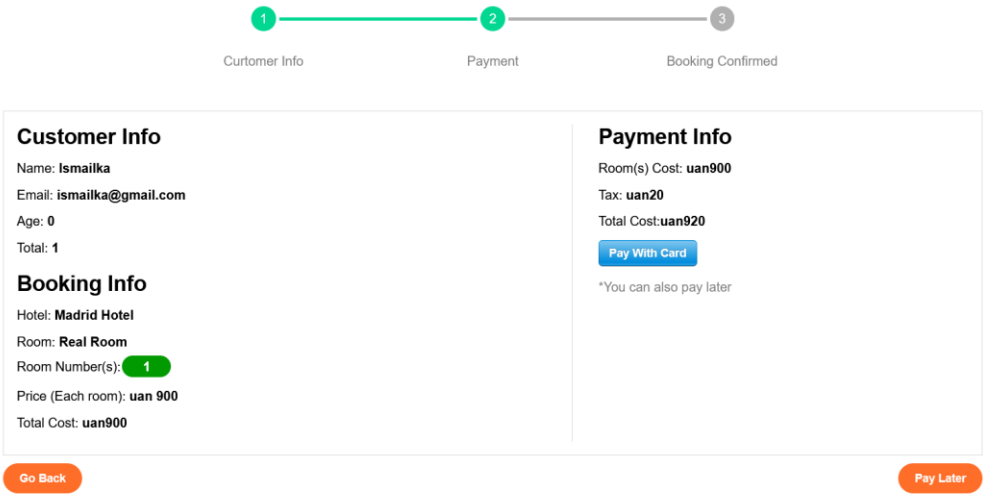

Рисунок 4.31- Вигляд сторінки з оплатою та інформацією про замовлення

Джерело: побудовано автором

Якщо користувач перевірив коректність замовлення, то система надасть йому повідомлення про успішне замовлення. На рисунку 4.32 представлено результат успішного замовлення.

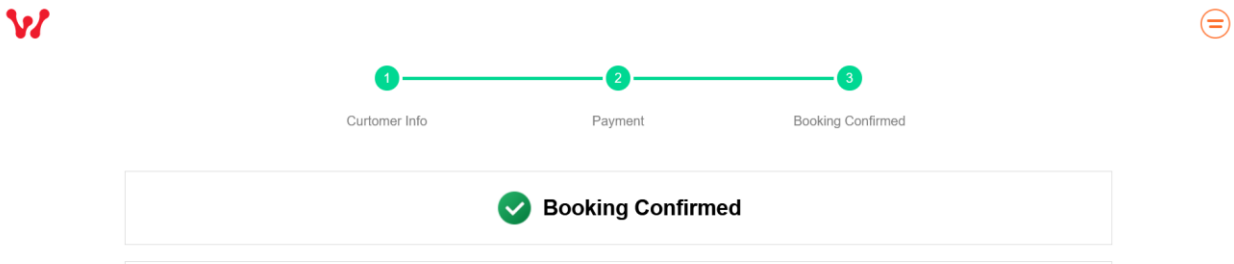

Рисунок 4.32 – Повідомлення про успішне замовлення

Джерело: побудовано автором

Також система надає можливість користувачу виконувати навігацію по сайту. На рисунку 4.33 представлено вигляд навігаційного меню.

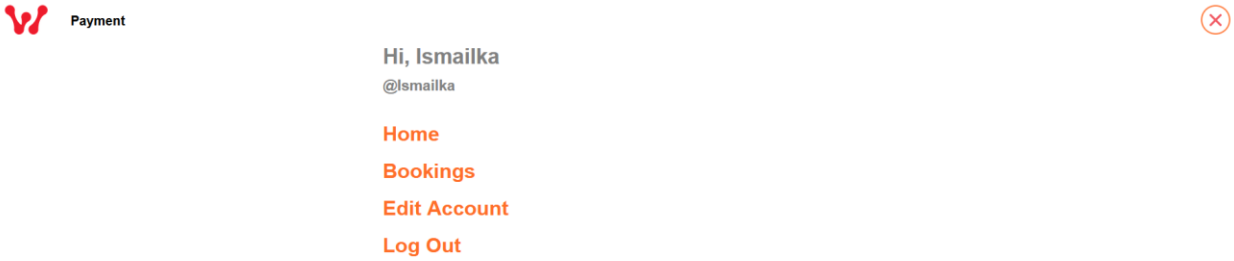

Рисунок 4.33 – Вигляд навігаційного меню

Джерело: побудовано автором

На сторінці «Bookings» користувач може переглянути власні замовлення, виконати операції з їх видаленням та виконати оплату, якщо ще цього не було зроблено. На рисунку 4.34 представлено вигляд сторінки «Bookings».

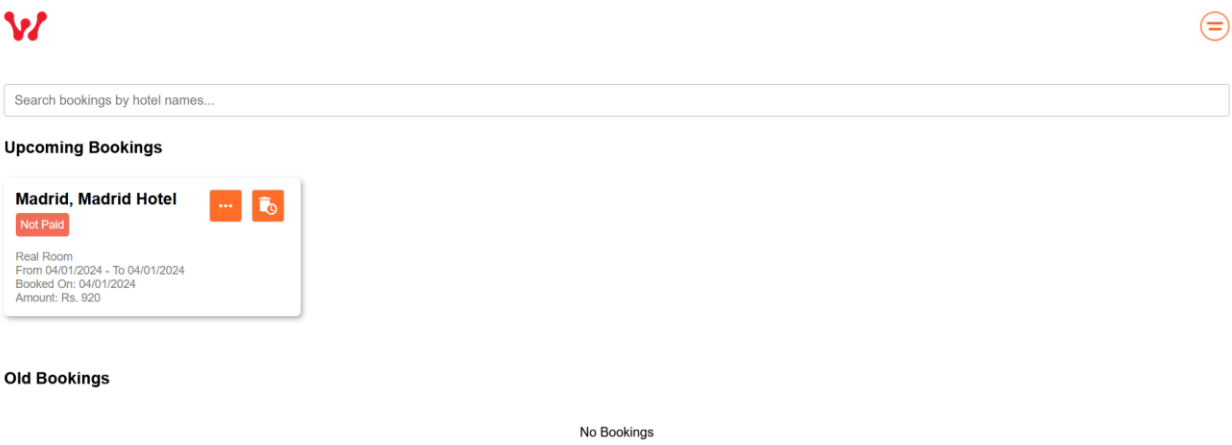

Рисунок 4.34 – Вигляд сторінки «Bookings»

Джерело: побудовано автором

Також користувач може переглянути та змінити власну інформацію, а також додати власний готель та приймати замовлення від інших користувачів. На рисунках 4.35 – 4.36 представлено вигляд сторінок з особистою інформацією та форму з додаванням власного готелю.

Рисунок 4.35 – Вигляд сторінки з особистою інформацією

Джерело: побудовано автором

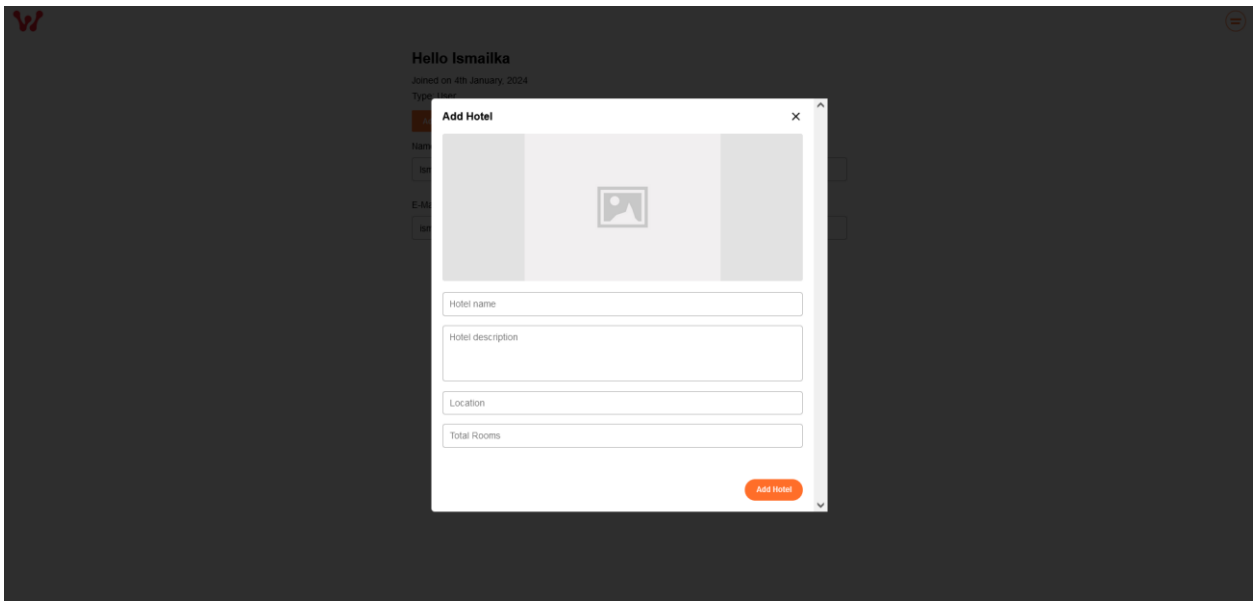

Рисунок 4.36 – Вигляд форми з додаванням власного готелю

Джерело: побудовано автором

ВИСНОВКИ

Результатом виконання кваліфікаційної роботи магістра є розроблена веб-система для аналізу та бронювання готельних номерів. Досліджувана технологія базується на використанні передових веб-технологій для забезпечення комфортної та ефективної взаємодії між користувачем та платформою.

Система здатна надавати швидкі та безперервні послуги користувачам за допомогою MongoDB, Express.js, React.js та Node.js. Використовуючи ці технології, можна реалізувати такі функції, як аналіз доступності, швидке бронювання та надання актуальної інформації про наявність номерів та ціни. Важливим аспектом є використання методів шифрування для забезпечення захисту персональних даних користувачів, що гарантує конфіденційність та безпеку інформації.

Аналіз існуючих інструментів бронювання готелів показує, що вони потребують адаптації та вдосконалення для задоволення мінливих вимог користувачів. Використовуючи структурно-функціональне моделювання, було визначено основні процеси, що забезпечуються веб-системою.

Таким чином, запропонована система для аналізу та бронювання готельних номерів є придатною і має великий потенціал для покращення взаємодії з користувачами, забезпечення безпеки даних та підвищення зручності використання. Впровадження системи сприятиме ефективній організації процесу бронювання номерів у готелях та підвищенню рівня задоволеності клієнтів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Smith, J. (2022). "React Native Mobile App Development: Best Practices for Performance Optimization." Web Development Journal, 45(3), 112-128.

2. Johnson, A. (2022). "Responsive Web Design: Techniques for Cross-Browser Compatibility." UX & UI Design Quarterly, 18(2), 55-70.

3. Brown, M. (2022). "Enhancing User Experience through Progressive Web Applications." Mobile Technology Innovations, 12(4), 189-204.

4. Top 10 Benefits of React.js for Application Development [Електронний ресурс] – Режим доступу до ресурсу: https://ncube.com/top-10-benefits-of-react-js-for-application-development/ (Дата звернення 09.11.2023).

5. React vs Angular: Which JS Framework to choose for Front-end Development? [Електронний ресурс] – Режим доступу до ресурсу: https://radixweb.com/blog/react-vs-angular (Дата звернення 09.11.2023).

6. 8 Benefits of ReactJS: Is It Worth Using in Your Project? [Електронний ресурс] – Режим доступу до ресурсу: https://procoders.tech/blog/advantages-of-using-reactjs/ (Дата звернення 09.11.2023).

7. Top 10 benefits and advantages of a hotel management system [Електронний ресурс] – Режим доступу до ресурсу: https://www.mews.com/en/blog/advantages-of-hotel-management-system (Дата звернення 09.11.2023).

8. Система аналізу пошукових запитів GoogleTrends [Електронний ресурс] – Режим доступу до ресурсу: https://trends.google.com.ua/trends/explore?geo=UA&q=%2Fm%2F0yxzc1z&hl=ua (Дата звернення 09.11.2023).

9. What are the MERN stack’s advantages? [Електронний ресурс] – Режим доступу до ресурсу: https://www.bocasay.com/what-mern-stacks-advantages/ (Дата звернення 09.11.2023).

10. Benefits of using Express for web development [Електронний ресурс] – Режим доступу до ресурсу: https://www.tempest.house/blog-posts/benefits-of-using-express-for-web-development (Дата звернення 09.11.2023).

11. REST API documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/docs/en/intelligent-promising?topic=apis-rest-api-documentation (Дата звернення 09.11.2023).

12. GitHub REST API documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.github.com/en/rest?apiVersion=2022-11-28 (Дата звернення 09.11.2023).

13. MERN in client-server architecture [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/mern-stack/ (Дата звернення 11.11.2023).

14. Understanding IDEF Diagram: An In-depth Look [Електронний ресурс] – Режим доступу до ресурсу: https://boardmix.com/knowledge/idef-diagram/ (Дата звернення 11.11.2023).

15. Olayele Adelakun, Sergio Galvan-Cruz & Fen Wang. Impacts of IDEF0-Based Models on the Usefulness, Learning, and Value Metrics of Scrum and XP Project Management Guides [Електронний ресурс] – Режим доступу до ресурсу: https://www.tandfonline.com/doi/abs/10.1080/10429247.2021.1958631 (Дата звернення 11.11.2023).

16. What Is Functional Decomposition? [Електронний ресурс] – Режим доступу до ресурсу: https://www.baeldung.com/cs/functional-decomposition (Дата звернення 11.11.2023).

17. What is Use Case Diagram? [Електронний ресурс] – Режим доступу до ресурсу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/ (Дата звернення 11.11.2023).

18. NoSQL Databases: Advantages and Disadvantages [Електронний ресурс] – Режим доступу до ресурсу: https://www.dataversity.net/nosql-databases-advantages-and-disadvantages/ (Дата звернення 11.11.2023).

19. Top 12 Database Design Principles in 2023 [Електронний ресурс] – Режим доступу до ресурсу: https://vertabelo.com/blog/database-design-principles/ (Дата звернення 11.11.2023).

20. Why Use MongoDB: What It Is and What Are the Benefits [Електронний ресурс] – Режим доступу до ресурсу: https://www.simplilearn.com/tutorials/mongodb-tutorial/what-is-mongodb (Дата звернення 11.11.2023).

21. MongoDB Atlas [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/cloud/atlas (Дата звернення 11.11.2023).

22. An introduction to the NPM package manager [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/en/learn/getting-started/an-introduction-to-the-npm-package-manager (Дата звернення 11.11.2023).

23. Introduction to Mutations [Електронний ресурс] – Режим доступу до ресурсу: https://tanstack.com/query/v4/docs/react/guides/mutations (Дата звернення 11.11.2023).

24. What Is SMART in Project Management? [Електронний ресурс] – Режим доступу до ресурсу: https://www.wrike.com/project-management-guide/faq/what-is-smart-in-project-management/ (Дата звернення 11.11.2023).

25. How to Define S.M.A.R.T. Goals in Project Management [Електронний ресурс] – Режим доступу до ресурсу: https://plaky.com/blog/smart-goals-project-management/ (Дата звернення 11.11.2023).

26. What is a Work Breakdown Structure? [Електронний ресурс] – Режим доступу до ресурсу: https://www.workbreakdownstructure.com/ (Дата звернення 11.11.2023).

27. How to Create an Effective Work Breakdown Structure? [Електронний ресурс] – Режим доступу до ресурсу: https://www.edvantis.com/blog/work-breakdown-structure/ (Дата звернення 11.11.2023).

28. A complete guide to SMART goals: everything you need to know [Електронний ресурс] – Режим доступу до ресурсу: https://monday.com/blog/project-management/smart-goals/ (Дата звернення 11.11.2023).

29. What Is Work Breakdown Structure in Project Management? [Електронний ресурс] – Режим доступу до ресурсу: https://www.wrike.com/project-management-guide/faq/what-is-work-breakdown-structure-in-project-management/ (Дата звернення 11.11.2023).

30. 7 steps to define Project's Organization Breakdown Structure (OBS) [Електронний ресурс] – Режим доступу до ресурсу: https://www.wrenchsp.com/define-project-organization-breakdown-structure-obs/ (Дата звернення 11.11.2023).

31. What is Gantt Diagram and its benefits for project management [Електронний ресурс] – Режим доступу до ресурсу: https://instagantt.com/project-management/what-is-gantt-diagram (Дата звернення 11.11.2023).

32. What Is Project Risk Management? [Електронний ресурс] – Режим доступу до ресурсу: https://blog.hubspot.com/the-hustle/project-risk-management (Дата звернення 11.11.2023).

33. What is Risk Management in Projects? [Електронний ресурс] – Режим доступу до ресурсу: https://projectriskcoach.com/benefits-of-risk-management-in-projects/ (Дата звернення 11.11.2023).

ДОДАТОК А

Деталізація мети проекту методом SMART

SMART – це мнемонічна абревіатура, яка встановлює критерії ідеальних цілей і завдань проекту. SMART означає конкретні, вимірювані, досяжні, актуальні та обмежені в часі [24].

– Конкретність – мета чітко сформульована, щоб кожен, хто її читає, міг зрозуміти, що буде зроблено і хто буде це робити.

– Вимірювана – мета включає в себе те, як буде вимірюватися дія. Вимірювання ваших цілей допомагає вам визначити, чи досягаєте ви прогресу. Це тримає вас на правильному шляху і в графіку.

– Досяжна – мета є реалістичною з огляду на реалії, з якими стикається громада. Постановка розумних цілей допомагає налаштувати проект на успіх

– Релевантна – релевантна мета має сенс, тобто вона відповідає меті гранту, культурі та структурі громади, а також враховує культурі та структурі громади, а також відповідає баченню проекту.

– Часові рамки – кожна ціль має конкретні часові рамки для завершення. Це означає, що мета повинна відповідати цим критеріям, щоб вважатися SMART-ціллю.

SMART-цілі допомагають керівникам проектів, бізнес-менеджерам і будь-яким іншим типам керівників команд визначити чіткі завдання, які повинні бути виконані їхніми командами. Найкраще те, що SMART-цілі можна використовувати для вимірювання ефективності практично будь-якого проекту або завдання [25].

Отже, можемо сформулювати мету нашого проекту за цими п’ятьма факторами. Результати наведені у таблиці Б.1.

Таблиця Б.1 – Формалізація мети за технологією SMART

Specific	Розробити web-орієнтовану систему, яка надає можливість клієнтам швидко та зручно аналізувати доступні номери в готелі, а також забезпечує аналіз зайнятості та статистику.
Measurable	Результат допоможе Забезпечити збільшення кількості успішних онлайн бронювань на 20% протягом перших шести місяців експлуатації системи.
Achievable	Проєкт реалізовується у відповідності до рівня досвіду та на основі затвердженого ТЗ.
Revelant	Результат сприятиме автоматизації процесів готельного бізнесу та покращить обслуговування клієнтів, що відповідає стратегії підвищення ефективності.
Time-bound	Проєкт виконується враховуючи встановлені на ранньому етапі обмеження в часі (грудень 2023).

Структура розподілу робіт (WBS)

WBS – це ієрархічна декомпозиція всього обсягу робіт, необхідних для завершення проекту. Вона розбиває проект на менші, більш керовані компоненти і забезпечує чітке і структуроване уявлення про роботу, необхідну для завершення проекту [26].

Важливість WBS в управлінні проектами багатогранна. Ось кілька ключових причин, чому WBS має вирішальне значення для ефективного управління проектами:

Допомагає з плануванням проекту: WBS є ключовим інструментом у плануванні проекту. Він допомагає менеджерам проектів розбивати складні проекти на менші, більш керовані компоненти, що полегшує планування та управління проектом. Розбиваючи проект на менші, більш керовані компоненти, менеджери проектів можуть визначити необхідну роботу, необхідні ресурси та терміни завершення [27].

Полегшує розподіл ресурсів: Маючи чітке і структуроване уявлення про роботу, необхідну для завершення проекту, WBS допомагає менеджерам проектів більш ефективно розподіляти ресурси. Вона допомагає визначити ресурси, необхідні на кожному рівні проекту, що полегшує управління ресурсами та забезпечує їх належний розподіл [28].

Полегшує моніторинг і контроль проекту: WBS надає чітке і структуроване уявлення про проект, що полегшує менеджерам проекту моніторинг прогресу і контроль над проектом. Вона забезпечує чітку основу для відстеження прогресу відповідно до етапів, виявлення проблем і ризиків, а також внесення необхідних коригувань [29].

На рисунку Б.1 представлено WBS-структуру робіт проекту

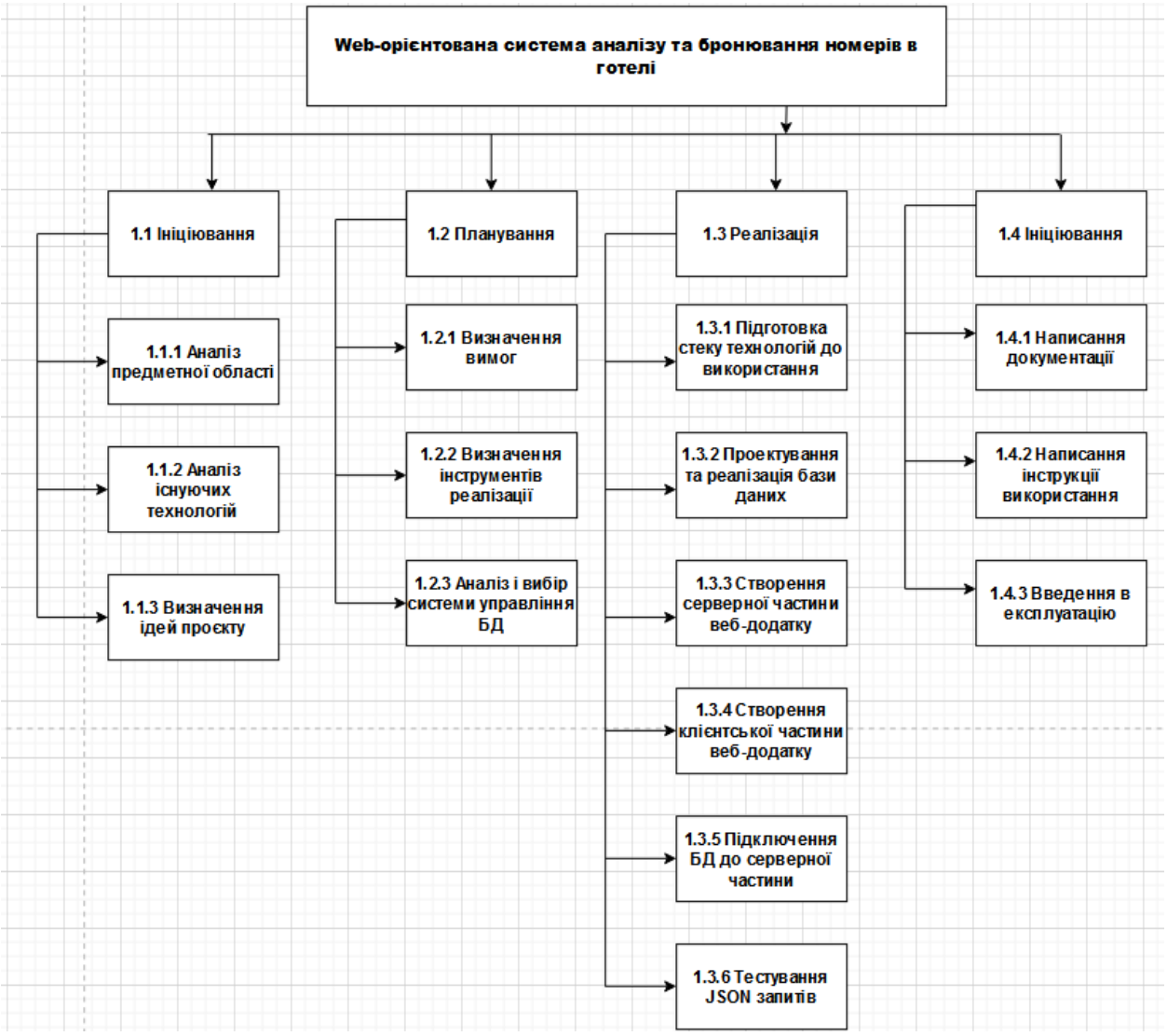

Рисунок Б.1 – WBS-структура робіт проекту

Джерело: розроблено автором

Організаційна структура робіт (OBS)

Організаційна структура – це спосіб, у який створена компанія, організація чи команда. Вона може бути ієрархічною, з різними рівнями управління. Або вона може бути дивізіональною, з різними продуктовими лініями та підрозділами.

Організаційні структури є важливими, оскільки вони допомагають бізнесу впроваджувати ефективні процеси прийняття рішень. Призначаючи

спеціалізовані ролі працівникам нижчого рівня, бізнес може швидше приймати кращі рішення [30].

Крім того, організаційні структури забезпечують чітку організаційну схему, яка допомагає бізнесу відстежувати свої людські ресурси.

На рисунку Б.2 представлено OBS-структуру робіт проекту

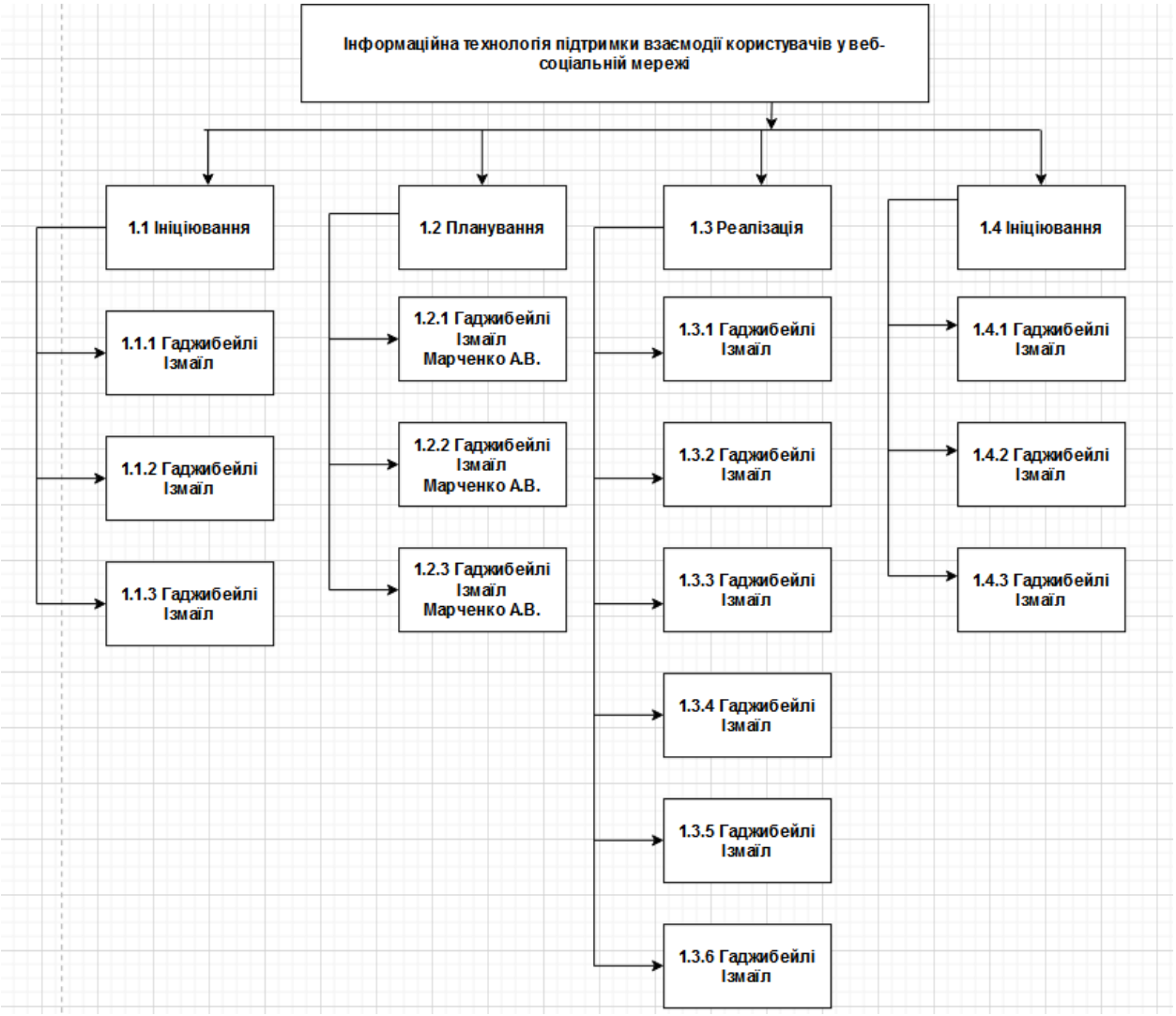

Рисунок Б.2 – OBS-структура робіт проекту

Джерело: розроблено автором

Діаграма Ганта

Створення календарного плану (діаграми Ганта) є одним з найважливіших етапів планування проекту і схоже на робочу програму з реалістичним розподілом дат. Це допомагає забезпечити врахування тривалості процесів з обмеженими ресурсами, включаючи вихідні та святкові дні [31].

Календарний план проекту показаний на рисунках В.3 - В.4.

Web-орієнтована система аналізу та бронювання номерів в готелі	58 дней	Вт 12.09.23	Пт 01.12.23				
Ініціалізація проекту	2 дней	Вт 12.09.23	Ср 13.09.23				
Опрашування ТЗ	1 день	Чт 14.09.23	Чт 14.09.23			Гаджибейлі Ісмаїл	
Аналіз сервісів	1 день	Пт 15.09.23	Пт 15.09.23	3		Гаджибейлі Ісмаїл	$10000
Пошук контенту	1 день	Сб 16.09.23	Сб 16.09.23	4		Гаджибейлі Ісмаїл	$1,116.00
Ідентифікація проекту	1 день	Вс 17.09.23	Вс 17.09.23	5		Марченко А.В.	$72.00
Планування проекту	**15 дней**	Пн 18.09.23	Пт 06.10.23	6			$72.00
Структура проекту	2 дней	Ср 18.10.23	Чт 19.10.23	7		Гаджибейлі Ісмаїл	$72.00
Розробка дизайну та макету web-орієнтованої системи аналізу та бронювання номерів в готелі	2 дней	Пт 20.10.23	Пн 23.10.23	8		Гаджибейлі Ісмаїл	$900.00
Розробка календарного плану «Розробка Web-орієнтованої системи аналізу та бронювання номерів в готелі»	5 дней	Пн 23.10.23	Пт 27.10.23	9		Гаджибейлі Ісмаїл	$1,432.00
Управління ресурсами	1 день	Сб 28.10.23	Сб 28.10.23	10		Гаджибейлі Ісмаїл	$72.00
Управління фінансами	1 день	Вс 29.10.23	Вс 29.10.23	11		Гаджибейлі Ісмаїл	$72.00
Управління якістю	1 день	Пн 30.10.23	Пн 30.10.23	12		Гаджибейлі Ісмаїл	$100.00
Оптимізація плану	1 день	Вт 31.10.23	Вт 31.10.23	13		Гаджибейлі Ісмаїл	$72.00
Затвердження плану	1 день	Ср 01.11.23	Ср 01.11.23	14		Марченко А.В.	$72.00
Реалізація проекту	15 дней	Вс 01.10.23	Чт 19.10.23			Гаджибейлі Ісмаїл	$7,020.00
Розробка інформаційної технології	5 дней	Ср 01.11.23	Вт 07.11.23			Гаджибейлі Ісмаїл	$3,250.00
Реалізація дизайну та макету	4 дней	Вт 07.11.23	Пт 10.11.23			Гаджибейлі Ісмаїл	$1,300.00

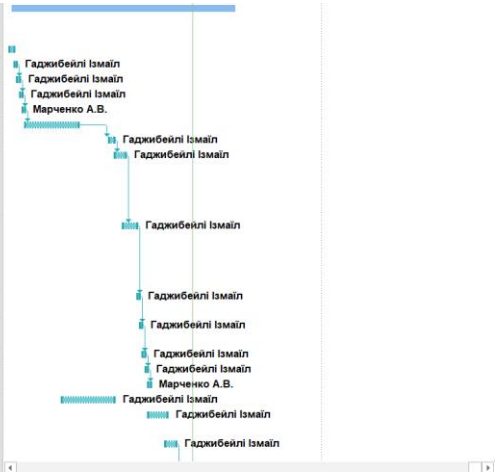

Рисунок В.3 – Діаграма Ганта, частина перша

Джерело: розроблено автором

Оптимізація плану	1 день	Вт 31.10.23	Вт 31.10.23	13		Гаджибейлі Ісмаїл	$72.00
Затвердження плану	1 день	Ср 01.11.23	Ср 01.11.23	14		Марченко А.В.	$72.00
Реалізація проекту	15 дней	Вс 01.10.23	Чт 19.10.23			Гаджибейлі Ісмаїл	$7,020.00
Розробка інформаційної технології	5 дней	Ср 01.11.23	Вт 07.11.23			Гаджибейлі Ісмаїл	$3,250.00
Реалізація дизайну та макету	4 дней	Вт 07.11.23	Пт 10.11.23			Гаджибейлі Ісмаїл	$1,300.00
Розробка Front-end частини	2 дней	Пт 10.11.23	Пн 13.11.23	18		Гаджибейлі Ісмаїл	$1,500.00
Розробка Back-end частини	5 дней	Ср 22.11.23	Вт 28.11.23	19		Гаджибейлі Ісмаїл	$350.00
Проектування БД	3 дней	Ср 29.11.23	Пт 01.12.23	20		Гаджибейлі Ісмаїл	$100.00
Створення компонентів	1 день	Чт 30.11.23	Чт 30.11.23	21		Гаджибейлі Ісмаїл	$100.00
Створення зв'язку між користувачем та додатком	**21 дней**	Пт 01.12.23	Пт 29.12.23			Гаджибейлі Ісмаїл	$3,270.00
Розробка модулю реєстрації нового номеру з готелем	1 день	Сб 02.12.23	Сб 02.12.23			Гаджибейлі Ісмаїл	$270.00
Розробка модулів реєстрації та авторизації нового користувача	1 день	Пн 04.12.23	Пн 04.12.23	24		Гаджибейлі Ісмаїл	$1,500.00
Розробка модулю резервації готелю	2 дней	Вт 05.12.23	Ср 06.12.23	25		Гаджибейлі Ісмаїл	$1,500.00
Реалізація зв'язку БД з додатком	4 дней	Чт 07.12.23	Вт 12.12.23	26		Гаджибейлі Ісмаїл	$500.00
Тестування	2 дней	Ср 13.12.23	Чт 14.12.23	27		Гаджибейлі Ісмаїл	$500.00
Завершення проекту	15 дней	Пт 15.12.23	Чт 04.01.24	28		Гаджибейлі Ісмаїл	$432.00
Створення документації	6 дней	Пт 15.12.23	Пт 22.12.23	29		Гаджибейлі Ісмаїл	$72.00
Створення інструкції користувача	6 дней	Сб 23.12.23	Пт 29.12.23	30		Гаджибейлі Ісмаїл	$72.00
Запуск і підтримка	2 дней	Сб 30.12.23	Пн 01.01.24	31		Гаджибейлі Ісмаїл	$72.00
Архівація проекту	1 день	Вт 02.01.24	Вт 02.01.24	32		Гаджибейлі Ісмаїл	$72.00

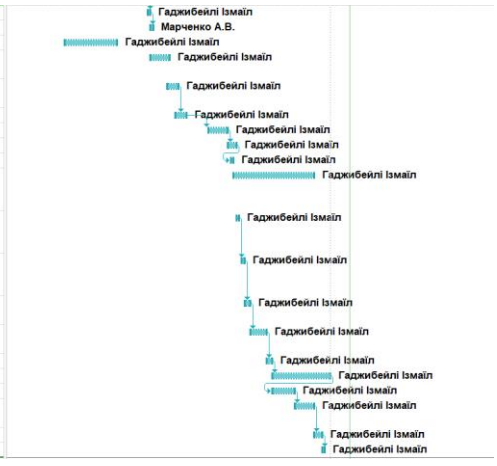

Рисунок В.4 – Діаграма Ганта, частина друга

Джерело: розроблено автором

Управління ризиками

Управління проектними ризиками – це процес виявлення, аналізу та реагування на будь-які ризики, що виникають протягом життєвого циклу проекту, щоб допомогти проекту не відставати від графіка та досягти своєї мети. Управління ризиками не зводиться лише до реагування - воно має бути частиною процесу планування, щоб з'ясувати, який ризик може виникнути в проекті і як контролювати цей ризик, якщо він справді виникне [32].

Ризик - це все, що потенційно може вплинути на терміни, продуктивність або бюджет проекту. Ризики - це потенційні можливості, і в контексті управління проектами, якщо вони стають реальністю, вони класифікуються як "проблеми", які необхідно вирішувати за допомогою плану реагування на ризики. Отже, управління ризиками - це процес виявлення, класифікації, визначення пріоритетів і планування ризиків до того, як вони стануть проблемами [33].

Таблиця А.2. Ймовірність виникнення і величина ризику

№	Ризики	Виникнення	Втрати
1	Кібербезпека та злам системи	2	4
2	Відмова в обслуговуванні (DoS) атаки	3	3
3	Технічні несправності та відмови в роботі	3	5
4	Проблеми з конфіденційністю даних	2	3
5	Неадекватна обробка та зберігання даних	4	5
6	Застарілість та неактуальність інформації	2	3

Джерело: розроблено автором

Таблиця А.3 – Матриця впливу

Вірогідність виникнення	Матриця впливу				
5			3	5	
4		4			
3		6	2		
2				1	
1					
Ступінь впливу	1	2	3	4	5

Джерело: розроблено автором

ДОДАТОК Б

Лістинг коду server.js

```
const express = require('express')
const cors = require('cors')
const {graphqlHTTP} = require('express-graphql')
const mongoose = require('mongoose')
const schema = require("./schemas/Schema.js")
const isAuth = require('./middlewares/isAuth.js')
const dotenv = require('dotenv')

const app = express()
app.use(cors())
app.use(express.json())
app.use(isAuth)
dotenv.config()

const mongoURL = process.env.MONGO_URL
mongoose.connect(mongoURL, {
    useNewUrlParser: true,
    useUnifiedTopology: true
})
mongoose.connection.once('open', () => console.log("DB Connected..."))

app.use("/graphql", graphqlHTTP({
    schema,
    graphiql: true
}))
const port = process.env.PORT || 8080;
app.listen(port, () => console.log("Sever is running..."))
app.get('/', (req,res) => res.send("Auth system..."))
```

ДОДАТОК В

Лістинг коду bookingMutation.js

```
const graphql = require('graphql')
const { UserType, AuthType, HotelType, BookingType, PeopleType } = require("../Type.js")
const bcrypt = require('bcryptjs')
const User = require("../../models/User.js")
const Hotel = require("../../models/Hotel.js")
const verifyToken = require('../../middlewares/verifyToken.js')
const jwt = require('jsonwebtoken')
const Room = require('../../models/Room.js')
const Booking = require("../../models/Booking.js")
const GraphQLDate = require('graphql-date')
const Stripe = require('stripe');
const stripe = new Stripe('sk_test_51Hr13fE7BvSkBO4pKZaivsbiLMdykp67V4jDLs2sb8Gjuu9crHPBs9yHKv59acXvEa89INAoNgL2PXHPOcdGgnDc005AlcrJd1');

const {
    GraphQLID,
    GraphQLInt,
    GraphQLString,
    GraphQLList,
    GraphQLNonNull,
    GraphQLObjectType,
    GraphQLInputObjectType,
    GraphQLBoolean
} = graphql


const addBooking = { // For adding new booking
    type: BookingType,
    args: {
        from: { type: new GraphQLNonNull(GraphQLDate) },
        to: { type: new GraphQLNonNull(GraphQLDate) },
        roomNumbers: { type: new GraphQLNonNull(new GraphQLList(GraphQLInt)) },
        paid: { type: new GraphQLNonNull(GraphQLBoolean) },
        amount: { type: new GraphQLNonNull(GraphQLInt) },
        bookedBy: { type: new GraphQLNonNull(GraphQLID) },
        people: { type: PeopleType },
```

```
        room: { type: new GraphQLNonNull(GraphQLID) },
        hotel: { type: new GraphQLNonNull(GraphQLID) }
    },
    async resolve(parent, args) {
        let hotelData = await Hotel.findById(args.hotel)
        if (!hotelData) throw new Error('Hotel ID is wrong.')
        let roomData = await Room.findById(args.room)
        if (!roomData) throw new Error('Room ID is wrong.')

        let query = await Booking.findOne({ from: args.from, to: args.to, room:
args.room })
        if (query) {
            throw new Error('Cannot book room(s) on same date.')
        }
        else {

            let cntDays = Math.abs(new Date(args.to).getDate() - new
Date(args.from).getDate()) + 1
            const total = args.people.children + args.people.adults

            let booking = new Booking({
                from: args.from,
                to: args.to,
                days: cntDays,
                roomNumbers: args.roomNumbers,
                paid: args.paid,
                amount: args.amount,
                numOfPeople: total,
                location: hotelData.location,
                bookedBy: args.bookedBy,
                people: args.people,
                room: args.room,
                hotel: args.hotel
            })
            let res = await booking.save()
            roomData.bookings.push(res._id)
            await roomData.save()
            return res
        }
    }
}
```

```
const cancelBooking = { // For cancelling hotel
    type: BookingType,
    args: {
        id: { type: new GraphQLNonNull(GraphQLID) }
    },
    async resolve(parent, args) {
        let booking = await Booking.findById(args.id)
        if (!booking) {
            throw new Error('Booking not found.')
        }
        else {
            let room = await Room.findById(booking.room)
            await room.bookings.remove(args.id)
            await room.save()
            let res = await booking.delete()
            return res
        }
    }
}


module.exports = {
    addBooking,
    cancelBooking
}
```

Лістинг коду hotelMutation.js

```
const graphql = require('graphql')
const { UserType, AuthType, HotelType } = require("../Type.js")
const bcrypt = require('bcryptjs')
const User = require("../../models/User.js")
const Hotel = require("../../models/Hotel.js")
const verifyToken = require('../../middlewares/verifyToken.js')
const jwt = require('jsonwebtoken')
const Room = require('../../models/Room.js')
const Booking = require('../../models/Booking.js')

const {
    GraphQLID,
    GraphQLInt,
    GraphQLString,
    GraphQLList,
    GraphQLNonNull,
```

```
    GraphQLObjectType,
} = graphql


const addHotel = { // For adding new hotel
    type: HotelType,
    args: {
        image: { type: GraphQLString },
        name: { type: new GraphQLNonNull(GraphQLString) },
        description: { type: new GraphQLNonNull(GraphQLString) },
        totalRooms: { type: new GraphQLNonNull(GraphQLInt) },
        manager: {type: new GraphQLNonNull(GraphQLID)},
        location: { type: new GraphQLNonNull(GraphQLString) }
    },
    async resolve(parent, args) {
        let query = await Hotel.findOne({ manager: args.manager })
        if (query) {
            throw new Error('Cannot add multiple hotels.')
        }
        else {
            let hotel = new Hotel({
                image: args.image,
                name: args.name,
                description: args.description,
                location: args.location,
                manager: args.manager,
                totalRooms: args.totalRooms,
                roomsMap: {},
                rooms: []
            })
            let res = await hotel.save()
            return res
        }
    }
}

const updateHotel = { // For updating hotel
    type: HotelType,
    args: {
        id: { type: new GraphQLNonNull(GraphQLID) },
        name: { type: new GraphQLNonNull(GraphQLString) },
        description: { type: new GraphQLNonNull(GraphQLString) },
        image: { type: GraphQLString },
```

```
        location: { type: new GraphQLNonNull(GraphQLString) },
        ratings: { type: GraphQLInt },
        totalRooms: { type: new GraphQLNonNull(GraphQLInt) },
    },
    async resolve(parent, args) {
        if (!args.id) throw new Error("ID is not given.");

        let hotel = await Hotel.findByIdAndUpdate(args.id, {
            id: args.id,
            name: args.name,
            description: args.description,
            image: args.image,
            location: args.location,
            ratings: args.ratings,
            totalRooms: args.totalRooms,
        },{ new: true })

        return hotel
    }
}

const deleteHotel = { // For deleting hotel
    type: HotelType,
    args: {
        id: { type: new GraphQLNonNull(GraphQLID) }
    },
    async resolve(parent, args) {
        let hotel = await Hotel.findById(args.id)
        if (!hotel) {
            throw new Error('Hotel not found.')
        }
        else {
            hotel.rooms.forEach(async r => {
                let room = await Room.findById(r._id)
                room.bookings.forEach(async b => {
                    let booking = await Booking.findById(b._id)
                    await booking.delete()
                })
                await room.delete()
            })
            let res = await hotel.delete()
            return res
        }
```

```
    }
}


module.exports = {
    addHotel,
    updateHotel,
    deleteHotel
}
```

Лістинг коду paymentMutation.js

```
const graphql = require('graphql')
const { UserType, AuthType, HotelType, BookingType, PeopleType } =
require("../Type.js")
const User = require("../../models/User.js")
const Booking = require("../../models/Booking.js")
const Stripe = require('stripe');
const dotenv = require('dotenv')

dotenv.config()
const stripe = new Stripe(process.env.STRIPE_SK);

const {
    GraphQLID,
    GraphQLInt,
    GraphQLString,
    GraphQLList,
    GraphQLNonNull,
} = graphql

const payAmount = { // For payment
    type: BookingType,
    args: {
        tokenId: { type: new GraphQLNonNull(GraphQLID) },
        bookingId: { type: new GraphQLNonNull(GraphQLID) },
        bookedBy: { type: new GraphQLNonNull(GraphQLID) },
    },
    async resolve(parent, args) {
        if (!args.tokenId) throw new Error("Token ID is not given.");
        const booking = await Booking.findById(args.bookingId)
        const user = await User.findById(args.bookedBy)

        if (!user || !booking) throw new Error("No booking or user found.");
```

```
        const customer = await stripe.customers.create({
            email: user.email,
            source: args.tokenId
        }).catch(e => {
            throw new Error("Payment failed.");
        })

        const charge = await stripe.charges.create({
            amount: booking.amount * 100,
            currency: 'INR',
            customer: customer.id,
            receipt_email: user.email,
            description: "Transaction",
        }, { idempotencyKey: Math.round(Math.random() * 10000) })
            .catch(e => {
                throw new Error("Payment failed.");
            })

        if (charge) {
            let res = await Booking.findByIdAndUpdate(args.bookingId,
                {
                    from: booking.from,
                    to: booking.to,
                    days: booking.days,
                    roomNumber: booking.roomNumber,
                    amount: booking.amount,
                    numOfPeople: booking.numOfPeople,
                    location: booking.location,
                    bookedBy: booking.bookedBy,
                    people: booking.people,
                    room: booking.room,
                    hotel: booking.hotel,
                    paid: true
                }, { new: true })
            return res
        }
        else throw new Error("Payment failed.")
    }
}

module.exports = {
    payAmount
```

```
}
```

Лістинг коду roomMutation.js

```
const graphql = require('graphql')
const { UserType, AuthType, HotelType, RoomType, ImageType } =
require("../Type.js")
const bcrypt = require('bcryptjs')
const User = require("../../models/User.js")
const Room = require("../../models/Room.js")
const verifyToken = require('../../middlewares/verifyToken.js')
const jwt = require('jsonwebtoken')
const Hotel = require('../../models/Hotel.js')
const Booking = require("../../models/Booking.js")

const {
    GraphQLID,
    GraphQLInt,
    GraphQLString,
    GraphQLList,
    GraphQLNonNull,
    GraphQLObjectType,
    GraphQLInputObjectType,
    GraphQLFloat
} = graphql


const addRoom = { // For adding new room
    type: RoomType,
    args: {
        hotel: { type: new GraphQLNonNull(GraphQLID) },
        images: { type: new GraphQLList(ImageType) },
        name: { type: new GraphQLNonNull(GraphQLString) },
        description: { type: new GraphQLNonNull(GraphQLString) },
        occupancy: { type: new GraphQLNonNull(GraphQLInt) },
        others: { type: new GraphQLList(GraphQLString) },
        price: { type: new GraphQLNonNull(GraphQLInt) },
        roomNumbers: { type: new GraphQLNonNull(new GraphQLList(GraphQLInt)) },
    },
    async resolve(parent, args) {
        let hotelData = await Hotel.findById(args.hotel)
        if (!hotelData) throw new Error('Hotel ID is wrong.')
        let query = await Room.findOne({ name: args.name, hotel: args.hotel })
        if (query) {
```

```
            throw new Error('Cannot add multiple rooms with same name.')
        }
        let ass = []
        args.roomNumbers.forEach(n => {
            if (hotelData.roomsMap.get(n.toString())) ass.push(n)
        })
        if (ass.length > 0) {
            throw new Error('Some room numbers are already assigned. Please try
different room numbers.')
        }
        else {
            let room = new Room({
                hotel: args.hotel,
                images: args.images,
                name: args.name,
                description: args.description,
                occupancy: args.occupancy,
                others: args.others,
                price: args.price,
                roomNumbers: args.roomNumbers,
                bookings: []
            })
            let res = await room.save()
            hotelData.rooms.push(res._id)
            room.roomNumbers.forEach(n => {
                hotelData.roomsMap.set(n.toString(), room.name)
            })

            await hotelData.save()
            return res
        }
    }
}

const updateRoom = { // For updating room
    type: RoomType,
    args: {
        id: { type: new GraphQLNonNull(GraphQLID) },
        images: { type: new GraphQLList(ImageType) },
        name: { type: new GraphQLNonNull(GraphQLString) },
        description: { type: new GraphQLNonNull(GraphQLString) },
        occupancy: { type: new GraphQLNonNull(GraphQLInt) },
        others: { type: new GraphQLList(GraphQLString) },
```

```
        price: { type: new GraphQLNonNull(GraphQLInt) },
        roomNumbers: { type: new GraphQLNonNull(new GraphQLList(GraphQLInt)) },
    },
    async resolve(parent, args) {
        if (!args.id) throw new Error("ID is not given.");

        let room = await Room.findByIdAndUpdate(args.id, {
            id: args.id,
            images: args.images,
            name: args.name,
            description: args.description,
            occupancy: args.occupancy,
            others: args.others,
            price: args.price,
            roomNumbers: args.roomNumbers,
        }, { new: true })

        let hotelData = await Hotel.findById(room.hotel)

        room.roomNumbers.forEach(n => {
            if(!hotelData.roomsMap.get(n.toString())){
                hotelData.roomsMap.set(n.toString(), room.name)
            }
        })

        await hotelData.save()
        return room
    }
}

const deleteRoom = { // For deleting room
    type: RoomType,
    args: {
        id: { type: new GraphQLNonNull(GraphQLID) }
    },
    async resolve(parent, args) {
        let room = await Room.findById(args.id)
        if (!room) {
            throw new Error('Room not found.')
        }
        else {
            let hotel = await Hotel.findById(room.hotel)
            room.roomNumbers.forEach(n => {
```

```
                hotel.roomsMap.delete(n.toString())
            })
            await hotel.rooms.remove(args.id)
            await hotel.save()
            room.bookings.forEach(async b => {
                let booking = await Booking.findById(b._id)
                await booking.delete()
            })
            let res = await room.delete()
            return res
        }
    }
}


module.exports = {
    addRoom,
    updateRoom,
    deleteRoom
}
```

Лістинг коду userMutation.js

```
const graphql = require('graphql')
const { UserType, AuthType } = require("../Type.js")
const bcrypt = require('bcryptjs')
const User = require("../../models/User.js")
const verifyToken = require('../../middlewares/verifyToken.js')
const jwt = require('jsonwebtoken')
const GraphQLDate = require('graphql-date')

const { GraphQLID,
    GraphQLInt,
    GraphQLString,
    GraphQLList,
    GraphQLNonNull,
    GraphQLObjectType
} = graphql


const createUser = { // For creating new user
    type: UserType,
    args: {
        name: { type: new GraphQLNonNull(GraphQLString) },
```

```
        username: { type: new GraphQLNonNull(GraphQLString) },
        dob: { type: new GraphQLNonNull(GraphQLDate) },
        email: { type: new GraphQLNonNull(GraphQLString) },
        password: { type: new GraphQLNonNull(GraphQLString) }
    },
    async resolve(parent, args) {
        let query = await User.findOne({ email: args.email })
        if (query) {
            throw new Error('User already exists.')
        }
        else {
            let passHash = await bcrypt.hash(args.password, 12)

            const accessToken = await jwt.sign({ email: args.email },
'accessToken', {
                expiresIn: '4h'
            })
            const refreshToken = await jwt.sign({ email: args.email },
'refreshToken', {
                expiresIn: '7d'
            })

            let user = new User({
                username: args.username,
                name: args.name,
                email: args.email,
                password: passHash,
                dob: args.dob,
                accessToken: accessToken,
                refreshToken: refreshToken,
                accessTokenExp: '4h',
                refreshTokenExp: '7d'
            })
            let res = await user.save()
            return res
        }
    }
}

const generateToken = { // For generating new access token
    type: UserType,
    args: {
        refreshToken: { type: new GraphQLNonNull(GraphQLString) },
```

```
    },
    async resolve(parent, args) {
        if (!args.refreshToken) {
            throw new Error("Not a refresh token.")
        }
        else {
            let userEmail = await verifyToken(args.refreshToken)
            let user = await User.findOne({ email: userEmail })
            const accessToken = await jwt.sign({ email: user.email },
'accessToken', {
                expiresIn: '4h'
            })
            const refreshToken = await jwt.sign({ email: user.email },
'refreshToken', {
                expiresIn: '7d'
            })
            return {
                username: user.username,
                email: user.email,
                dob: user.dob,
                name: user.name,
                id: user._id,
                accessToken: accessToken,
                refreshToken: refreshToken,
                accessTokenExp: user.accessTokenExp,
                refreshTokenExp: user.refreshTokenExp,
                isAdmin: user.isAdmin,
                isManager: user.isManager,
                isBlocked: user.isBlocked,
                joined: user.joined
            }
        }
    }
}

const updateProfile = { // For updating user profile
    type: UserType,
    args: {
        id: { type: new GraphQLNonNull(GraphQLID) },
        name: { type: new GraphQLNonNull(GraphQLString) },
        email: { type: new GraphQLNonNull(GraphQLString) },
        dob: { type: new GraphQLNonNull(GraphQLDate) }
    },
```

```
    async resolve(parent, args) {
        if (!args.id) throw new Error("ID is not given.");

        let user = await User.findByIdAndUpdate(args.id, {
            id: args.id,
            name: args.name,
            email: args.email,
            dob: args.dob
        },{ new: true })

        return user
    }
}

const deleteAccount = { // For deleting user account
    type: UserType,
    args: {
        id: { type: new GraphQLNonNull(GraphQLID) }
    },
    async resolve(parent, args) {
        if (!args.id) throw new Error("ID is not given.");

        let res = await User.findByIdAndRemove(args.id).exec()
        return res
    }
}

const makeManager = { // For making an user manager
    type: UserType,
    args: {
        id: { type: new GraphQLNonNull(GraphQLID) },
    },
    async resolve(parent, args) {
        if (!args.id) {
            throw new Error("ID is not given.")
        }
        else {
            let user = await User.findByIdAndUpdate(args.id, {
                isManager: true
            }, { new: true })
            return user
        }
    }
```

```
}

const blockUser = { // For block user or manager
    type: UserType,
    args: {
        id: { type: new GraphQLNonNull(GraphQLID) },
    },
    async resolve(parent, args) {
        if (!args.id) {
            throw new Error("ID is not given.")
        }
        else {
            let user = await User.findByIdAndUpdate(args.id, {
                isBlocked: true
            }, { new: true })
            return user
        }
    }
}


module.exports = {
    createUser,
    generateToken,
    updateProfile,
    deleteAccount,
    makeManager,
    blockUser,
}
```

ДОДАТОК Г

Лістинг коду BookingModal.js

```
import React from 'react'
import { FormButton } from '../GlobalStyles/FormStyles'
import { FlexBox, ModalBox, ModalContainer, ModalTitle } from
'../GlobalStyles/ModalStyles'
import 'react-toastify/dist/ReactToastify.css';
import CloseIcon from '@mui/icons-material/Close';
import "./animation.css"
import { getDate, getEasyDate } from '../../utils/utilFunctions'
import { Text } from '../GlobalStyles/PageStyles'
import Tippy from '@tippyjs/react'
import { ButtonsContainer } from '../../pages/Auth/ModuleStyles'
import { useNavigate } from 'react-router-dom'

const BookingModal = (props) => {

    const booking = props.booking
    const navigate = useNavigate()

    const payLink = () => {
        navigate(`/payment/${booking.hotel.id}/${booking.room.id}/1`, {
state: booking })
    }

    return (
        <ModalContainer>
            <ModalBox className="modal-box">
                <CloseIcon className="close-icon"
                    onClick={() => props.setModal({ state: false, param:
null, title: '' })} />
                <ModalTitle>{props.title}</ModalTitle>

                <br />

                <Text className="small">Booked By:
<span>{booking.bookedBy.name}</span></Text>
                <Text className="small">Username:
<span>{booking.bookedBy.username}</span></Text>
```

```
                <Tippy interactive={true}
content={getEasyDate(booking.bookedOn)} placement="bottom">
                    <Text className="small">Booked On:
<span>{getDate(booking.bookedOn)}</span></Text>
                </Tippy>

                <hr />
                <br />

                <FlexBox>

                    <div style={{ flexBasis: '46%' }}>
                        <Text className="small">Hotel:
<span>{booking.hotel.name}</span></Text>
                        <Text className="small">Location:
<span>{booking.location}</span></Text>
                        <Text className="small">Room:
<span>{booking.room.name}</span></Text>
                        <Tippy interactive={true}
content={getEasyDate(booking.from)} placement="bottom">
                            <Text className="small">From:
<span>{getDate(booking.from)}</span></Text>
                        </Tippy>
                        <Tippy interactive={true}
content={getEasyDate(booking.to)} placement="bottom">
                            <Text className="small">To:
<span>{getDate(booking.to)}</span></Text>
                        </Tippy>
                        <Text className="small">Duration:
                            <span> {booking.days} {booking.days === 1 ? 'Day'
: 'Days'}</span>
                        </Text>
                    </div>

                    <div style={{ flexBasis: '46%' }}>
                        <Text className="small">Rooms:
                            {booking.roomNumbers.map(r =>
                                (<span className="highlight" style={{ margin:
'4px 2px' }}>{r}</span>)
                            )}
                        </Text>
                        <Text className="small">Number of persons:
<span>{booking.numOfPeople}</span></Text>
```

```
                        <Text className="small">Adults:
<span>{booking.people.adults}</span></Text>
                        <Text className="small">Children:
<span>{booking.people.children}</span></Text>
                    </div>
                </FlexBox>

                <br />
                <hr />
                <br />
                <Text className="small">Room Price: <span>Rs.
{booking.room.price}</span></Text>
                <Text className="small">Amount {booking.paid ? 'Paid' : 'To
Be Paid'}: <span>Rs. {booking.amount}</span></Text>
                <Text className="small">Payment Status:
                    <span> {booking.paid ? 'Paid' : 'Not Paid'}</span>
                </Text>
                <ButtonsContainer>
                    <FormButton>View Hotel & Room</FormButton>
                    {!booking.paid ? <FormButton onClick={payLink}>Pay
Now</FormButton> : null}
                </ButtonsContainer>
            </ModalBox>
        </ModalContainer>
    )
}

export default BookingModal
```

Лістинг коду HotelModal.js

```
import { useMutation } from '@apollo/client'
import React, { useState } from 'react'
import { FormButton, Input, TextArea } from '../GlobalStyles/FormStyles'
import { ModalBox, ModalContainer, ModalTitle } from
'../GlobalStyles/ModalStyles'
import { toast } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';
import CloseIcon from '@mui/icons-material/Close';
import "./animation.css"
import { GET_HOTEL } from '../../graphql/queries/hotelQueries'
import { ADD_HOTEL, UPDATE_HOTEL } from
'../../graphql/mutations/hotelMutations'
```

```
import ImageUpload from '../ImageUpload/ImageUpload';
import { imageUpload } from '../../utils/utilFunctions';
import Loader from '../Loaders/Loader';
import { MAKE_MANAGER } from '../../graphql/mutations/userMutations';

const HotelModal = (props) => {

    const propsHotel = props.hotel
    const user = JSON.parse(localStorage.getItem('user'))

    const [hide, setHide] = useState(false)
    const [loading, setLoading] = useState(false)

    const [hotel, setHotel] = useState({
        name: propsHotel ? propsHotel.name : '',
        description: propsHotel ? propsHotel.description : '',
        id: props.hotel ? props.hotel.id : null,
        image: propsHotel ? propsHotel.image : '',
        ratings: propsHotel ? propsHotel.ratings : null,
        totalRooms: propsHotel ? propsHotel.totalRooms : null,
        location: propsHotel ? propsHotel.location : ''
    })

    const [makeManager] = useMutation(MAKE_MANAGER, {
        variables: {
            id: user.id
        }
    })

    const [addHotel] = useMutation(ADD_HOTEL)

    const [updateHotel] = useMutation(UPDATE_HOTEL, {
        refetchQueries: [
            GET_HOTEL,
            { variables: { id: props.hotel?.id } }
        ],
    })

    const updateCurHotel = async (e) => {
        e.preventDefault()
        setLoading(true)
        const assignedrooms = Object.keys(propsHotel.roomsMap).length
        if (assignedrooms > hotel.totalRooms) {
```

```
            toast.warning(`You have already assigned ${assignedrooms} rooms.
Unassign those rooms to decrease room numbers.`, {
                autoClose: 5500,
                pauseOnHover: true
            })
            return
        }

        const refPath = `images/hotels/${hotel.id}/hotelImage`
        let imageUrl = null
        if (typeof hotel.image !== 'string') {
            imageUrl = await imageUpload(hotel.image, refPath)
        }

        updateHotel({
            variables: {
                name: hotel.name,
                description: hotel.description,
                id: hotel.id,
                image: imageUrl ? imageUrl : hotel.image,
                ratings: hotel.ratings,
                totalRooms: hotel.totalRooms,
                location: hotel.location
            }
        })
            .then(res => {
                toast.success("Updated hotel.", {
                    autoClose: 5500,
                    pauseOnHover: true,
                    onClose: props.setHotelModal(false)
                })
                setHide(true)
                setLoading(false)
            })
            .catch(err => {
                toast.error(err.message, {
                    autoClose: 5500,
                    pauseOnHover: true
                })
                setLoading(false)
            })
    }
```

```
    const addNewHotel = async (e) => {
        e.preventDefault()
        setLoading(true)

        addHotel({
            variables: {
                manager: user.id,
                image: null,
                name: hotel.name,
                description: hotel.description,
                totalRooms: hotel.totalRooms,
                location: hotel.location
            }
        })
            .then(async res => {
                const refPath =
`images/hotels/${res.data.addHotel.id}/hotelImage`
                let imageUrl = null
                if (typeof hotel.image !== 'string') {
                    imageUrl = await imageUpload(hotel.image, refPath)
                }
                updateHotel({
                    variables: {
                        name: hotel.name,
                        description: hotel.description,
                        id: res.data.addHotel.id,
                        image: imageUrl,
                        ratings: res.data.addHotel.ratings,
                        totalRooms: hotel.totalRooms,
                        location: hotel.location
                    }
                })
                    .then(res => {
                        toast.success("Hotel added.",{
                            autoClose: 5500,
                            pauseOnHover: true
                        })
                        makeManager({
                            variables: {
                                id: user.id
                            }
                        }).then(res => {
```

```
                            localStorage.setItem('user',
JSON.stringify(res.data.makeManager))
                            setHide(true)
                            setLoading(false)
                            props.setHotelModal(false)
                        })
                        .catch(err => {
                            toast.error(err.message, {
                                autoClose: 5500,
                                pauseOnHover: true
                            })
                            setLoading(false)
                        })
                    })
                    .catch(err => {
                        toast.error(err.message, {
                            autoClose: 5500,
                            pauseOnHover: true
                        })
                        setLoading(false)
                    })

            })
            .catch(err => {
                toast.error(err.message, {
                    autoClose: 5500,
                    pauseOnHover: true
                })
                setLoading(true)
            })
    }


    return (
        <ModalContainer>
            <ModalBox className="modal-box">
                {!loading ? (
                    <>
                        <CloseIcon className="close-icon" onClick={() =>
props.setHotelModal(false)} />
                        <ModalTitle>{props.title}</ModalTitle>
                        <ImageUpload imageUrl={hotel.image}
refPath={`images/hotels/${hotel.id}/hotelImage`}
```

```
                            setImageURL={(val) => setHotel({ ...hotel, image:
val })} single={true} />

                        <form onSubmit={props.action === 'update' ?
updateCurHotel : addNewHotel}>
                            <Input required="true" style={{ marginBottom:
'16px' }}
                                value={hotel.name} onChange={(e) =>
setHotel({ ...hotel, name: e.target.value })}
                                placeholder="Hotel name">
                            </Input>

                            <TextArea required="true" style={{ marginBottom:
'16px' }}
                                value={hotel.description} onChange={(e) =>
setHotel({ ...hotel, description: e.target.value })}
                                placeholder="Hotel description"></TextArea>

                            <Input required="true" style={{ marginBottom:
'16px' }}
                                value={hotel.location} onChange={(e) =>
setHotel({ ...hotel, location: e.target.value })}
                                placeholder="Location"></Input>

                            <Input required="true" style={{ marginBottom:
'16px' }}
                                value={hotel.totalRooms} onChange={(e) =>
setHotel({ ...hotel, totalRooms: Number(e.target.value) })}
                                placeholder="Total Rooms"></Input>

                            {!hide && (
                                <FormButton type="submit"
                                    style={{ marginLeft: 'auto', marginTop:
'40px' }}>
                                    {props.title}
                                </FormButton>
                            )}
                        </form>
                    </>
                ) : <Loader />}
            </ModalBox>
        </ModalContainer>
    )
```

```
}

export default HotelModal
```