МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Відокремлений структурний підрозділ
«Тернопільський фаховий коледж
Тернопільського національного технічного університету імені Івана Пулюя»

Відділення телекомунікацій та електронних систем

Циклова комісія комп'ютерних наук

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи

фахового молодшого бакалавра

на тему: Розробка інформаційної системи управління готельним фондом та бронюванням «Montreux Regent Hotel»

Виконав: студент IV курсу, групи КН-423
спеціальності: 122 Комп'ютерні науки
Богдан ПРОЦИК

Керівник Руслан СЛОБОДЯН

Рецензент
(ім'я та прізвище)

м. Тернопіль – 2026

ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ТЕРНОПІЛЬСЬКИЙ ФАХОВИЙ КОЛЕДЖ
ТЕРНОПІЛЬСЬКОГО НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ
ІМЕНІ ІВАНА ПУЛЮЯ»

Відділення телекомунікацій та електронних систем
Циклова комісія комп'ютерних наук
Освітньо-професійний ступінь «фаховий молодший бакалавр»
Спеціальність 122 Комп'ютерні науки
Галузь знань 12 Інформаційні технології

ЗАТВЕРДЖУЮ
Голова циклової комісії
комп'ютерних наук
_____ Галина МАРЦІЯШ
« 02 » березня 2026 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Процику Богдану Олеговичу
(прізвище, ім'я, по батькові)

1. Тема роботи: Розробка інформаційної системи управління готельним фондом та бронюванням «Montreux Regent Hotel»,

керівник роботи Слободян Руслан Олесійович,
(прізвище, ім'я, по батькові)
затверджені наказом вищого навчального закладу № 4/9-132 від 27.02.2026 р.
2. Строк подання студентом роботи: 19.06.2026 р.
3. Вихідні дані до роботи: технічне завдання на розробку програмного забезпечення, мова програмування TypeScript; React, NodeJS, СКБД Mongo DB, стандарти: ДСТУ ISO/IEC/IEEE 29148:2025, ДСТУ ISO/IEC/IEEE 29119-1:2017, ДСТУ 3008:2015, ДСТУ 8302:2015.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):

1 Загальний розділ

1.1 Аналітичний огляд існуючих рішень

1.2 Технічне завдання

1.2.1 Найменування та область застосування

1.2.2 Призначення розробки

1.2.3 Вимоги до функціоналу інформаційної системи

1.2.4 Вимоги до програмної документації

1.2.5 Техніко-економічні показники

1.2.6 Стадії та етапи розробки

1.2.7 Порядок тестування та прийому

2 Розробка технічного та робочого проєкту

2.1 Розробка структури інформаційної системи

2.2 Створення та верстка сторінок інформаційної системи

2.3 Розробка структури бази даних інформаційної системи

2.4 Програмування інформаційної системи

2.4.1 Написання клієнтської частини

2.4.2 Написання серверної частини

2.5 Тестування інформаційної системи

3 Спеціальний розділ

3.1 Інструкція з розміщення інформаційної системи в Інтернеті

3.2 Інструкція з обслуговування та наповнення інформаційної системи

3.3 Інструкція з популяризації та підтримки інформаційної системи

4 Економічний розділ

4.1 Визначення стадій технологічного процесу та загальної тривалості проведення розробки інформаційної системи

4.2 Визначення витрат на оплату праці та відрахувань на соціальні заходи

4.3 Розрахунок витрат на електроенергію

4.4 Розрахунок суми амортизаційних відрахувань

4.5 Обчислення накладних витрат

4.6 Складання кошторису витрат та визначення собівартості інформаційної системи

4.7 Розрахунок ціни інформаційної системи

4.8 Визначення економічної ефективності і терміну окупності капітальних вкладень

5 Охорона праці, техніка безпеки та екологічні вимоги

5.1 Гарантія прав на охорону праці.

5.2 Шкідливі речовини, що виділяються при паянні. Вимоги до вентиляції робочого місця

6 Висновки

Додаткові вказівки: виконання кваліфікаційної роботи із розробкою програмного продукту – інформаційної системи «Montreux Regent Hotel».

5. Перелік графічного матеріалу:

1. Схема структурна клієнтської частини
2. UML-діаграма варіантів використання
3. ER-діаграма бази даних
4. Таблиця техніко-економічних показників

6. Консультанти розділів роботи

Розділ	**Ім'я, прізвище та посада консультанта**	**Підпис, дата**	
		завдання видав	**завдання прийняв**
Економічний розділ	Любов КАЛУШКА		
Охорона праці, техніка безпеки та екологічні вимоги	Генадій ГОРЯЧЕК		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Отримання і аналіз технічного завдання	20.03.2026	
2	Збір і узагальнення інформації	01.05.2026	
3	Написання першого розділу	15.05.2026	
4	Розробка технічного та робочого проєкту	29.05.2026	
5	Написання спеціального розділу	05.06.2026	
6	Розрахунок економічної частини	08.06.2026	
7	Написання розділу охорони праці	09.06.2026	
8	Виконання графічної частини	10.06.2026	
9	Оформлення пояснювальної записки	11.06.2026	
10	Погодження нормоконтролю	12.06.2026	
11	Попередній захист кваліфікаційної роботи	.06.2026	
12	Захист кваліфікаційної роботи	.06.2026	

7. Дата видачі завдання: 06 березня 2026 р.

Студент _____ Богдан ПРОЦИК
(підпис)

Керівник роботи _____ Руслан СЛОБОДЯН
(підпис)

ЗМІСТ

АНОТАЦІЯ

Тема кваліфікаційної роботи: Розробка інформаційної системи управління готельним фондом та бронюванням «Montreux Regent Hotel».

Метою кваліфікаційної роботи є розробка інформаційної системи для управління готельним фондом.

Пояснювальна записка складається з 5 розділів.

У загальній частині описуються аналітичний огляд існуючих рішень та аналіз технічного завдання програмного продукту.

У другому розділі представлено розробку технічного та робочого проєкту інформаційної системи, зокрема розробку структури системи, створення та верстку її сторінок, розробку структури бази даних, програмування клієнтської та серверної частин, а також тестування розробленого програмного продукту.

В спеціальній частині описано процес розміщення інформаційної системи в Інтернеті, інструкцію з обслуговування та наповнення системи, а також інструкцію з популяризації та підтримки вебзастосунку «Montreux Regent Hotel».

Розрахунок вартості розробки та економічної ефективності приведено в економічній частині в четвертому розділі.

Основні питання охорони праці та техніки безпеки розглянуто в п’ятому розділі.

Обсяг пояснювальної записки 108 сторінок.

До складу кваліфікаційної роботи входить графічна частина, яка складається з схеми структурної клієнтської частини інформаційної системи, UML-діаграми варіантів використання, ER-діаграми бази даних та таблиці техніко-економічних показників, що виконані на окремих аркушах формату А1.

ABSTRACT

Topic of the qualification thesis: Development of the "Montreux Regent Hotel" information system for hotel property and reservation management.

The purpose of the qualification thesis is to develop software for a hotel property management information system.

The explanatory note consists of 5 chapters. The general section provides an analytical review of existing solutions and an analysis of the software product's technical specifications.

The second chapter presents the development of the technical and detailed design of the information system, including the system structure design, page creation and layout, database structure design, client-side and server-side programming, as well as testing of the developed software product.

The dedicated section describes the deployment process of the information system to the Internet, maintenance and content management instructions, as well as instructions for the promotion and support of the "Montreux Regent Hotel" web application.

The calculation of development costs and economic efficiency is presented in the economic section in the fourth chapter.

Key issues of occupational health and safety are addressed in the fifth chapter.

The volume of the explanatory note is 108 pages.

The qualification work includes a graphic part consisting of a structural diagram of the client side of the information system, a UML use case diagram, an ER diagram of the database, and a table of technical and economic indicators, presented on separate A1 format sheets.

ВСТУП

Сьогодні важко уявити готельний бізнес без власного веб-сайту. Люди звикли шукати та бронювати готелі онлайн, порівнювати ціни, читати відгуки та обирати номери прямо зі смартфона чи комп'ютера. Тому наявність зручного та сучасного веб-сайту для готелю є не просто бажаною, а необхідною умовою успішної роботи.

Актуальність теми кваліфікаційної роботи «Розробка інформаційної системи управління готельним фондом та бронюванням «Montreux Regent Hotel»» пов'язана з тим, що попит на онлайн-бронювання готелів постійно зростає. Користувачі очікують від готельних сайтів зручного інтерфейсу, швидкої роботи та простого процесу бронювання. Використання сучасних веб-технологій дозволяє створити потрібний ресурс.

Метою кваліфікаційної роботи є розробка інформаційної системи управління готельним фондом та бронюванням «Montreux Regent Hotel», який дозволяє переглядати інформацію про готель та його послуги, обирати та бронювати номери онлайн, користуватися особистим кабінетом, а також керувати вмістом сайту через адміністративну панель. Для досягнення успішного виконання були поставлені такі завдання:

1. проаналізувати існуючі готельні веб-сайти;
2. спроєктувати структуру веб-сайту та базу даних;
3. розробити інтерфейс користувача;
4. реалізувати функціонал для користувачів та адміністратора;
5. провести тестування системи та оформити результати розробки.

Під час дослідження були використані відомі онлайн систем бронювання готельних номерів для аналізу та збору інформаціїї. Кваліфікаційна робота виконана відповідно до методичних вказівок щодо виконання та оформлення кваліфікаційних робіт здобувачів фахової передвищої освіти спеціальності 122 Комп'ютерні науки [1].

1 ЗАГАЛЬНИЙ РОЗДІЛ

1.1 Аналітичний огляд існуючих рішень

Перед початком розробки інформаційної системи було проведено аналіз існуючих вебсайтів готелів з метою визначення їх переваг і недоліків, а також формування вимог до власного рішення. Об'єктами аналізу стали вебсайти готелів Hilton Garden Inn та JAG Hotel.

Hilton Garden Inn — представник відомої міжнародної мережі готелів, що має розвинену онлайн-інфраструктуру для бронювання. Вигляд головної сторінки сайту зображено на рисунку 1.1.

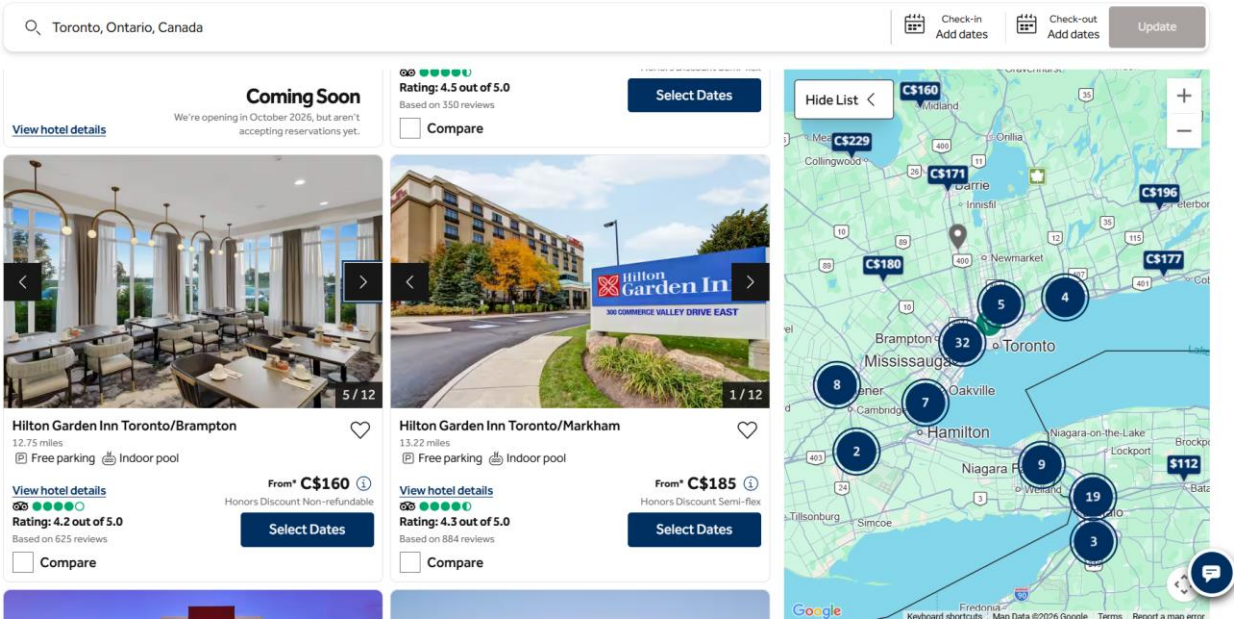


Рисунок 1.1 – Головна сторінка вебсайту Hilton Garden Inn

Аналіз вебсайту Hilton Garden Inn дозволив виявити такі переваги:

– інтуїтивно зрозуміла система онлайн-бронювання номерів;
– широкий вибір категорій номерів та варіантів розміщення;
– якісний фотоконтент із детальними описами кожного номера;
– наявність опції розміщення з домашніми тваринами.

Серед недоліків було виявлено:

– незрозумілу навігацію між розділами сайту;

– відсутність відображення вартості проживання безпосередньо на сторінці номера;

– відсутність можливості налаштування інтерфейсу користувачем.

JAG Hotel – готель із власним стилем і концепцією, що представлений відповідним вебсайтом. Вигляд головної сторінки сайту зображено на рисунку 1.2.

Рисунок 1.2 – Головна сторінка вебсайту JAG Hotel

Аналіз вебсайту JAG Hotel дозволив виявити такі переваги:

– якісне візуальне оформлення головної сторінки;

– помітна та виразна кнопка «Book Now», що стимулює до дії;

– лаконічне та зрозуміле головне меню;

– наявність інформації про власний концертний зал;

– активна присутність у соціальних мережах.

Серед недоліків було виявлено:

– логотип погано читається на тлі фонового зображення;

– відсутність відображення цін за номер на сторінці готелю;

– процес бронювання перенаправляє користувача на сторонній ресурс;

– наявність великої кількості неоптимізованих відеоматеріалів, що уповільнює завантаження сторінок;

– надмірна кількість розділів, що ускладнює навігацію.

За результатами аналізу було зрозуміло, що основними факторами для готельного фонду є легкий інтерфейс, зручна функція пошуку та фільтрації номерів, якісні фотографії та детальні описи, а також адаптивний дизайн для коректного відображення на мобільних пристроях.

1.2 Технічне завдання

1.2.1 Найменування та область застосування

Найменування програмного забезпечення — «Інформаційна система управління готельним фондом та бронюванням «Montreux Regent Hotel»».

Система призначена для автоматизації основних процесів роботи готелю: онлайн-бронювання номерів, управління номерним фондом, обліку клієнтів та обробки платежів. Забезпечує зручний і адаптивний доступ користувачів до інформації про готель, його послуги та доступні номери.

1.2.2 Призначення розробки

Метою розробки є створення сучасної вебсистеми, яка надає потенційним гостям змогу швидко отримати необхідну інформацію про готель і без зайвих зусиль оформити бронювання. Система також формує перше цифрове враження про заклад, тому її якість безпосередньо впливає на репутацію готелю в мережі.

Розроблена система повинна вирішувати такі завдання:

– надання гостям повної інформації про номерний фонд, послуги та ціни;

– забезпечення зручного та безпечного процесу онлайн-бронювання;

– автоматизація адміністративної роботи персоналу готелю через панель адміністратора;

– зберігання та управління даними про користувачів, бронювання та номери.

1.2.3 Вимоги до функціоналу вебсайту інформаційної системи

Функціональні вимоги.

Базові вимоги до контенту та структури інформаційної системи управління готельним фондом та бронюванням «Montreux Regent Hotel»:

– Головна сторінка повинна містити коротку презентацію готелю, переваги закладу та якісні фотографії інтер'єрів.

– Каталог номерів із можливістю фільтрації за датами заїзду/виїзду, типом номера та ціновим діапазоном.

– Сторінка кожного номера з детальним описом, переліком зручностей, фотогалереєю та актуальною вартістю.

– Окрема форма бронювання з вибором дат, типу номера та кількості гостей.

– Сторінка контактів із поштовою адресою, картою розташування та формою зворотного зв'язку.

– Особистий кабінет користувача з переглядом історії та статусів бронювань.

– Панель адміністратора для самостійного управління контентом без залучення розробників.

Вимоги до функцій користувача:

– перегляд каталогу номерів та детальних сторінок номерів;

– пошук і фільтрація номерів за датами, категорією та ціною;

– реєстрація та авторизація в системі;

– оформлення бронювання та онлайн-оплата;

– перегляд, управління та скасування власних бронювань;

– отримання email-підтвердження після успішного бронювання.

Вимоги до функцій адміністратора:

– додавання, редагування та видалення номерів і їх фотографій;

– перегляд усіх бронювань та управління їх статусами;

– перегляд списку зареєстрованих користувачів;

– управління доступністю номерів.

Нефункціональні вимоги.

Інформаційна система управління готельним фондом та бронюванням «Montreux Regent Hotel» повинна мати:

– адаптивний дизайн для коректного відображення на пристроях різних розмірів (комп'ютер, планшет, смартфон);

– час завантаження сторінок — не більше 3 секунд за стандартних умов з'єднання;

– стабільна робота системи при одночасному використанні кількома користувачами;

– захист персональних даних користувачів відповідно до сучасних стандартів безпеки;

– зрозуміла навігація: доступ до будь-якого розділу не більше ніж за 2–3 кліки.

Вебсайт повинен включати: головну сторінку, каталог номерів, сторінку окремого номера з детальним описом, форму бронювання, особистий кабінет користувача та інформаційні сторінки про готель і його послуги.

1.2.4 Вимоги до програмної документації

Документація до інформаційної системи повинна містити:

– опис призначення системи та її функціональних можливостей;

– опис архітектури системи та взаємодії її компонентів;

– перелік використаних технологій і сторонніх сервісів;

– інструкцію з розміщення інформаційної системи в Інтернеті;

– інструкцію з обслуговування та наповнення;
– інструкцію з популяризації та підтримки.

1.2.5 Техніко-економічні показники

До основних техніко-економічних показників належать:
– витрати часу на проєктування, розробку та тестування системи;
– вартість використаних програмних засобів і технічних ресурсів;
– витрати на підключення та використання сторонніх сервісів;
– витрати на подальшу підтримку та розвиток системи.

Використання сучасних безкоштовних і відкритих інструментів веб-розробки (React.js, Node.js, MongoDB) дозволило суттєво мінімізувати фінансові витрати на реалізацію проєкту. Залучені сторонні сервіси (Clerk, Stripe, Cloudinary, Brevo) мають безкоштовні тарифні плани, достатні для початкового етапу експлуатації системи. Готова інформаційна система відповідає сучасним вимогам до швидкодії, зручності використання та адаптивності інтерфейсу.

1.2.6 Стадії та етапи розробки

Процес розробки інформаційної системи «Montreux Regent Hotel» передбачає виконання таких етапів:

– Аналіз і проєктування — аналіз вимог до системи, огляд існуючих рішень, розроблення технічного завдання та проєктування архітектури системи і дизайну інтерфейсу.

– Розробка серверної частини — реалізація REST API на Node.js та Express.js, налаштування та підключення бази даних MongoDB.

– Інтеграція сторонніх сервісів — налаштування Cloudinary для зберігання зображень, підключення системи авторизації Clerk, інтеграція платіжної системи Stripe, підключення сервісу Brevo для надсилання email-підтверджень.

– Розробка клієнтської частини — реалізація інтерфейсу на React.js, розробка всіх сторінок та компонентів, підключення до серверного API.

– Тестування — функціональне, структурне та модульне тестування клієнтської і серверної частин, виявлення та усунення помилок.

– Розгортання — деплой системи на платформу Vercel, перевірка коректної роботи в продакшн-середовищі..

1.2.7 Порядок тестування та прийому

Тестування інформаційної системи управління готельним фондом та бронюванням «Montreux Regent Hotel» є обов'язковим етапом розробки та проводиться з метою підтвердження відповідності системи встановленим функціональним і нефункціональним вимогам перед її введенням в експлуатацію.

Під час перевірки якості розробленої системи застосовуються такі види тестування:

– функціональне тестування — перевірка відповідності реалізованого функціоналу задекларованим вимогам;

– модульне тестування — перевірка окремих компонентів клієнтської та серверної частин у ізольованому середовищі;

– структурне тестування — перевірка коректності взаємодії між модулями системи;

– тестування інтерфейсу — перевірка коректності відображення системи на пристроях різних типів і розмірів екранів.

Тестування проводиться у середовищі, максимально наближеному до реальних умов експлуатації. Для перевірки платіжної системи використовуються тестові карткові реквізити, що надаються платформою Stripe. Тестування авторизації здійснюється із залученням тестових облікових записів у сервісі Clerk.

Перелік об'єктів тестування:

– Модуль реєстрації та авторизації — перевірка коректності процесу створення облікового запису, входу та виходу з системи; реакція системи на

некоректно введені дані (порожні поля, невалідний формат email, недостатня довжина пароля).

– Модуль роботи з базою даних — перевірка відповідності збережених даних введеним користувачем; коректність структури документів у колекціях MongoDB; відсутність дублювання записів; цілісність зв'язку між обліковими записами Clerk та записами у MongoDB.

– Модуль каталогу та перегляду номерів — коректність відображення інформації про номери, фотографій, цін та доступності на обрані дати.

– Модуль бронювання — коректність роботи форми бронювання, правильність розрахунку вартості залежно від обраного періоду проживання та типу номера.

– Модуль оплати — коректність проведення тестових транзакцій через Stripe; обробка успішних платежів і платежів із помилками (відмова банку, некоректні реквізити); відповідність поведінки інтерфейсу очікуваному результату.

– Панель адміністратора — перевірка функцій додавання, редагування та видалення номерів; управління бронюваннями та користувачами системи.

– Адаптивність інтерфейсу — коректність відображення системи на мобільних телефонах, планшетах і настільних комп'ютерах з різними роздільними здатностями екрана.

Система вважається готовою до прийому за умови виконання таких критеріїв:

– усі задекларовані функціональні вимоги реалізовано та перевірено;

– відсутні критичні та блокуючі помилки у роботі системи;

– інтерфейс коректно відображається на всіх типах пристроїв;

– час завантаження сторінок не перевищує встановлених нормативів;

– усі виявлені в ході тестування помилки усунено до здачі роботи.

2 РОЗРОБКА ТЕХНІЧНОГО ТА РОБОЧОГО ПРОЄКТУ

2.1 Розробка структури інформаційної системи

Метою розробки є розробка інформаційної системи управління готельним фондом та бронюванням «Montreux Regent Hotel», яка виконує автоматизацію основних процесів готелю: бронювання номерів, управління номерним фондом, обліку клієнтів та онлайн-оплати послуг.

Реалізація даної системи покликана значно спростити повсякденну діяльність персоналу готелю та підвищити загальний рівень сервісу, що надається клієнтам на всіх етапах взаємодії із закладом.

Сучасний готельний бізнес вимагає оперативного управління інформацією про стан готельного фонду, клієнтів та бронювань. В умовах постійно зростаючої конкуренції на ринку готельних послуг швидкість та якість обслуговування клієнтів стають одними з ключових факторів, що визначають успішність закладу.

Ручне ведення обліку є неефективним, схильним до помилок і не допускає належної швидкості обслуговування, оскільки вимагає значних затрат часу персоналу на виконання рутинних операцій та підвищує ризик виникнення людського фактору при опрацюванні великих обсягів інформації.

Розроблена програма прибирає ці недоліки шляхом централізованого зберігання та обробки даних у режимі реального часу, що дозволяє персоналу готелю миттєво отримувати актуальну інформацію про стан номерного фонду незалежно від місця знаходження та часу доби.

Функція реалізована за послідовністю, що забезпечує чітке розмежування між клієнтською та серверною логікою застосунку та дозволяє незалежно розвивати кожну з цих частин у подальшому. Серверна частина розроблена на мові програмування JavaScript [15] з використанням середовища Node.js [6] та фреймворку Express.js [3] з використанням бібліотеки React [4].

Обраний технологічний стек є одним із найпоширеніших у сучасній веб-розробці, що забезпечує наявність широкої спільноти розробників, великої

кількості документації та готових рішень для типових задач. Дані зберігаються у базі даних MongoDB [5], яка завдяки своїй документо-орієнтованій структурі добре підходить для зберігання даних зі змінною та гнучкою структурою, що характерно для інформації про готельні номери з різним набором характеристик та зручностей.

Розробка здійснювалась у середовищі Visual Studio Code, яке є одним із найпопулярніших інструментів серед розробників завдяки зручному інтерфейсу та широкій підтримці розширень для роботи з JavaScript та супутніми технологіями.

Система передбачає два типи користувачів: клієнт та адміністратор. Розподіл ролей виконується механізмом авторизації на стороні сервера, що забезпечує надійний захист адміністративного функціоналу від несанкціонованого доступу з боку звичайних клієнтів системи.

Перелік функціоналу:

– реєстрацію нових клієнтів та авторизацію адміністратора. Доступ до адміністративних функцій надається виключно авторизованим користувачам з відповідною роллю;

– адміністратор має можливість додавати нові номери, редагувати їх характеристики та видаляти номери з системи. Кожен номер свій варіант доступних послуг що відображаються у вигляді іконок;

– клієнт має можливість переглядати доступні номери та здійснювати бронювання на обраний період;

– адміністратор має доступ до переліку всіх активних бронювань, може переглядати деталі кожного бронювання, зокрема дати заїзду та виїзду, та керувати їх статусом.

– включає функціонал онлайн-оплати бронювань, який є повністю реалізованим та функціональним.

2.2 Створення та верстка сторінок інформаційної системи

Головна сторінка сайту це перше, що бачить користувач при заході на сайт. Зверху меню з вкладками Home, Hotels, Experience, About і кнопка Dashboard для адміністратора, а поруч іконка пошуку та аватар, через який реалізовано вхід в систему за допомогою Clerk [11]. По центру екрану фото готельного хома з заголовком і коротким описом готелю, а нижче форма швидкого пошуку можна вказати напрямок, дати заїзду й виїзду та кількість гостей і одразу перейти до списку номерів. Верстка сторінок виконана з використанням сучасних стандартів HTML, CSS та JavaScript [15] (див. рис. 2.1).

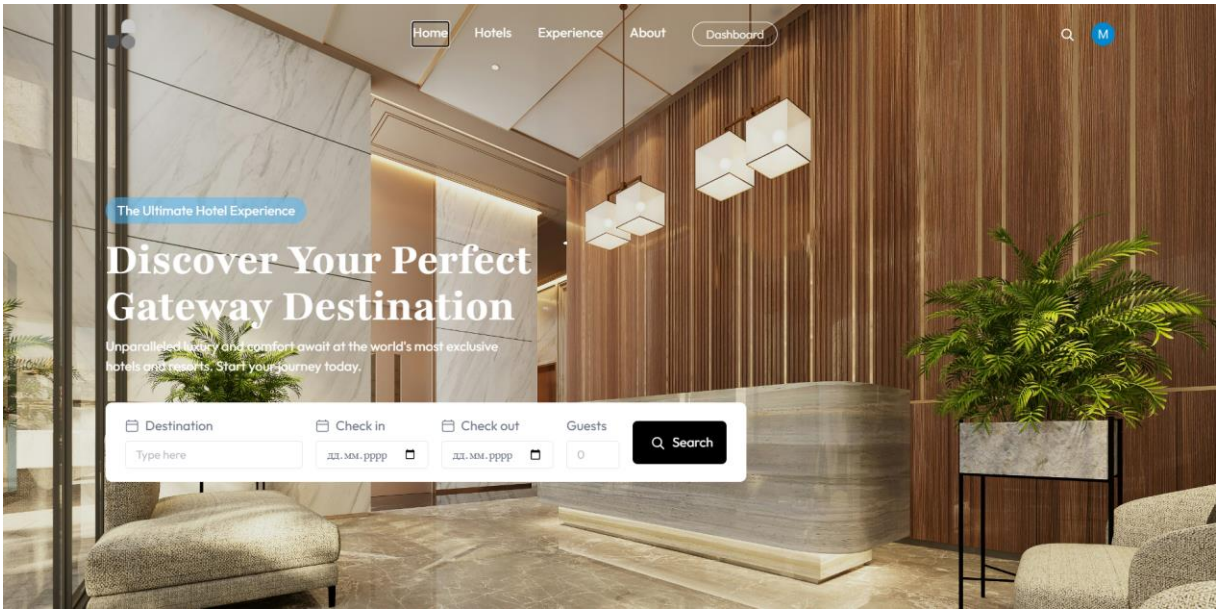


Рисунок 2.1 – Головна сторінка сайту

Після пошуку користувач потрапляє на сторінку Hotels де він може побачити повну інформацію бро доступні готельні номери:

– ціну;
– тип номеру
– інформацію про включені послуги;
– рейтинг готелю.

Також на цій стадії реалізовано систему фільтрації готельних номерів за

такими ознаками:

– тип номеру;
– ціна;
– сортування від найдешевшого до найдорожчого і наоборот;
– за датою додавання.

Вкладка з готельними номерами показано на рисунку 2.2.

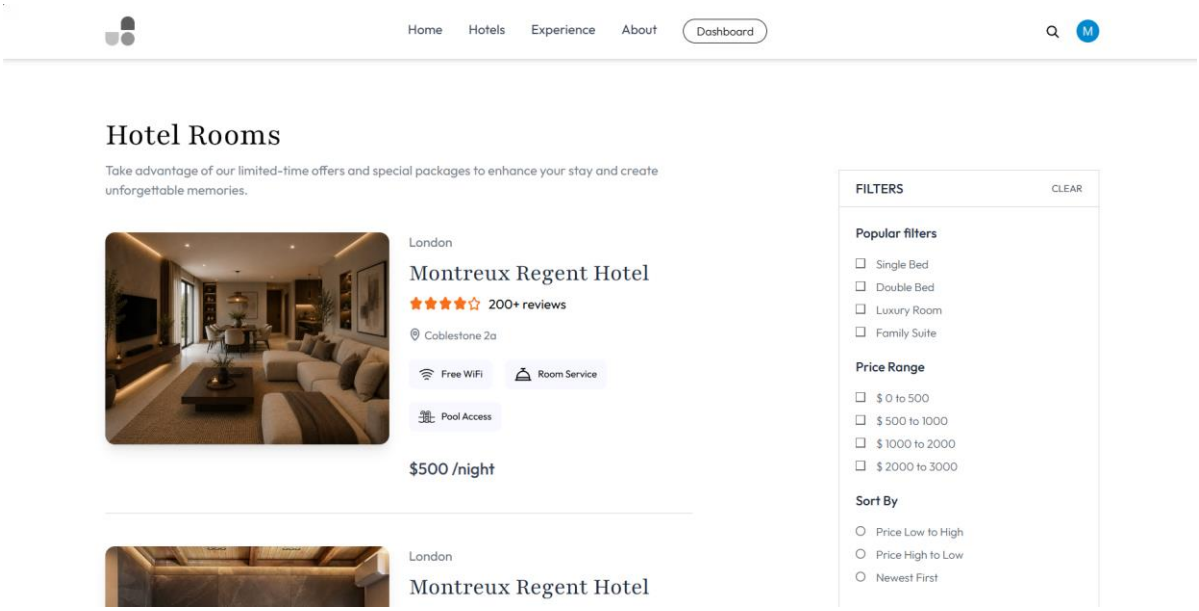


Рисунок 2.2 – Вкладка з готельними номерами

Якщо натиснути на конкретний номер, відкривається його детальна сторінка з фотографіями, заголовком, описом і позначкою, чи номер вільний на потрібні дати. Така структура сторінки дозволяє клієнту отримати максимально повне уявлення про номер ще до моменту прийняття рішення про бронювання, не звертаючись додатково до персоналу готелю за уточненнями.

На цій же сторінці видно ціну за добу і доступні зручності готельного номеру, що відображаються у вигляді переліку з відповідними піктограмами для зручності сприйняття інформації. Завдяки такому наочному поданню клієнт може швидко порівняти кілька варіантів номерів між собою та обрати найбільш відповідний власним потребам і бюджету, не витрачаючи додатковий час на пошук цієї інформації в інших розділах сайту.

Та вибором кількості гостей і кнопкою Book Now, яка перенаправляє користувача на сторінку з системою бронювання та оплати готельного номеру. Натискання даної кнопки ініціює подальший процес оформлення бронювання, у межах якого користувач переходить до завершального етапу взаємодії із системою підтвердження обраних параметрів проживання та проведення оплати через платіжну систему Stripe.

Під формою ще короткий опис переваг проживання, що додатково підкреслює основні сильні сторони обраного номера та сприяє прийняттю клієнтом остаточного рішення про бронювання (див. рис. 2.3).

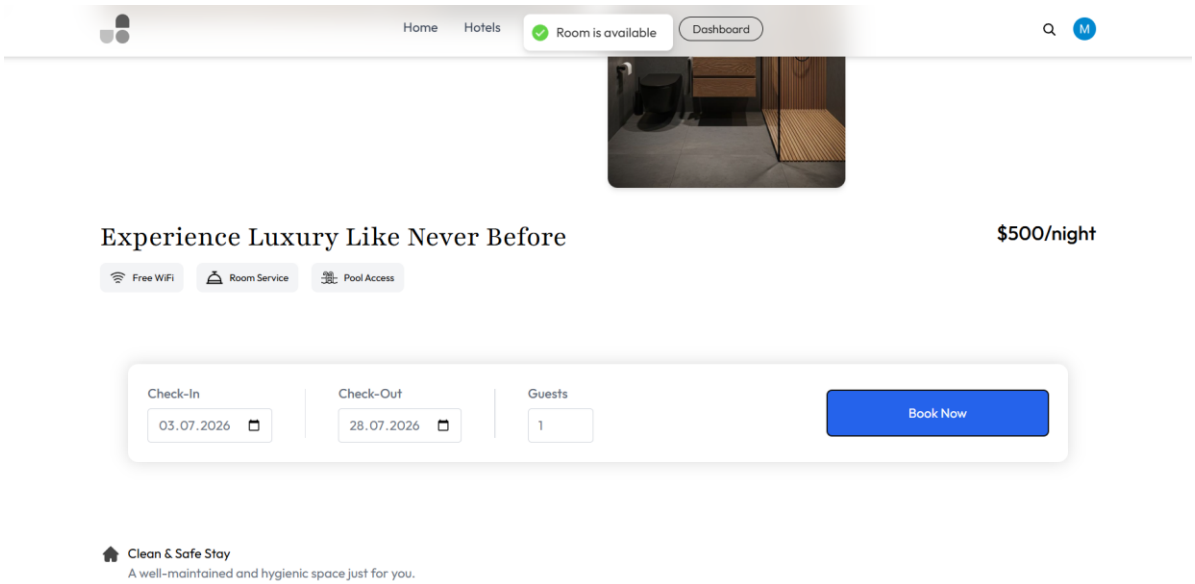


Рисунок 2.3 – Система бронювання

Через аватар у верхньому правому куті відкривається випадаюче меню облікового запису, реалізоване сервісом Clerk, де відображається назва акаунту готелю, прив'язана електронна адреса, а також пункти Manage account та Sign out.

Така реалізація дозволяє користувачу швидко отримати доступ до основних дій з обліковим записом, не заплутуючись у зайвих елементах інтерфейсу.

При натисканні на пункт Manage account відкривається повноцінне вікно Account, у якому користувачу доступно два розділи:

– profile;

– security.

Використання готового рішення від Clerk для реалізації даного функціоналу дозволило не витрачати додатковий час на самостійну розробку інтерфейсу керування обліковим записом, а також забезпечило високий рівень безпеки зберігання та обробки персональних даних користувачів відповідно до сучасних стандартів. У розділі profile користувач має змогу переглянути та за потреби відредагувати основну інформацію про свій обліковий запис, тоді як розділ security відповідає за параметри, пов'язані безпосередньо з безпекою доступу до системи.

Де можна побачити свою пошту, додати номер телефону або перевірити підключені акаунти, наприклад Google. Наявність можливості підключення стороннього акаунту значно спрощує процес автентифікації для користувача, оскільки звільняє його від необхідності запам'ятовувати окремий пароль для входу до системи готелю та дозволяє здійснювати вхід в один клік за допомогою вже існуючого облікового запису Google (див. рис. 2.4).

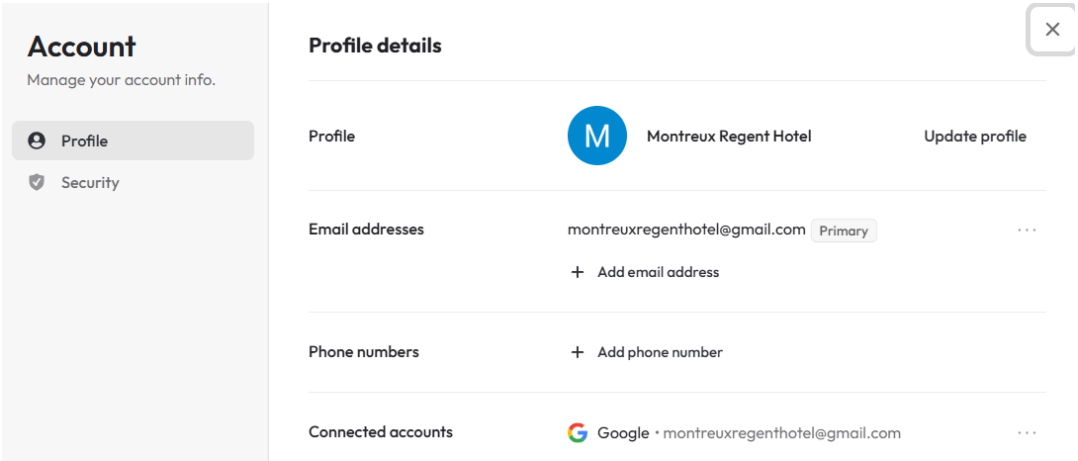


Рисунок 2.4 – Меню облікового запису користувача

Окремо для адміністратора є панель Dashboard, яку звичайний користувач не бачить. Доступ до даного розділу регулюється механізмом авторизації на стороні сервера, завдяки чому навіть у разі спроби прямого переходу за відповідним посиланням користувач без адміністративної ролі не отримає

доступу до функціоналу управління готелем. Така реалізація забезпечує надійне розмежування можливостей між клієнтами та персоналом готелю в межах однієї системи.

В цій вкладці доступне меню з трьома пунктами: Dashboard, Add Room, List Room. Подібна структура навігації дозволяє адміністратору швидко переміщатись між основними розділами керування без необхідності постійного повернення до головної сторінки системи.

На самій панелі видно загальну кількість бронювань і суму доходу, а нижче таблиця останніх бронювань з іменем користувача, типом номера, сумою та статусом оплати.

Завдяки наявності зведеної інформації у верхній частині сторінки адміністратор має змогу одним поглядом оцінити поточний стан справ готелю, не заглиблюючись у деталі кожного окремого бронювання, а таблиця останніх бронювань дозволяє при необхідності оперативно відстежити, які платежі ще очікують підтвердження, а які вже були успішно завершені (див. рис. 2.5).

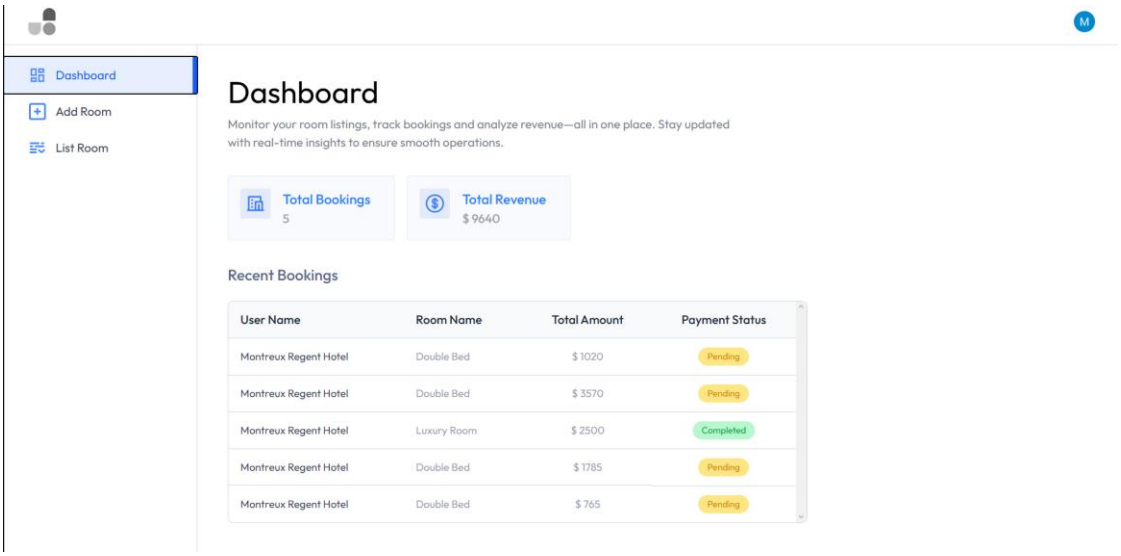


Рисунок 2.5 – Панель адміністратора

З цьої панелі адміністратор також може додати новий номер через форму Add Room, завантажити до чотирьох фото, вибрати тип номеру серед доступних:

– single;

– double;

– luxury;

– family.

Після вибору типу номеру потрібно вказати ціну за добу і позначити доступні зручності серед яких:

– free WiFi

– free breakfast

– room service

– mountain view

– pool access

Після заповнення форм номер додається кнопкою Add Room (див. рис. 2.6).

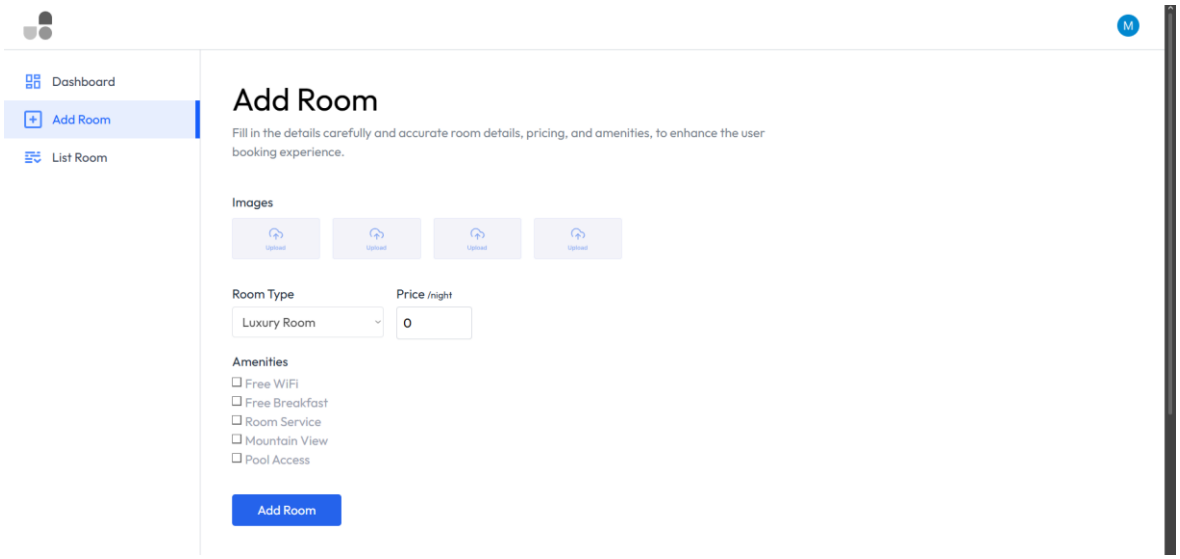


Рисунок 2.6 – Додавання нового номеру

2.3 Розробка структури бази даних інформаційної системи

Система передбачає два типи акторів: User та Admin. Кожен актор має доступ до певного набору функцій системи відповідно до своєї ролі.

Користувач має доступ до таких функцій: перегляд інформації про готель, перегляд інформації про доступні номери, детальної інформації про обраний номер, оформити бронювання обраного номера із зазначенням дат прибуття та

відбуття, після чого додаток автоматично розраховує загальну вартість проживання та надає можливість здійснити оплату.

Адміністратор має доступ до таких функцій: перегляд інформації усіх номерів, доступ до іформація про бронювання, додавання нового номера в систему готелю.

Функціональність описує кожну можливу взаємодію користувача із системою. На базовому рівні використання для кожної функції визначаються вхідні дані, вихідні дані та зміни, що відбуваються під час її виконання. Функція перегляду інформації про номер приймає на вхід запит користувача на отримання даних про конкретний номер.

Система звертається до бази даних, отримує відповідну інформацію з комірки rooms та повертає користувачу повну інформацію про номер, включаючи інформацію доступних додакових послуг. Функція бронювання номера приймає на вхід ідентифікатор номера, дату прибуття та дату відбуття, перевіряє доступність номера на вказаний період, розраховує загальну ціну проживання та створює запис у комірці bookings. Статус номера змінюється на book.

Користувач отримує підтвердження бронювання з деталями та сумою до оплати, а також відповідне сповіщення на електронну пошту, надсилання якого реалізовано за допомогою сервісу Brevo [13]. Функція додавання нового номера доступна лише адміністратору та приймає характеристики номера. Програма створює новий список у комірці rooms та відображає адміністратору оновлений перелік готових номерів. Функція перегляду інформації про бронювання доступна лише адміністратору та користувачу який зробив бронював цього номеру. База повертає повний для адміністратора повний перелік активних бронювань з усіма деталями, відсортованих за датою заїзду у порядку зростання а для користувача лише його поточну бронь. Структуру зовнішнього проєктування системи, побудовану з використанням нотації уніфікованої мови моделювання UML [2], наведено на UML плакаті.

2.3.2 Проєктування логіки програми

Визначення інформаційних зв'язків між компонентами розробки інформаційної системи управління готельним фондом та бронюванням «Montreux Regent Hotel» визначаються багатосистемною структурою застосунку та реалізуються через чітко визначені інтерфейси взаємодії між функціями.

Вірна організація інформаційних зв'язків є одним із ключових факторів, що забезпечують безпеку, розширенність та зручність використатння програмного комплексу. Загальна структура інформаційних зв'язків системи побудована за принципом односпрямованої залежності між продуктами, де кожен продукт взаємодіє лише з безпосередньо суміжнимі не має прямого доступу до інших рівні.

Це дозволяє забезпечити чітку картину відповідальності між компонентами системи та мінімізує ризик виникнення небажаних залежностей, які могли б ускладнити подальший розвиток і підтримку застосунку.

Керування потоками даних у системі відбувається за наступним принципом: серверна частина, реалізована на мові програмування JavaScript з використанням середовища Node.js та фреймворку Express.js, приймає вхідні запити через відповідні контролери, які виступають точкою початку для взаємодії. Після початкової обробки запиту контролер передає отримані дані до сервісної функції, яка реалізує основну логіку системи, у свою чергу вона звертається до репозиторіїв для отримання або збереження даних у базі даних MongoDB, після чого сформована відповідь передається у зворотному напрямку до клієнтської частини.

Зв'язки між окремими компонентами програми організовані відповідно до їх функціонального призначення:

– hotelController приймає запити, пов'язані з інформацією про готель, та взаємодіє з HotelService, який реалізує логіку обробки даних про готель і використовує HotelRepository для роботи з відповідною колекцією у MongoDB;

– roomController відповідає за обробку запитів, пов'язаних з управлінням

номерів, та взаємодіє з RoomService і RoomRepository для отримання, додавання, редагування та видалення інформації про номери готелю;

– bookingController є одним із ключових компонентів системи, оскільки обробляє запити, пов'язані з бронюванням номерів, та взаємодіє з BookingService, який у процесі виокнання нового бронювання використовує одразу BookingRepository для збереження записів про бронювання і RoomRepository для перевірки поточної доступності обраного номера на вказаний період;

– userController відповідає за обробку запитів з управління обліковими записами користувачів, та взаємодіє з UserService і UserRepository для отримання та оновлення даних клієнтів системи.

Передача даних між функціями системи реалізується за допомогою спеціальних об'єктів передачі даних. Використання таких об'єктів дозволяє приховати внутрішню структуру користувачів бази даних від зовнішніх API та надає легкість у виданні відповідей сервера залежно від потреб клієнтської частини.

Контролери отримують вхідні дані у вигляді відповідних об'єктів від клієнтської частини та передають їх до сервісного компоненту для подальшої обробки. Сервіс виконує необхідні зміни між об'єктами та користувачами бази даних, застосовує базові правила та логіку компіляції, після чого передає оброблені дані до репозиторіїв або формує відповідь для клієнта.

Між колекціями бази даних MongoDB встановлено логічні зв'язки через ідентифікатори, що забезпечують цілісність та відповідність системних даних. Часина бронювань містить посилання на ідентифікатор користувача та ідентифікатор номера, що дозволяє отримувати повну інформацію про кожне бронювання одаразу із відповідної інформацією про клієнта та характеристики потрібного номера.

Колекція номерів містить посилання на ідентифікатор готелю, який потрібний для збереження данних про номер та дозволяє швидко отримувати перелік усіх існуючих номерів у готелі. Такий підхід до організації зв'язків між колекціями для ефективної роботи системи та дозволяє виконувати складні

запити до бази даних з мінімальними витратами ресурсів.

2.4 Програмування інформаційної системи

2.4.1 Написання клієнтської частини

Клієнтська частина додатку виконана з використанням бібліотеки React, що дозволяє створювати динамічні веб-сторінки з структурою. Система відображення інтерфейсу виконана на основі окремих компонентів, кожен з яких відповідає за свою частину функціоналу сайту:

– roomCard - віртуальний показ готельного номера, відображаючи його назву, вартість, короткий опис, а також кнопки для перегляду деталей або здійснення бронювання;

– RoomDetails - відповідає за відображення розширеної інформації про вибраний номер;

– addRoom - виконує форму додавання нового готельного номера, яка дозволяє адміністратору вказати тип номера, вартість за добу, перелік доступних додаткових послуг, зокрема безкоштовний Wi-Fi, сніданок, обслуговування номерів, вид на гори та доступ до басейну, а також завантажити до чотирьох фотографій номера;

– myBookings та hotelreg - використані для реалізації додаткових функцій, таких як перегляд бронювань користувача та керування даними готелю для адміністратора.

Переміщення між сторінками додатку робляться за допомогою React Router, що допомагає робити швидкі переходи між розділами без перезаватаження сторінки. Для успішної авторизації користувачів використовується сервіс Clerk, який забезпечує безпечну обробку облікових даних та використання токенів на стороні сервера. Обмін даними між клієнтською та серверною частинами виконується через HTTP-запити з використанням бібліотеки Axios.

2.4.2 Написання серверної частини

Серверна частина застосунку розроблена та розподілена між кількома системами, що забезпечує чітке розмежування відповідальності між окремими модулями та спрощує подальшу підтримку й масштабування системи. Така архітектура дозволяє незалежно розвивати кожен компонент серверної частини без необхідності суттєвого втручання в логіку інших модулів.

Система контролерів містить класи, які визначають REST API систему та відповідають за приймання вхідних HTTP-запитів і формування відповідей у форматі JSON. Кожен контролер відповідає за окрему предметну область функціоналу системи, зокрема за роботу з номерами, бронюваннями та користувачами, що дозволяє підтримувати логічну структурованість коду та полегшує його подальше тестування й налагодження. Лістинг реалізації контролера обробки бронювань наведено у Додатку І – Лістинг файлу «bookingController.js».

Для безпечного обміну даними між клієнтською та серверною частинами реалізовано підтримку CORS, що дозволяє контролювати, з яких джерел дозволяється надсилати запити до сервера, та забезпечує захист від небажаних звернень із сторонніх доменів. Налаштування CORS є важливою складовою забезпечення безпеки веб-застосунку, оскільки без належної конфігурації даного механізму система могла б бути вразливою до атак, пов'язаних із міжсайтовим виконанням запитів. Головний файл застосунку, у якому відбувається ініціалізація основних модулів та підключення проміжного програмного забезпечення, наведено у Додатку В – Лістинг файлу «App.jsx».

Взаємодія між клієнтом і сервером дозволяється через такі базові системи: POST /rooms/create для створення нового готельного номера, GET /rooms для отримання списку всіх доступних номерів, POST /bookings для здійснення бронювання та GET /my-bookings для перегляду власних бронювань користувача. Кожен із зазначених маршрутів реалізований із дотриманням принципів REST архітектури, що забезпечує передбачуваність та зрозумілість взаємодії між

клієнтською та серверною частинами застосунку для будь-якого розробника, що працюватиме із системою в подальшому.

На клієнтській частині звернення до зазначених маршрутів реалізовано через глобальний контекст застосунку, лістинг якого наведено у Додатку Б – Лістинг файлу «AppContext.jsx», що дозволяє централізовано керувати станом даних про номери та бронювання в межах усього інтерфейсу користувача.
Зокрема, для відображення інформації про окремий готельний номер у каталозі реалізовано відповідний компонент, лістинг якого наведено у Додатку А – Лістинг файлу «HotelCard.jsx».

Для перегляду повного переліку доступних номерів використовується окремий компонент, що звертається до маршруту GET /rooms, лістинг якого наведено у Додатку Є – Лістинг файлу «AllRooms.jsx», а для відображення детальної інформації про конкретний номер та можливості його бронювання реалізовано компонент, лістинг якого наведено у Додатку И – Лістинг файлу «RoomDetails.jsx».

Для інтеграції функціоналу оплати у систему підключено платіжну систему Stripe [10], яка є повністю функціональною та готовою до використання та забезпечення зручної обробки фінансових операцій та повернення коштів. Вибір саме цієї платіжної системи обумовлений її широкою популярністю, високим рівнем безпеки проведення транзакцій та наявністю детальної документації, що значно спрощує процес інтеграції. Stripe також дозволяє зберігати webhook для автоматичного оновлення статусів бронювань після успішного проведення оплати, автоматизувати виставлення оплат та зберігання даних клієнтів і транзакцій у захищеному середовищі. Завдяки використанню webhook-механізму система здатна оперативно реагувати на зміну статусу платежу без необхідності постійного опитування сервера, що позитивно впливає на загальну продуктивність застосунку та зменшує навантаження на серверну інфраструктуру.

Перегляд користувачем власних оформлених бронювань та їх статусів оплати реалізовано через окремий компонент, лістинг якого наведено у Додатку З – Лістинг файлу «MyBookings.jsx».

З боку адміністрування системи реалізовано окрему адміністративну панель, через яку здійснюється керування номерами та бронюваннями. Функціонал додавання нового номера до каталогу, що звертається до маршруту POST /rooms/create, реалізовано в окремому компоненті, лістинг якого наведено у Додатку Г – Лістинг файлу «AddRoom.jsx». Загальна структура адміністративної панелі, що об'єднує всі підрозділи керування, наведена у Додатку Ґ – Лістинг файлу «Dashboard.jsx», а перелік доданих номерів із можливістю їх редагування відображається у компоненті, лістинг якого наведено у Додатку Е – Лістинг файлу «ListRoom.jsx». Загальна розмітка сторінок застосунку, що визначає розташування навігаційних елементів та основного контенту, наведена у Додатку Д – Лістинг файлу «Layout.jsx».

Конфігураційні параметри системи, зокрема ключі Stripe, налаштування CORS та параметри підключення до бази даних MongoDB, визначені у конфігураційних файлах середовища виконання. Такий підхід до зберігання конфігураційних даних дозволяє відокремити чутливу інформацію від основного програмного коду та забезпечує можливість гнучкого розгортання застосунку в різних середовищах, зокрема тестовому та робочому, без необхідності внесення змін безпосередньо до коду системи.

Головна сторінка застосунку, з якої розпочинається взаємодія користувача із системою, реалізована у компоненті, лістинг якого наведено у Додатку Ж – Лістинг файлу «Home.jsx».

2.5 Тестування інформаційної системи

Тестування розробки інформаційної системи управління готельним фондом та бронюванням «Montreux Regent Hotel» проводилось шляхом створення середовища, наближеного до реального середовища функціонування програми, а саме інформаційного та програмного. Метою тестування було виявлення та усунення помилок у роботі системи, перевірка правильної обробки вхідних даних, а також підтвердження відповідності готового функціоналу встановленим

вимогам. Якісно проведене тестування є запорукою надійної та стабільної роботи системи в умовах реального використання, тому даному етапу розробки приділялась особлива увага.

Тестування проводилось за такою схемою:

– першим етапом визначались типи тестів, що використовуються для виявлення помилок у різних компонентах системи, а також формувався перелік пріоритетних сценаріїв, які підлягали обов'язковій перевірці в першу чергу;

– другим етапом застосовувались основні правила тестування, а саме прохід усіх основних ділянок та комірок програмного коду, вірність перевірки отриманих результатів, мінімальність обчислень при виконанні тестових завдань та перевірку програми в цілому з урахуванням усіх можливих варіантів взаємодії користувача з системою;

– третім етапом було порівняння отриманих результатів з базовими показниками для перевірки коректної роботи продукту, що дозволяло об'єктивно оцінити відповідність системи встановленим технічним вимогам та функціональним специфікаціям.

Наявність помилок у програмі є допустимим явищем у процесі розробки програмного забезпечення, однак їх своєчасне виявлення та усунення є критично важливим для забезпечення якості кінцевого продукту. У цьому проєкті застосовувались три методи виявлення і попередження помилок:

– перший метод відповідав за підвищення якості процесу розробки шляхом дотримання базових функцій та стандартів програмування, що мінімізувало ймовірність виникнення типових помилок ще на етапі написання коду;

– другим методом було введення у кінці кожної стадії розробки окремого циклу перевірки, що дозволяло виявляти та редагувати помилки на початкових етапах до переходу до наступної частини, тим самим запобігаючи накопиченню технічного боргу та ускладненню процесу налагодження на пізніших стадіях;

– третій метод заключався в орієнтації конкретних процесів тестування

на вірні процеси розробки для виявлення класу помилок, що забезпечувало достовірний та систематизований підхід для перевірки кожного окремого компонента системи незалежно від інших модулів.

У процесі тестування застосовувались такі типи перевірок. Функціональне тестування проводилось для перевірки коректності роботи всіх базових функцій системи, серед яких реєстрація та авторизація користувачів, перегляд інформації про номери, оформлення бронювань та проведення оплат. Даний вид тестування дозволив переконатись у тому, що кожна функція системи працює відповідно до визначених вимог і повертає очікуваний результат при різних варіантах вхідних даних. Структурне тестування застосовувалось для перевірки логіки програмних систем, таких як коректності обробки запитів серед контролерів, правильності виконання правил у сервісній частині та коректності взаємодії з базою даних MongoDB через репозиторії. Завдяки структурному тестуванню вдалось виявити ряд прихованих логічних помилок, які не проявлялись при звичайному використанні системи, проте могли призвести до некоректної роботи в окремих граничних випадках.

Для перевірки коректної роботи серверної частини проводилось тестування REST API за допомогою інструменту Postman [16]. Під час тестування надсилались HTTP-запити різних типів до ключових маркерів системи, аналізувались статуси відповідей, структура та зміст даних у форматі JSON, а також обробка некоректних або неповних запитів. Особлива увага приділялась перевірці граничних випадків, зокрема відправці запитів із відсутніми обов'язковими полями, некоректними типами даних та недійсними токенами авторизації.

Зокрема, перевірялась коректність роботи таких маркерів: POST /rooms/create для створення нового номера, GET /rooms для отримання списку доступних номерів, POST /bookings для здійснення бронювання та GET /my-bookings для перегляду бронювань користувача. Кожен із зазначених маркерів тестувався як у штатному режимі роботи, так і в умовах некоректних або неповних вхідних даних з метою перевірки стійкості системи до помилкових

запитів.

Для клієнтської частини застосунку реалізовано кілька рівнів тестування. Модульні тести розроблено з використанням бібліотеки Jest [17], що дозволило перевірити коректність роботи окремих компонентів React у закритому середовищі без залежності від зовнішніх сервісів та модулів. Завдяки ізольованому характеру модульних тестів стало можливим швидко перевіряти коректність роботи кожного окремого компонента ще на ранніх етапах розробки, не очікуючи завершення розробки суміжних частин застосунку.

Інтеграційне тестування проводилось з використанням React Testing Library для перевірки взаємодії між компонентами та правильної передачі даних між ними, що є особливо важливим в умовах складної компонентної архітектури застосунку.

Даний рівень тестування дозволив переконатись у тому, що окремі компоненти, які успішно проходили модульні тести самостійно, так само коректно функціонують і в межах спільної взаємодії один з одним у реальних умовах роботи інтерфейсу.

Повне тестування виконувалось за допомогою інструменту Cypress [18], що дозволило відтворювати реальні дії користувача в інтерфейсі застосунку, серед них функція перегляду номерів, оформлення бронювання та проведення оплати. Використання Cypress забезпечило можливість автоматизованої перевірки повних користувацьких сценаріїв від початку до кінця, що значно підвищило достовірність результатів тестування та дозволило виявити недоліки, які могли залишитись непоміченими при ізольованій перевірці окремих компонентів чи їх з'єднань.

Першим етапом відбувається перевірка можливості реєстрації нового користувача на сайті. Тестується процес введення особистих даних та підтвердження акаунту через сервіс автентифікації Clerk.

Проводиться контроль коректності відображення форм реєстрації на кожному кроці створення облікового запису, аналізується реакція системи на некоректно введені дані, зокрема на порожні поля, невалідні формати електронної

пошти та недостатню довжину пароля.

Також оцінюється загальна стабільність роботи процесу входу та виходу з облікового запису користувача в різних сценаріях використання, що дозволяє переконатись у надійності даного етапу взаємодії користувача із системою ще до переходу до подальшого функціоналу застосунку (див. рис. 2.7).

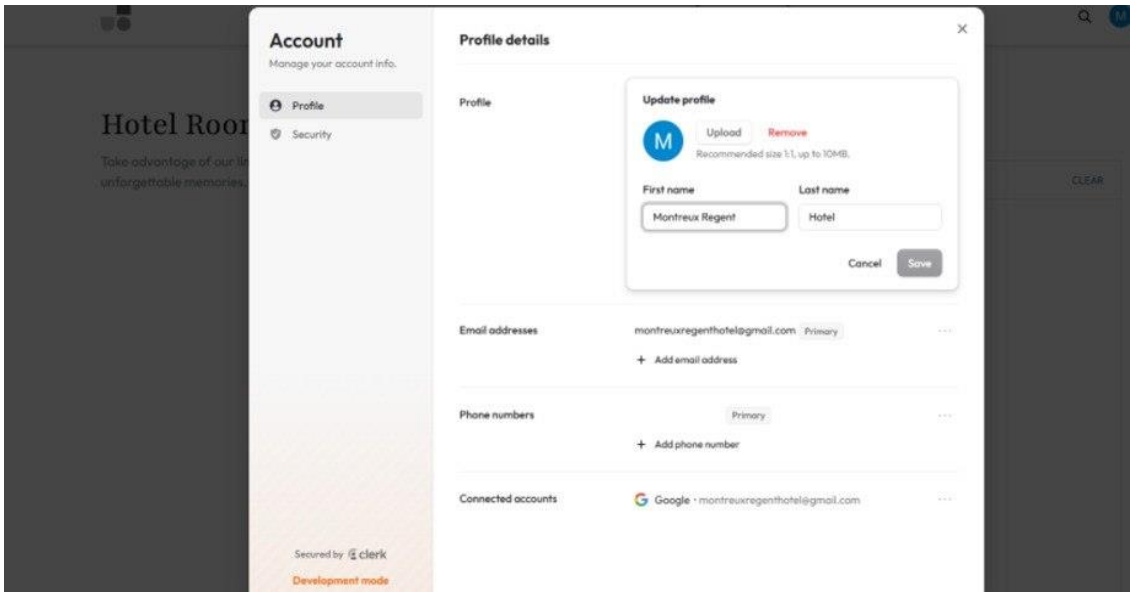


Рисунок 2.7 – Тестування авторизації

Другим етапом проводиться перевірка коректного відображення даних у базі даних MongoDB.

Аналізується відповідність введених користувачем даних тим, що були збережені в базі, правильність збереження імені, електронної пошти та ідентифікатора акаунту.

Здійснюється контроль цілісності структури документів у відповідних колекціях бази даних, а також проводиться перевірка відсутності дублювання записів при повторних спробах реєстрації з однаковими даними.

Особлива увага приділяється коректності зв'язку між обліковим записом у Clerk та відповідним записом у базі даних MongoDB, що є ключовим для забезпечення узгодженості даних між різними рівнями системи (див. рис 2.8).

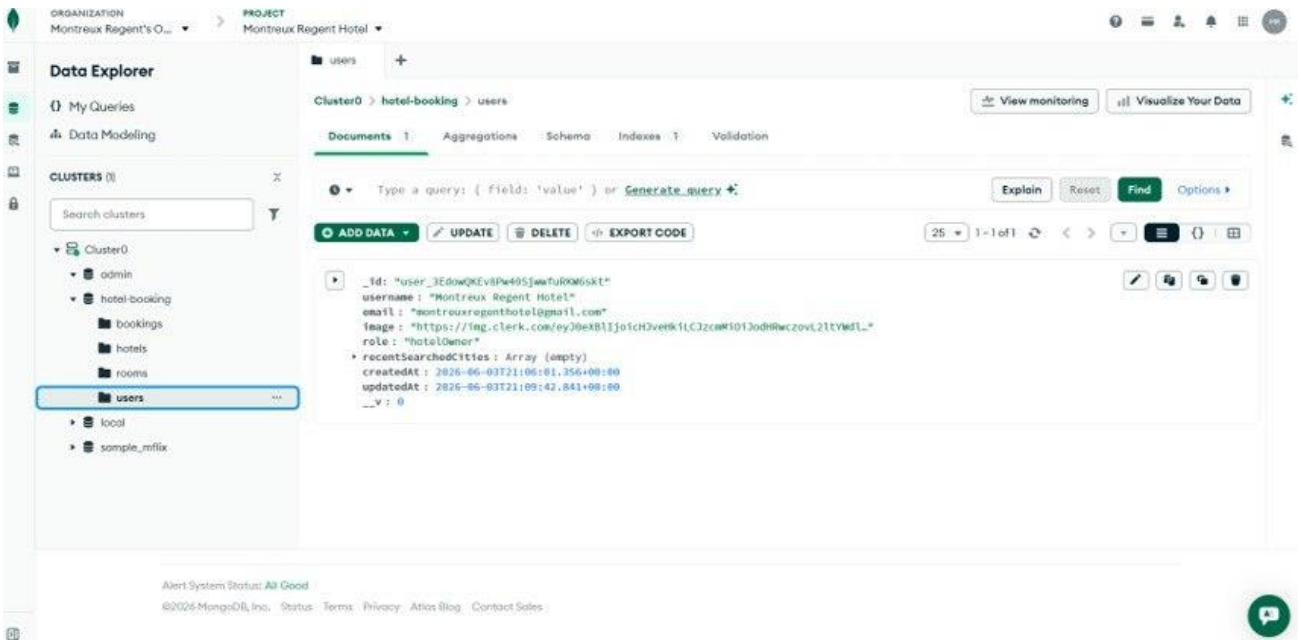


Рисунок 2.8 – Тестування авторизації БД

Третім етапом починається перевірка коректної роботи платіжної системи Stripe. Тестується проходження тестових транзакцій із використанням тестових карткових даних, що надаються самою платформою Stripe для цілей розробки та налагодження, що дозволяє повністю імітувати реальний процес оплати без необхідності використання справжніх банківських карток та проведення фактичних фінансових операцій (див. рис 2.9).

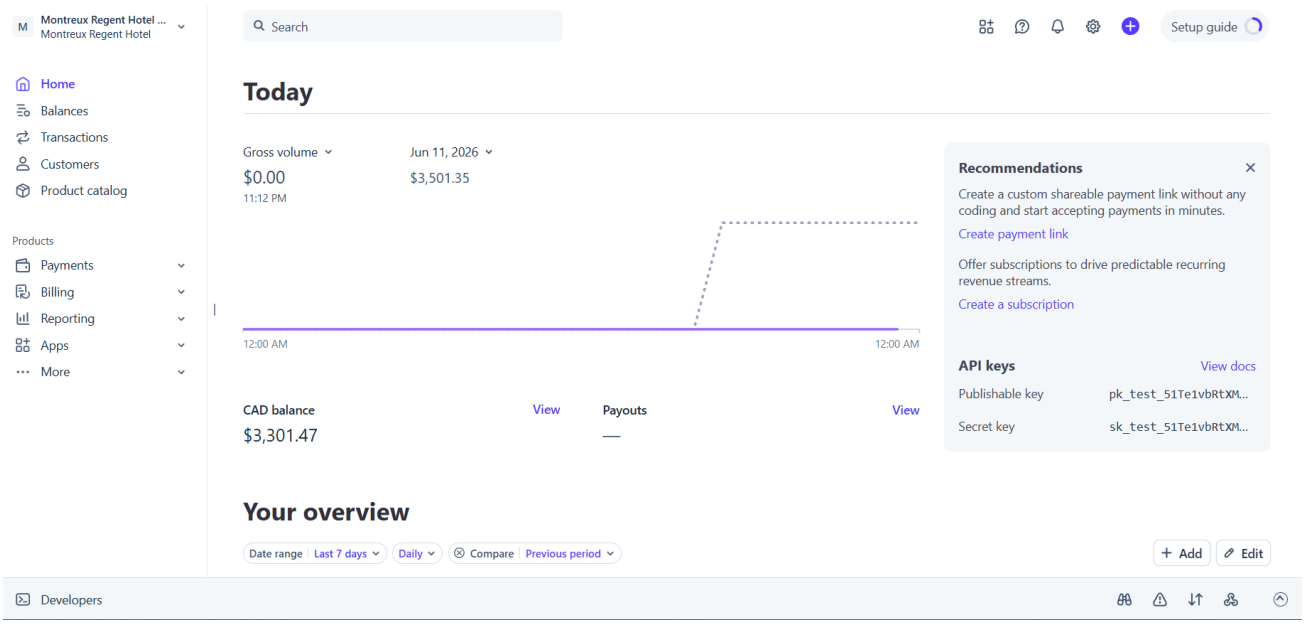


Рисунок 2.9 – Платіжна система

Налагодження програми виконувалось одночасно з процесом тестування. У випадку виявлення помилки у логічній конфігурації запитів або відображенні даних проводився детальний аналіз причин їх виникнення з подальшим внесенням необхідних змін до вихідного коду. Для покращення безпеки та надійності було реалізовано централізовану обробку випадкових ситуацій, яка забезпечувала використання зрозумілих та інформативних повідомлень про помилки для клієнтської частини, що суттєво спрощувало процес діагностики проблем як для розробників, так і для кінцевих користувачів системи.

Усі виявлені в ході тестування помилки та неточності були усунені до фінальної здачі роботи, що підтверджує відповідність розробленої системи встановленим вимогам якості.

3 СПЕЦІАЛЬНИЙ РОЗДІЛ

3.1 Інструкція з розміщення інформаційної системи в Інтернеті

Для коректного функціонування розробки інформаційної системи управління готельним фондом та бронюванням «Montreux Regent Hotel» необхідно забезпечити відповідну конфігурацію технічних та програмних засобів.

Мінімальна конфігурація комплексу для розгортання та функціонування системи передбачає наявність комп'ютера з процесором з тактовою частотою не менше 2.0 ГГц, оперативною пам'яттю обсягом не менше 4 ГБ, вільним місцем на жорсткому диску не менше 2 ГБ, а також стабільним підключенням до мережі. Для комфортної роботи з адміністративною панеллю рекомендується використовувати монітор з роздільною здатністю не менше 1280/720 пікселів.

Для функціонування серверної частини на сервері має бути встановлено Java Development Kit версії 17 або вище, що є обов'язковою умовою для запуску застосунку на базі Spring Boot. Для управління залежностями та збірки проєкту використовується система Apache Maven, яка має бути встановлена та налаштована у середовища операційної системи.

База даних MongoDB має бути встановлена та запущена, при цьому рекомендується використовувати версію 6.0 або вище. Як альтернативу локальній установці MongoDB допускається використання онлайн сервісу MongoDB Atlas з відповідним налаштуванням підключення у файлі застосунку.

Для клієнтської частини необхідна наявність Node.js версії 18 або вище та менеджер пакетів npm, який входить до складу Node.js. Клієнтську частину застосунку реалізовано на платформі Vercel [14], що виконує автоматичну реалізацію внесення змін до репозиторію та надає публічну адресу для доступу до застосунку.

Спочатку необхідно отримати вихідний код проєкту з репозиторію. Після цього у конфігураційному файлі application необхідно вказати параметри підключення до бази даних MongoDB, рядок підключення, назву бази даних

також порт на якому працюватиме сервер. Далі необхідно вказати ключі для інтеграції з платіжним сервісом Stripe у відповідних змінних середовища. Після налаштування конфігураційних файлів виконується збір проєкту за допомогою команди Maven, після чого запускається отриманий виконуваний файл.За замовчуванням серверна частина працює на порту 8080.

Формування клієнтської частини на платформі Vercel виконується шляхом підключення репозиторію до проєкту на платформі та налаштування змінних середовища,адреси серверного API та ключів сервісу Clerk для забезпечення аутентифікації користувачів. Після усспішного розгортання клієнтська частина буде доступа за публічною адресою, яка надається платформою Vercel.

Системне програмне забезпечення, в якому може використовуватись даний готельний комплекс включає мінімальні операційні системи:

1. Windows 10;
2. Ubuntu 20.04 ;
3. MacOS 12.

Для доступу до інтерфейсу системи зі сторони користувача потрібен лише любий веб-браузера, зокрема:

1. Google Chrome;
2. Mozilla Firefox;
3. Microsoft Edge.

3.2 Інструкція з обслуговування та наповнення інформаційної системи

Для забезпечення коректної та повноцінної роботи інформаційної системи управління готельним фондом та бронюванням «Montreux Regent Hotel» адміністратор має можливість самостійно наповнювати та обслуговувати систему через спеціально розроблену адміністративну панель. Дана панель є окремим захищеним розділом застосунку, доступ до якого надається виключно користувачам із роллю адміністратора після проходження процедури авторизації

через сервіс Clerk. Така організація доступу дозволяє унеможливити несанкціоноване втручання у структуру номерного фонду чи перегляд конфіденційної інформації про бронювання сторонніми особами.

Після успішного входу в систему адміністратор потрапляє до головного розділу адміністративної панелі Dashboard. Даний розділ надає узагальнену інформацію про поточний стан системи, зокрема загальну кількість оформлених бронювань, сумарний дохід готелю за весь період роботи системи, а також перелік останніх бронювань із зазначенням імені користувача, назви забронюваного номера, загальної суми та поточного статусу оплати.

Завдяки наявності даного розділу адміністратор має змогу оперативно відстежувати ключові показники роботи готелю без необхідності звернення до бази даних безпосередньо чи використання сторонніх засобів аналітики.

Інформація на панелі оновлюється в режимі реального часу, що дозволяє адміністратору завжди мати актуальну картину активних бронювань в готелі та фінансових показників (див. рис. 3.1).

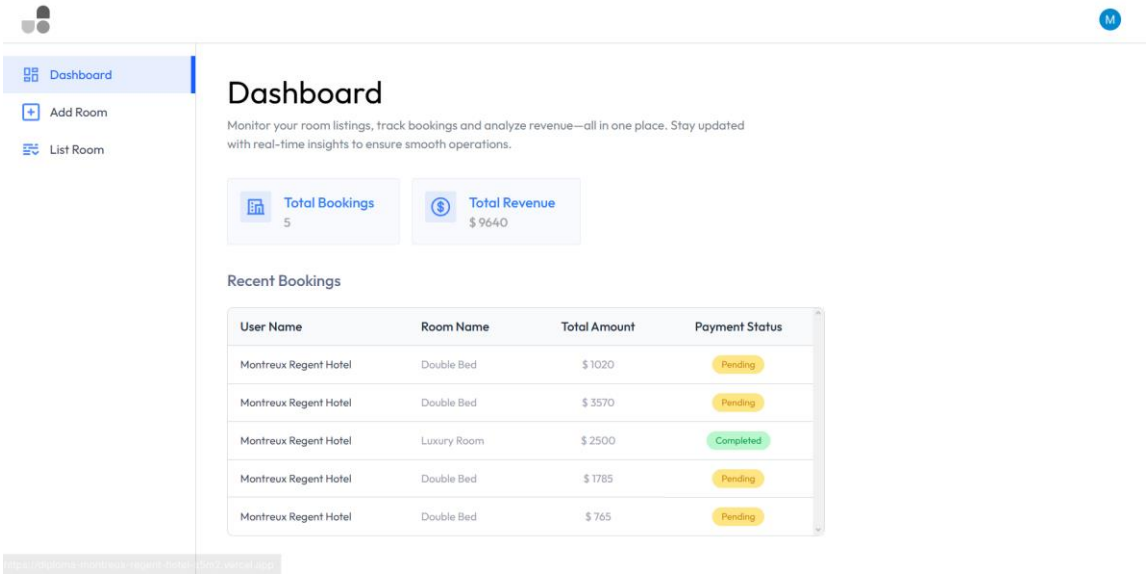


Рисунок 3.1 – Панель адміністратора

Наповнення системи інформацією про номерний фонд здійснюється через розділ Add Room, перехід до якого відбувається за допомогою відповідного

пункту бокового меню навігації.

Інтерфейс даного розділу спроєктовано таким чином, щоб процес додавання нового номера був максимально зрозумілим та інтуїтивним навіть для адміністратора без технічної підготовки див.рис 3.2.

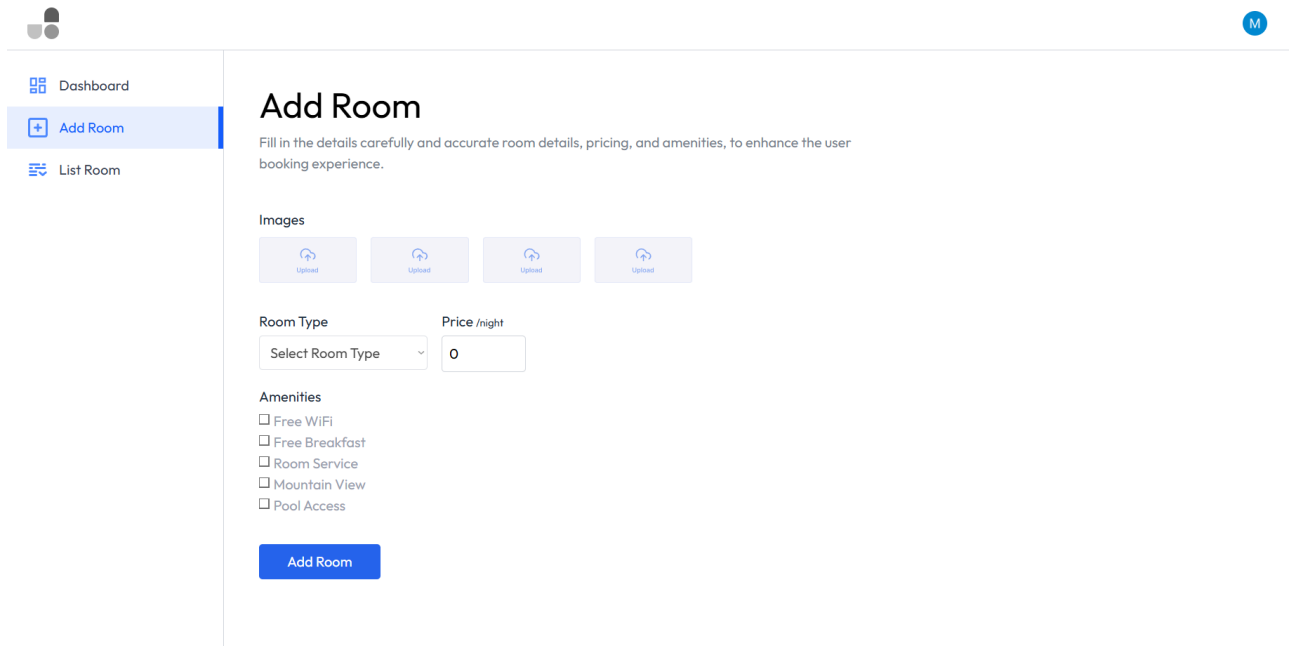


Рисунок 3.2 – Додавання нового номера

Для додавання нового готельного номера адміністратору необхідно виконати наступну послідовність дій:

– завантажити зображення номера у відповідні поля розділу Images, при цьому система передбачає можливість завантаження до чотирьох фотографій для кожного номера з метою якомога повнішого візуального представлення приміщення потенційним клієнтам;

– обрати тип номера у випадаючому списку Room Type, що визначає категорію готельного номера серед наявних у системі варіантів та впливає на подальше відображення номера у відповідних фільтрах каталогу;

– вказати вартість проживання за одну добу у відповідному полі Price, яка надалі використовується системою для автоматичного розрахунку загальної вартості бронювання залежно від обраного клієнтом періоду проживання;

– позначити доступні зручності у блоці Amenities, серед яких безкоштовний Wi-Fi, безкоштовний сніданок, обслуговування номерів, вид на гори та доступ до басейну, що дозволяє клієнтам більш усвідомлено обирати номер відповідно до власних потреб;

– підтвердити додавання номера натисканням кнопки Add Room, після чого введені дані передаються на сервер через маршрут POST /rooms/create та зберігаються у базі даних MongoDB.

Після успішного додавання номер автоматично стає доступним для перегляду в каталозі номерів на клієнтській частині застосунку, а також відображається у розділі List Room адміністративної панелі, де адміністратор має можливість переглянути перелік усіх доданих номерів та за необхідності відредагувати чи видалити будь-який із них. Подібний підхід до організації наповнення системи значно полегшує процес масштабування номерного фонду готелю в майбутньому, оскільки не вимагає залучення розробників для додавання чи зміни інформації про номери.

У разі необхідності внесення змін до інформації про вже доданий номер адміністратор переходить до розділу List Room, обирає потрібний номер зі списку та вносить відповідні корективи до його характеристик, зокрема може змінити ціну, перелік зручностей чи фотографії, після чого зміни автоматично синхронізуються з базою даних та практично миттєво відображаються на клієнтській частині застосунку.

Контроль за оформленими бронюваннями та статусами оплати здійснюється безпосередньо в розділі Dashboard, де у таблиці останніх бронювань адміністратор може відстежувати, які платежі перебувають у статусі очікування, а які вже були успішно завершені. Така організація роботи з системою дозволяє адміністратору ефективно підтримувати актуальність інформації про номерний фонд та оперативно реагувати на нові бронювання без необхідності залучення сторонніх інструментів чи додаткового програмного забезпечення для обслуговування готельного бізнесу.

3.3 Інструкція з популяризації та підтримки інформаційної системи

Для перевірки правильної роботи розробки інформаційної системи управління готельним фондом та бронюванням «Montreux Regent Hotel» розроблено набір тестів, що використовує всі основні функціональні можливості системи. Використання тестових наборів дозволяє виявити помилки у роботі розробленого програмного забезпечення і підтвердити відповідність реалізованого функціоналу встановленим вимогам. Регулярне проведення тестування є важливою складовою підтримки системи протягом усього періоду її експлуатації, оскільки дозволяє своєчасно виявляти потенційні проблеми ще до того, як вони вплинуть на роботу кінцевих користувачів та призведуть до фінансових чи репутаційних втрат для готелю.

У процесі тестування системи застосовуються наступні типи тестів:

– аварійні призначені для перевірки поведінки системи в екстремальних або нештатних ситуаціях, зокрема при втраті з'єднання з базою даних чи недоступності платіжного сервісу;

– формальні використовуються для перевірки граничних значень вхідних даних, наприклад максимально допустимої довжини текстових полів чи мінімальної та максимальної вартості номера;

– комплексні включають перевірку взаємодії між кількома компонентами системи одночасно, зокрема узгодженості роботи модулів авторизації, бронювання та оплати в межах єдиного користувацького сценарію;

– основні призначені для перевірки коректності роботи ключових функцій системи при введенні коректних вхідних даних у звичайних умовах експлуатації;

– стикові використовуються для перевірки коректності взаємодії між клієнтською та серверною частинами системи, зокрема правильності передачі даних через API;

– структурні спрямовані на перевірку внутрішньої логіки програмних компонентів незалежно від зовнішнього інтерфейсу системи;

– функціональні - проводяться для підтвердження відповідності реалізованого функціоналу вимогам технічного завдання в цілому.

Кожен із зазначених типів тестів відіграє свою специфічну роль у загальному процесі забезпечення якості системи, а їх комплексне застосування дозволяє охопити перевіркою як окремі програмні модулі, так і систему в цілому з урахуванням усіх можливих сценаріїв використання. У разі розбіжності між отриманими та базовими результатами проводиться детальний аналіз причин виникнення помилки та вносяться відповідні зміни до коду з подальшим повторним тестуванням проблемної ділянки до повного усунення виявлених недоліків.

Розробка інформаційної системи управління готельним фондом та бронюванням «Montreux Regent Hotel» розрахована на три типи користувачів, кожен з яких має власний набір функціональних можливостей та рівень доступу до системи:

– гостя;
– клієнта
– адміністратора.

Доступ до системи виконується через браузер за адресою, що надається платформою Vercel, без необхідності встановлення додаткового програмного забезпечення на пристрій користувача. Для коректної роботи системи рекомендується використовувати актуальні версії браузерів: Google Chrome, Mozilla Firefox або Microsoft Edge зі стабільним підключенням до мережі, оскільки використання застарілих версій браузерів може призводити до некоректного відображення окремих елементів інтерфейсу або обмеженої підтримки сучасних веб-технологій, застосованих у розробці системи.

Для початку роботи з системою користувач має виконати реєстрацію, заповнити відповідну форму із зазначенням персональних даних та паролю.

Процес реєстрації спроєктовано таким чином, щоб мінімізувати кількість обов'язкових полів та зробити вхід до системи максимально швидким для нового користувача.

Користувач, який ще не пройшов реєстрацію, потрапляє до системи у статусі гостя. Гість має обмежений рівень доступу до функціоналу системи та може лише переглядати перелік доступних готельних номерів разом із їхніми характеристиками, фотографіями та вартістю проживання, проте не має можливості здійснити бронювання чи скористатись сервісом оплати до моменту проходження реєстрації.

Таким чином, перегляд каталогу номерів є відкритим для будь-якого відвідувача сайту, тоді як подальша взаємодія з системою, зокрема оформлення бронювання, можлива лише після створення облікового запису.

Після успішної реєстрації користувач отримує доступ до функціоналу клієнтської частини системи. Авторизація реалізована через сервіс Clerk через введення електронної адреси та паролю, після чого система видає роль користувача та перенаправляє його до відповідного розділу, який йому доступний відповідно до призначеної ролі.

Клієнт після авторизації отримує доступ до наступних функцій

- перегляду доступних номерів;
- отримання детальної інформації про обраний номер;
- оформлення бронювання;
- сервісу оплати;
- перегляду власних бронювань.

Адміністратор після авторизації отримує доступ до адміністративної панелі Dashboard, яка надає розширені можливості управління системою для:

- перегляду повного переліку номерного фонду;
- додавання нового номера;
- редагування існуючого номера;
- перегляду інформації про наявні бронювання та внесення змін.

Рекомендації з розміщення командних та інформаційних файлів системи передбачають наступну організацію: вихідний код серверної частини розміщується у відповідному репозиторії та організований відповідно до

стандартної структури Maven-проєкту, що забезпечує зрозумілість структури проєкту для будь-якого розробника, знайомого із загальноприйнятими стандартами Java-розробки. Конфігураційні файли серверної частини, зокрема application.properties із параметрами підключення до бази даних MongoDB та налаштуваннями сервера, розташовані у каталозі src/main/resources.

Змінні середовища, що містять ключі платіжного сервісу Stripe та інші конфіденційні параметри, зберігаються у захищеному середовищі та не включаються до репозиторію вихідного коду, що відповідає загальноприйнятим практикам безпеки під час розробки веб-застосунків та запобігає випадковому витоку чутливих даних у разі публічної доступності репозиторію.

Конфігураційні файли клієнтської частини, що містять адресу серверного API та ключі сервісу Clerk, налаштовуються у відповідному розділі платформи Vercel як змінні середовища, що дозволяє оперативно змінювати конфігурацію застосунку без необхідності повторного розгортання коду.

Обсяг завантаженого модуля серверної частини застосунку після збірки за допомогою Maven становить орієнтовно 121 КБ, що включає всі необхідні залежності та вбудований сервер, необхідний для автономного функціонування серверної частини. Для коректної роботи серверної частини системи рекомендується використовувати не менше ніж 512 МБ оперативної пам'яті, що забезпечить стабільну обробку запитів від користувачів у базовому режимі роботи без істотних затримок чи відмов у обслуговуванні навіть за умов помірного навантаження на систему.

4 ЕКОНОМІЧНИЙ РОЗДІЛ

Метою економічної частини кваліфікаційної роботи є проведення економічних розрахунків, спрямованих на визначення економічної ефективності розробки інформаційної системи управління готельним фондом та бронюванням «Montreux Regent Hotel», прийняття рішення про подальший розвиток і впровадження або ж недоцільність проведення відповідної розробки.

Об’єктом розробки є інформаційна система управління готельним фондом та бронюванням «Montreux Regent Hotel».

Розрахунок вартості розробки виконується в декілька етапів:

– описати технологічний процес розробки із зазначенням трудомісткості кожної операції;

– визначити суму витрат на оплату праці основного і допоміжного персоналу, включаючи відрахування на соціальні заходи;

– обчислити витрати на електроенергію;

– нарахувати суму амортизаційних відрахувань;

– визначити суму накладних витрат;

– скласти кошторис та визначити собівартість робіт;

– розрахувати ціну робіт;

– визначити економічну ефективність та термін окупності.

4.1 Визначення стадій технологічного процесу та загальної тривалості проведення НДР

В цьому підрозділі розглянемо основні етапи технологічного процесу для розробки інформаційної системи управління готельним фондом та бронюванням «Montreux Regent Hotel». Для визначення загальної тривалості проведення робіт доцільно дані витрат часу по окремих операціях технологічного процесу звести у таблицю.

Таблиця 4.1 - Середній час виконання робіт по обслуговуванню та стадії (операції) технологічного процесу

№ п/п	Назва операції (стадії)	Виконавець	Середній час виконання операції, год.
1	Планування та аналіз	Кер. проєкту Pm	8
		Інженер (l1)	8
2	Розробка технічного завдання	Кер. проєкту (Pm)	7
		Інженер (l1)	8
3	Дизайн інтерфейсу	Інженер (l1)	16
		Інженер (l2)	24
4	Розробка функціоналу	Інженер (l1)	48
5	Тестування та відладка	Тестувальник	8
6	Документування	Інженер (l1)	2
7	Розгортання та підтримка	Інженер (l2)	16
8	Управління проєктом	Кер. проєкту (Pm)	24
Разом			169

Сумарний час виконання операцій технологічного процесу становить 169 годин.

4.2 Визначення витрат на оплату праці та відрахувань на соціальні заходи

У даному підрозділі проводиться аналіз і розрахунок витрат, пов'язаних з оплатою праці та відрахуваннями на соціальні заходи, що необхідні для розробки нформаційної системи управління готельним фондом та бронюванням «Montreux RegentHotel»

Розмір заробітної плати залежить від складності та умов виконуваної роботи, професійно-ділових якостей працівника, результатів його праці та

діяльності підприємства.

Основна заробітна плата розраховується за формулою:

$$З_{осн.} = Т_с * К_г \quad (4.1)$$

де: T_c – тарифна ставка, грн. (приймаємо для керівника проєкту (Pm) – 450 грн./год, інженера (l2) – 272 грн./год.), інженера (l1) – 113 грн./год., тестувальник – 100 грн./год.; $К_г$ – кількість відпрацьованих годин.

Отже, основна заробітна плата для:

Керівника проекту (Pm) $З_{осн}$= 39 * 450 = 17 550 грн.

Інженера (l2) $З_{осн}$= 40 * 272 = 10 880 грн.

Інженера (l1) $З_{осн}$= 82 * 113 = 9 266 грн.

Тестувальник $З_{осн}$= 8 * 100 = 800 грн.

Сумарна основна заробітна плата становить

$$З_{осн} = 17\,550+10\,880+ 9\,266+800 = 38\,496 \text{ грн.}$$

Додаткова заробітна плата становить 10 – 15 % від суми основної заробітної плати.

$$З_{дод.} = З_{осн.} \cdot К_{допл.} \quad (4.2)$$

де: $К_{допл.}$ – коефіцієнт додаткових виплат працівникам.

Отже додаткова заробітна плата по категоріях працівників становить:

Керівника проекту $З_{дод}$= 17 550 * 0,1 = 1 755 грн.

Інженера (l2) $З_{дод}$= 10 880 * 0,1 = 1 088 грн.

Інженера (l1) $З_{дод}$= 9 266 * 0,1 = 926 грн.

Тестувальник $З_{дод}$= 800 * 0,1 = 80 грн.

Загальна додаткова заробітна плата становить:

$$З_{дод}= 1\,755 + 1\,088 + 926 + 80 = 3\,849 \text{ грн.}$$

Звідси загальні витрати на оплату праці ($B_{о.п.}$) визначаються за формулою:

$$В_{о.п.} = З_{осн.} + З_{дод.} \quad (4.3)$$

$$В_{о.п}= 38\,496 + 3\,849 = 42\,345 \text{ грн.}$$

Єдиний соціальний внесок (ЄСВ – 22%) визначається за формулою:

$$В_{єсв} = Воп * 0,22 \quad (4.4)$$

$$В_{єсв} = 42\,345 * 0,22 = 9\,315 \text{ грн.}$$

Проведені розрахунки витрат на оплату праці наведено у таблиці 4.2.

Таблиця 4.2 – Зведені розрахунки витрат на оплату праці

№ п/п	Категорія працівників	Основна заробітна плата, грн.			Додаткова заробітна плата, грн.	ЄСВ, грн.	Всього витрати на оплату праці, грн. 6 = 3+4+5
		Тарифна ставка, грн.	К-сть годин	Фактично нарах. зарплати, грн.			
1	Кер. проєкту (Pm)	450	39	17 550	1755		
2	Інженера (l2)	272	40	10 880	1088		
3	Інженера (l1)	113	82	9 266	926		
4	Тестувальник	100	8	800	80		
Разом				38 496	3 849	9 316	51 661

Отже, загальні витрати на оплату праці становлять 51 661 грн.

4.3 Розрахунок витрат на електроенергію

Розрахуємо вартість електроенергії. Затрати на електроенергію 1-ці обладнання визначаються за формулою:

$$З_в = W * T * S \quad (4.5)$$

де: W – необхідна потужність, кВт; T – кількість годин роботи обладнання; S – вартість кіловат-години електроенергії (приймаємо 15,94 грн).

В нашій системі є 1 ПК. Витрати на електроенергію для цього комп'ютера обчислимо окремо, взявши за основу, що час роботи обладнання обчислюється в залежності від виконуваних робіт (згідно табл. 4.1) і споживані потужності наступні: комп'ютер – 0,85 кВт/год.

$$З_{ек} = 0,85*169*15,94 = 2\ 290 \text{ грн.}$$

Витрати на електроенергію становлять 2 290 грн.

4.4 Розрахунок суми амортизаційних відрахувань під час розробки інформаційної системи «Montreux RegentHotel».

Характерною особливістю застосування основних фондів у процесі виробництва є їх відновлення. Для відновлення засобів праці у натуральному виразі необхідне їх відшкодування у вартісній формі, яке здійснюється шляхом амортизації.

Амортизація – це процес перенесення вартості основних фондів на вартість новоствореної продукції з метою їх повного відновлення

Комп'ютери та оргтехніка належать до четвертої групи основних фондів.

Амортизація на них нараховується лише в випадку, якщо мінімально допустимі строки їх корисного використання 2 роки. Для визначення амортизаційних відрахувань застосовуємо формулу:

$$A = \frac{Б_В * Н_А}{100\%} * T, \quad (4.6)$$

де: А – амортизаційні відрахування за звітний період, грн.;

$Б_В$ – балансова вартість групи основних фондів на початок звітного періоду грн.;

$Н_А$ – норма амортизації, 0,04 %.

Оскільки для написання програми та її тестування використовується один ПК, вартістю 50000,00 грн., то сума амортизаційних відрахувань становитиме:

$$A = \frac{50\ \ 000{,}00 * 0{,}04}{150} * 169 = 2\ \ 253\ \ \text{грн.}$$

4.5 Обчислення накладних витрат

Накладні витрати пов’язані з обслуговуванням виробництва, утриманням апарату управління підприємства (фірми) та створення необхідних умов праці.

В залежності від організаційно-правової форми діяльності господарюючого суб'єкта, накладні витрати можуть становити 35% від суми основної та додаткової заробітної плати працівників.

$$Н_в=В_{о.п.} * 0,35 \quad (4.7)$$

де: $Н_В$ – накладні витрати.

$$Н_В= 42\ 345 *0,35= 14\ 821 \text{ грн.}$$

4.6 Складання кошторису витрат та визначення собівартості під час розробки інформаційної системи «Montreux RegentHotel»

Для складення кошторису витрат та визначення собівартості, результати проведених вище розрахунків зведемо у таблиці 4.3.

Таблиця 4.3 - Кошторис витрат під час розробки інформаційної системи «Montreux RegentHotel».

№	Зміст витрат	Сума, грн.	В % до загальної суми
1.	Витрати на оплату праці	51 661	73
2.	Витрати на електроенергію	2 290	3
3.	Амортизаційні відрахування	2 253	3
4.	Накладні витрати	14 821	31
5.	Собівартість	71 025	100

Собівартість ($С_В$) НДР розрахуємо за формулою:

$$Св = Во.п + Вс.з + Зе + А + Нв \quad (4.8)$$

Отже, собівартість дорівнює $С_В$= 71 025 грн.

4.7 Розрахунок ціни під час розробки інформаційної системи «Montreux RegentHotel»

Розрахунок ціни науково-дослідної роботи включає в себе урахування різноманітних факторів, таких як рівень рентабельності, собівартість та податкова ставка.

Ціну робіт можна визначити за формулою:

$$Ц = Св * (1+Р_{рен}) * (1+ПДВ), \quad (4.9)$$

де: Св – собівартість; $Р_{рен.}$ – рівень рентабельності; ПДВ – ставка податку на додану вартість.

Ц = 71 025*(1+0,32)*(1+0,2)= 112 503 грн.

4.8 Визначення економічної ефективності і терміну окупності капітальних вкладень

Ефективність виробництва – це узагальнене і повне відображення кінцевих результатів використання робочої сили, засобів та предметів праці на підприємстві за певний проміжок часу.

Для визначення ефективності продукту розраховують чисту теперішню вартість (ЧТВ) і термін окупності (Ток).

$$ЧТВ = -Св + \sum_{i=1}^{t} \frac{Г_П}{(1+i)^t}, \quad (4.10)$$

де: Св – собівартість розробки; $Г_п$ – грошовий потік за t – ий рік; t – відповідний рік проекту; і – величина дисконтної ставки (10…15%).

$$ЧТВ = -71\ \ 025 + \frac{41478}{(1+0,1)^1} + \frac{41478}{(1+0,1)^2} + \frac{41478}{(1+0,1)^3} \ \ = 32147 \ \ грн$$

Якщо ЧТВ ≥ 0, то проект може бути рекомендований до впровадження.

Термін окупності визначається за формулою:

$$T_{ОК} = T_{ПВ} + \frac{Н_В}{Г_{ПР}} \quad (4.11)$$

де: $Т_{ПВ}$ – період до повного відшкодування витрат, років; $Н_В$ – невідшкодовані витрати на початок року, грн.; $Г_{ПР}$ – грошовий потік на початок року, грн.

$$Т_{ок} = 2 + \frac{1011}{41478} = 2{,}2р.$$

Всі дані внесемо в зведену таблицю 4.4.

Таблиця 4.4 – Техніко-економічні показники під час розробки інформаційної системи «Montreux RegentHotel».

№ п/п	Показник	Значення
1.	Собівартість, грн.	71 025
2.	Плановий прибуток або грошовий потік, грн.	41 478
3.	Ціна, грн.	112 503
4.	Чиста теперішня вартість, грн.	32 147
5.	Термін окупності, рік	2,2

Прибутковість проєкту та термін окупності свідчать про його фінансову ефективність та здатність повернути капітальні вкладення протягом 2026 року. Отже, на основі отриманих показників можна зробити висновок, що витрати під час розробки системи управління готельним фондом та бронюванням «Montreux RegentHotel» є доцільними з економічної точки зору.

5 ОХОРОНА ПРАЦІ, ТЕХНІКА БЕЗПЕКИ ТА ЕКОЛОГІЧНІ ВИМОГИ

5.1 Гарантія прав на охорону праці

Охорона праці - це система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів та засобів, спрямованих на збереження життя, здоров'я і працездатності людини у процесі трудової діяльності.

Відповідно до Закону України Про охорону праці кожен працівник має право на належні, безпечні та здорові умови праці. Держава гарантує кожному громадянину захист його права на працю в умовах, що відповідають вимогам безпеки та гігієни.

Роботодавець зобов'язаний:

– забезпечити на робочому місці умови праці, що відповідають нормативно-правовим актам з охорони праці;

– проводити інструктажі з охорони праці для всіх працівників;

– надавати працівникам засоби індивідуального захисту відповідно до встановлених норм;

– своєчасно інформувати працівників про наявні на виробництві шкідливі та небезпечні фактори;

– забезпечувати проходження обов'язкових медичних оглядів працівниками, які працюють в умовах підвищеної небезпеки;

– відшкодовувати збитки у разі ушкодження здоров'я працівника внаслідок виробничої травми чи професійного захворювання.

Працівник у свою чергу має право:

– відмовитися від виконання роботи у разі виникнення загрози його життю або здоров'ю без будь-яких негативних наслідків для себе;

– розірвати трудовий договір за власним бажанням, якщо роботодавець не виконує вимог законодавства про охорону праці;

– звернутися до органів державного нагляду за охороною праці у разі

порушення своїх прав.

Держава гарантує нагляд та контроль за дотриманням законодавства про охорону праці через Державну службу України з питань праці та інші уповноважені органи. За порушення вимог охорони праці передбачена адміністративна та кримінальна відповідальність.

5.2 Шкідливі речовини, що виділяються при паянні. Вимоги до вентиляції робочого місця

Паяння є одним із поширених технологічних процесів у виробництві електронного обладнання та радіоапаратури. Незважаючи на удавану простоту, цей процес супроводжується виділенням цілого ряду шкідливих речовин, що становлять небезпеку для здоров'я працівника.

Під час паяння у повітря робочої зони виділяються такі шкідливі речовини:

– пари свинцю та олова - основних компонентів припою, які є токсичними важкими металами;

– пари каніфолі та флюсів - викликають подразнення дихальних шляхів та алергічні реакції;

– оксид вуглецю - утворюється при термічному розкладанні флюсів та може спричинити отруєння;

– аерозолі припою - дрібні частинки металу, що потрапляють до дихальних шляхів;

– альдегіди та органічні кислоти - продукти розкладання флюсів при високих температурах.

Тривалий вплив цих речовин на організм людини може спричинити хронічне отруєння важкими металами, захворювання дихальних шляхів.

Вимоги до вентиляції робочого місця паяльника є одними з найважливіших умов безпечної роботи. Робоче місце обов'язково повинно бути обладнане місцевою витяжною вентиляцією, яка видаляє шкідливі пари та аерозолі безпосередньо від місця їх утворення. Витяжний зонт або відсмоктувач

розміщується на відстані не більше 300–350 мм від місця паяння.

Вимоги до вентиляційної системи:

– швидкість повітря у зоні видалення шкідливих речовин має бути не менше 0,5 м/с;

– загальнообмінна вентиляція приміщення повинна забезпечувати не менше 10-кратний повітрообмін на годину;

– система вентиляції повинна регулярно перевірятися та обслуговуватися;

– фільтри витяжної вентиляції необхідно очищати або замінювати відповідно до встановленого графіка;

– приплив свіжого повітря має компенсувати обсяг видаленого забрудненого повітря.

Забороняється виконувати паяльні роботи у приміщеннях без належної вентиляції або при несправній вентиляційній системі. Додатково рекомендується використовувати засоби індивідуального захисту: респіратор для захисту органів дихання та захисні окуляри.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи досягнуто поставленої мети — розроблено повнофункціональну інформаційну систему управління готельним фондом та бронюванням «Montreux Regent Hotel». Отримані результати дозволяють зробити такі висновки:

1) Проведено аналітичний огляд існуючих рішень (сайтів готелів Hilton Garden Inn та JAG Hotel), на основі якого сформульовано ключові вимоги до розробки: зручний інтерфейс, наявність пошуку та фільтрації номерів, якісний фотоконтент, детальні описи та адаптивний дизайн.

2) Сформовано технічне завдання, визначено функціональні та нефункціональні вимоги до системи, а також обґрунтовано вибір технологічного стеку: React.js — для клієнтської частини, Node.js та Express.js — для серверної частини, MongoDB — для зберігання даних, із залученням сторонніх сервісів Clerk, Stripe, Cloudinary та Brevo.

3) Розроблено архітектуру інформаційної системи, побудовано UML-діаграму варіантів використання, визначено доменну модель, а також структуру контролерів, сервісів і репозиторіїв.

4) Реалізовано клієнтську частину застосунку на базі React, включаючи головну сторінку, каталог номерів з фільтрацією та сортуванням, сторінку окремого номера з формою бронювання, профіль користувача та панель адміністратора.

5) Реалізовано серверну частину застосунку на основі Node.js та Express.js з підтримкою REST API; інтегровано сервіси авторизації (Clerk), обробки онлайн-платежів (Stripe), зберігання зображень (Cloudinary) та email-розсилки (Brevo).

6) Здійснено функціональне, структурне та модульне тестування клієнтської й серверної частин системи, за результатами якого виявлені недоліки усунено до завершення розробки.

7) Виконано розгортання системи на платформі Vercel, що забезпечило

публічний доступ до застосунку.

Таким чином, розроблена інформаційна система автоматизує основні процеси готелю — бронювання номерів, управління номерним фондом, облік клієнтів та проведення онлайн-оплати — і повністю відповідає сучасним вимогам до швидкодії, зручності використання та адаптивності інтерфейсу. Система готова до практичного впровадження та може бути використана для реальної автоматизації роботи готелю «Montreux Regent Hotel».

ПЕРЕЛІК ПОСИЛАНЬ

1) Марціяш Г.Я., Слободян Р.О. Методичні вказівки до виконання кваліфікаційної роботи для здобувачів фахової передвищої освіти спеціальності 122 «Комп'ютерні науки». Тернопіль : ВСП «ТФК ТНТУ ім. І. Пулюя», 2024, 48с.

2) OMG Unified Modeling Language (UML) : офіційна специфікація. URL: https://www.omg.org/spec/UML (дата звернення: 17.04.2026).

3) Casciaro M., Mammino L. Node.js Design Patterns, 3rd Edition. Birmingham : Packt Publishing, 2020. 664 p.

4) Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps, 2nd Edition. Sebastopol : O'Reilly Media, 2020.

5) Bradshaw S., Brazil E., Chodorow K. MongoDB: The Definitive Guide, 3rd Edition. Sebastopol : O'Reilly Media, 2019. 511 p.

6) Node.js Documentation : вебсайт. URL: https://nodejs.org/en/docs (дата звернення: 10.04.2026).

7) Express – Node.js web application framework : вебсайт. URL: https://expressjs.com (дата звернення: 10.04.2026).

8) React Documentation : вебсайт. URL: https://react.dev (дата звернення: 17.04.2026).

9) MongoDB Documentation : вебсайт. URL: https://www.mongodb.com/docs (дата звернення: 17.04.2026).

10) Stripe Documentation : вебсайт. URL: https://stripe.com/docs (дата звернення: 24.04.2026).

11) Clerk Documentation : вебсайт. URL: https://clerk.com/docs (дата звернення: 24.04.2026).

12) Cloudinary Documentation : вебсайт. URL: https://cloudinary.com/documentation (дата звернення: 02.05.2026).

13) Brevo Documentation : вебсайт. URL: https://developers.brevo.com (дата звернення: 02.05.2026).

14) Vercel Documentation : вебсайт. URL: https://vercel.com/docs (дата

звернення: 09.05.2026).

15) MDN Web Docs : вебсайт. URL: https://developer.mozilla.org (дата звернення: 09.05.2026).

16) Postman Learning Center : вебсайт. URL: https://learning.postman.com (дата звернення: 16.05.2026).

17) Jest Documentation : вебсайт. URL: https://jestjs.io/docs/getting-started (дата звернення: 16.05.2026).

18) Cypress Documentation : вебсайт. URL: https://docs.cypress.io (дата звернення: 23.05.2026).

ДОДАТКИ

Додаток А. Лістинг файлу «Hotel Card.jsx»»

```
import React from 'react'
import { Link } from 'react-router-dom'
import { assets } from '../assets/assets'
import { useAppContext } from '../context/AppContext';

const HotelCard = ({room,index}) => {

    const { currency } = useAppContext();

    return (
          <Link to={'/rooms/' + room._id} onClick={() => scrollTo(0, 0)} key={room._id} className='relative max-w-70 w-full rounded-xl overflow-hidden bg-white text-gray-500/90 shadow-[0px_4px_4px_rgba(0,0,0,0.05)]'>
            <img src={room.images[0]} alt="hotel-img" draggable="false" />
              {index % 2 === 0 && <p className='px-3 py-1 absolute top-3 left-3 text-xs bg-white text-gray-800 font-medium rounded-full'>Best Seller</p>}
            <div className='p-4 pt-5'>
                <div className='flex items-center justify-between'>
                     <p className='font-playfair text-xl font-medium text-gray-800'>{room.hotel.name}</p>
                    <div className='flex items-center gap-1'>
                        <img src={assets.starIconFilled} alt="star-icon" />
4.5
                    </div>
                </div>
                <div className='flex items-center gap-1 text-sm'>
                    <img src={assets.locationIcon} alt="location-icon" />
                    <span>{room.hotel.address}</span>
                </div>
```

```
                <div className='flex items-center justify-between mt-4'>
                                    <p><span  className='text-xl  text-gray-
800'>{currency}{room.pricePerNight}</span>/night</p>
                     <button className='px-4 py-2 text-sm font-medium border
border-gray-300    rounded    hover:bg-gray-50    transition-all    cursor-
pointer'>Book Now</button>
                </div>
            </div>
        </Link>
    )
}

export default HotelCard
```

Додаток Б. Лістинг файлу «AppContext.jsx»

```
import { useAuth, useUser } from "@clerk/clerk-react";
import { createContext, useContext, useEffect, useState } from "react";
import axios from "axios";
import { toast } from 'react-hot-toast'
import { useNavigate } from "react-router-dom";
import { assets } from "../assets/assets";

axios.defaults.baseURL = import.meta.env.VITE_BACKEND_URL;

const AppContext = createContext();

export const AppProvider = ({ children }) => {

    const currency = import.meta.env.VITE_CURRENCY || "$";
    const navigate = useNavigate();
    const { user } = useUser();
    const { getToken } = useAuth()

    const [isOwner, setIsOwner] = useState(false);
    const [showHotelReg, setShowHotelReg] = useState(false);
    const [rooms, setRooms] = useState([]);
    const [searchedCities, setSearchedCities] = useState([]);

    const facilityIcons = {
        "Free WiFi": assets.freeWifiIcon,
        "Free Breakfast": assets.freeBreakfastIcon,
        "Room Service": assets.roomServiceIcon,
        "Mountain View": assets.mountainIcon,
        "Pool Access": assets.poolIcon,
    };

    const fetchUser = async () => {
```

```
        try {
              const { data } = await axios.get('/api/user', { headers: {
Authorization: `Bearer ${await getToken()}` } })
            if (data.success) {
                setIsOwner(data.role === "hotelOwner");
                setSearchedCities(data.recentSearchedCities)
            } else {

                setTimeout(() => {
                    fetchUser();
                }, 2000);
            }
        } catch (error) {
            toast.error(error.message)
        }
    }

    const fetchRooms = async () => {
        try {
            const { data } = await axios.get('/api/rooms')
            if (data.success) {
                setRooms(data.rooms)
            }
            else {
                toast.error(data.message)
            }
        } catch (error) {
            toast.error(error.message)
        }
    }

    useEffect(() => {
        if (user) {
            fetchUser();
```

```
        }
    }, [user]);

    useEffect(() => {
        fetchRooms();
    }, []);

    const value = {
        currency, navigate,
        user, getToken,
        isOwner, setIsOwner,
        axios,
        showHotelReg, setShowHotelReg,
        facilityIcons,
        rooms, setRooms,
        searchedCities, setSearchedCities
    };

    return (
        <AppContext.Provider value={value}>
            {children}
        </AppContext.Provider>
    );

};

export const useAppContext = () => useContext(AppContext);
```

Додаток В. Лістинг файлу «App.jsx»

```
import React from 'react'
import { Route, Routes, useLocation } from 'react-router-dom'
import Home from './pages/Home'
import Navbar from './components/Navbar'
import Layout from './pages/hotelOwner/Layout'
import Dashboard from './pages/hotelOwner/Dashboard'
import AddRoom from './pages/hotelOwner/AddRoom'
import ListRoom from './pages/hotelOwner/ListRoom'
import HotelReg from './components/HotelReg'
import { useAppContext } from './context/AppContext'
import { Toaster } from 'react-hot-toast'
import AllRooms from './pages/AllRooms'
import RoomDetails from './pages/RoomDetails'
import Footer from './components/Footer'
import MyBookings from './pages/MyBookings'
import Loader from './components/Loader'

const App = () => {

  const isOwnerPath = useLocation().pathname.includes("owner");

  const { showHotelReg } = useAppContext();

  return (
    <div className='font-inter'>
      <Toaster />
      {!isOwnerPath && <Navbar />}
      {showHotelReg && <HotelReg />}
      <div className='min-h-[70vh]'>
      <Routes>
        <Route path='/' element={<Home />} />
        <Route path='/rooms' element={<AllRooms />} />
```

```
        <Route path='/rooms/:id' element={<RoomDetails />} />
        <Route path='my-bookings' element={<MyBookings />} />
        < Route path="/loader/:nextUrl" element={<Loader />} />
        <Route path="/owner" element={<Layout />}>
          <Route index element={<Dashboard />} />
          <Route path="add-room" element={<AddRoom />} />
          <Route path="list-room" element={<ListRoom />} />
        </Route>
      </Routes>
      </div>
      <Footer />
    </div>
  )
}

export default App
```

Додаток Г. Лістинг файлу «AddRoom.jsx»

```
import React, { useState } from 'react'
import { assets } from '../../assets/assets'
import Title from '../../components/Title'
import toast from 'react-hot-toast'
import { useAppContext } from '../../context/AppContext'

const AddRoom = () => {

    const { axios, getToken } = useAppContext()

     const [images, setImages] = useState({ 1: null, 2: null, 3: null, 4:
null })
    const [loading, setLoading] = useState(false);

    const [inputs, setInputs] = useState({
        roomType: '',
        pricePerNight: 0,
        amenities: {
            'Free WiFi': false,
            'Free Breakfast': false,
            'Room Service': false,
            'Mountain View': false,
            'Pool Access': false
        }
    })

    const onSubmitHandler = async (e) => {
        e.preventDefault()
         if (!inputs.roomType || !inputs.pricePerNight || !inputs.amenities
|| !Object.values(images).some(image => image)) {
            toast.error("Please fill in all the details")
            return;
```

```
        }
        setLoading(true);
        try {
            const formData = new FormData()
            formData.append('roomType', inputs.roomType)
            formData.append('pricePerNight', inputs.pricePerNight)
           Amenities
             const amenities = Object.keys(inputs.amenities).filter(key =>
inputs.amenities[key])
            formData.append('amenities', JSON.stringify(amenities))

            Object.keys(images).forEach((key) => {
                images[key] && formData.append('images', images[key])
            })

             const { data } = await axios.post('/api/rooms/', formData, {
headers: { Authorization: `Bearer ${await getToken()}` } })

            if (data.success) {
                toast.success(data.message)
                setInputs({
                    roomType: '',
                    pricePerNight: 0,
                    amenities: {
                        'Free WiFi': false,
                        'Free Breakfast': false,
                        'Room Service': false,
                        'Mountain View': false,
                        'Pool Access': false
                    }
                })
                setImages({ 1: null, 2: null, 3: null, 4: null })
            } else {
                toast.error(data.message)
```

```
            }

        } catch (error) {
            toast.error(error.message)
        } finally {
            setLoading(false);
        }
    }

    return (
        <form onSubmit={onSubmitHandler}>
                    <Title  align='left'  font='outfit'  title='Add  Room'
subTitle='Fill in the details carefully and accurate room details, pricing,
and amenities, to enhance the user booking experience.' />
            {/* Upload Area For Images */}
            <p className='text-gray-800 mt-10'>Images</p>
            <div className='grid grid-cols-2 sm:flex gap-4 my-2 flex-wrap'>
                {Object.keys(images).map((key) => (
                    <label key={key} htmlFor={`roomImage${key}`}>
                        <img className='max-h-13 cursor-pointer opacity-80'
src={images[key]  ?  URL.createObjectURL(images[key])  :  assets.uploadArea}
alt="" />
                                        <input  type="file"  accept='image/*'
id={`roomImage${key}`} hidden
                                onChange={e => setImages({ ...images, [key]:
e.target.files[0] })} />
                    </label>
                ))}
            </div>

            <div className='w-full flex max-sm:flex-col sm:gap-4 mt-4'>

                <div className='flex-1 max-w-48'>
                    <p className='text-gray-800 mt-4'>Room Type</p>
```

```
                    <select className='border opacity-70 border-gray-300
mt-1 rounded p-2 w-full' value={inputs.roomType} onChange={(e) =>
setInputs({ ...inputs, roomType: e.target.value })}>
                        <option value=''>Select Room Type</option>
                        <option value='Single Bed'>Single Bed</option>
                        <option value='Double Bed'>Double Bed</option>
                        <option value='Luxury Room'>Luxury Room</option>
                        <option value='Family Suite'>Family Suite</option>
                    </select>
                </div>

                <div>
                        <p  className='mt-4  text-gray-800'>Price  <span
className='text-xs'>/night</span></p>
                    <input type="number" placeholder='0' className='border
border-gray-300 mt-1 rounded p-2 w-24' value={inputs.pricePerNight}
onChange={(e) => setInputs({ ...inputs, pricePerNight: e.target.value })}
/>
                </div>

            </div>

            <p className='text-gray-800 mt-4'>Amenities</p>
            <div className='flex flex-col flex-wrap mt-1 text-gray-400 max-
w-sm'>
                {Object.keys(inputs.amenities).map((amenity, index) => (
                    <div key={index}>
                        <input type='checkbox' id={`amenities${index + 1}`}
checked={inputs.amenities[amenity]}
                                    onChange={() => setInputs({ ...inputs,
amenities: { ...inputs.amenities, [amenity]: !inputs.amenities[amenity] }
})}
                        />
                        <label htmlFor={`amenities${index + 1}`}> {amenity}
```

```
</label>
                    </div>
                ))}
            </div>

            <button className='bg-primary text-white px-8 py-2 rounded mt-8
cursor-pointer' disabled={loading}>
                {loading ? "Adding..." : "Add Room"}
            </button>
        </form>
    )
}

export default AddRoom
```

Додаток Ґ. Лістинг файлу «Dashboard.jsx»

```
import React, { useEffect, useState } from 'react'
import { assets } from '../../assets/assets'
import Title from '../../components/Title';
import { useAppContext } from '../../context/AppContext';

const Dashboard = () => {

    const { currency, user, getToken, toast, axios } = useAppContext();

    const [dashboardData, setDashboardData] = useState({
        bookings: [],
        totalBookings: 0,
        totalRevenue: 0,
    });

    const fetchDashboardData = async () => {
        try {
               const { data } = await axios.get('/api/bookings/hotel', {
headers: { Authorization: `Bearer ${await getToken()}` } })
            if (data.success) {
                setDashboardData(data.dashboardData)
            } else {
                toast.error(data.message)
            }
        } catch (error) {
            toast.error(error.message)
        }
    }

    useEffect(() => {
        if (user) {
            fetchDashboardData();
```

```
        }
    }, [user]);

    return (
        <div>
                    <Title  align='left'  font='outfit'  title='Dashboard'
subTitle='Monitor your room listings, track bookings and analyze revenue—
all in one place. Stay updated with real-time insights to ensure smooth
operations.' />
            <div className='flex gap-4 my-8'>
                    <div  className='bg-primary/3  border  border-primary/10
rounded flex p-4 pr-8'>
                                    <img   className='max-sm:hidden   h-10'
src={assets.totalBookingIcon} alt="" />
                    <div classNam='flex flex-col sm:ml-4 font-medium'>
                                <p  className='text-blue-500  text-lg'>Total
Bookings</p>
                               <p  className='text-neutral-400  text-base'>{
dashboardData.totalBookings }</p>
                    </div>
                </div>
                    <div  className='bg-primary/3  border  border-primary/10
rounded flex p-4 pr-8'>
                                    <img   className='max-sm:hidden   h-10'
src={assets.totalRevenueIcon} alt="" />
                    <div classNam='flex flex-col sm:ml-4 font-medium'>
                                <p  className='text-blue-500  text-lg'>Total
Revenue</p>
                                    <p  className='text-neutral-400  text-
base'>{currency} { dashboardData.totalRevenue }</p>
                    </div>
                </div>
            </div>
```

```
                <h2 className='text-xl text-blue-950/70 font-medium mb-
5'>Recent Bookings</h2>

            <div className='w-full max-w-3xl text-left border border-gray-
300 rounded-lg max-h-80 overflow-y-scroll'>
                <table className='w-full' >
                    <thead className='bg-gray-50'>
                        <tr>
                            <th className='py-3 px-4 text-gray-800 font-
medium'>User Name</th>
                            <th className='py-3 px-4 text-gray-800 font-
medium max-sm:hidden'>Room Name</th>
                            <th className='py-3 px-4 text-gray-800 font-
medium text-center'>Total Amount</th>
                            <th className='py-3 px-4 text-gray-800 font-
medium text-center'>Payment Status</th>
                        </tr>
                    </thead>
                    <tbody className='text-sm'>
                        {
                            dashboardData.bookings.map((item, index) => (
                                <tr key={index}>
                                    <td className='py-3 px-4 text-gray-700
border-t border-gray-300'>{item.user.username}</td>
                                    <td className='py-3 px-4 text-gray-400
border-t border-gray-300 max-sm:hidden'>{item.room.roomType}</td>
                                    <td className='py-3 px-4 text-gray-400
border-t border-gray-300 text-center'>{currency} {item.totalPrice}</td>
                                        <td className='py-3 px-4  border-t
border-gray-300 flex'>
                                        <button className={`py-1 px-3 text-
xs rounded-full mx-auto ${item.isPaid ? "bg-green-200 text-green-600" :
"bg-amber-200 text-yellow-600"}`}>
                                            {item.isPaid ? "Completed" :
```

```
"Pending"}
                                        </button>
                                    </td>
                                </tr>
                            ))
                        }
                    </tbody>
                </table>
            </div>

        </div>
    )
}

export default Dashboard
```

Додаток Д. Лістинг файлу «Layout.jsx»

```
import React, { useEffect } from 'react'
import Navbar from '../../components/hotelOwner/Navbar'
import Sidebar from '../../components/hotelOwner/Sidebar'
import { Outlet } from 'react-router-dom'
import { useAppContext } from '../../context/AppContext'

const Layout = () => {

    const { isOwner, navigate } = useAppContext()

    useEffect(() => {
        if (!isOwner) {
            navigate('/')
        }
    }, [isOwner])

    return (
        <div className='flex flex-col h-screen'>
            <Navbar />
            <div className='flex h-full'>
                <Sidebar />
                <div className='flex-1 p-4 pt-10 md:px-10 h-full'>
                    <Outlet />
                </div>
            </div>
        </div>
    )
}

export default Layout
```

Додаток Е. Лістинг файлу «ListRoom.jsx»

```
import React, { useEffect } from 'react'
import Title from '../../components/Title'
import { useAppContext } from '../../context/AppContext';
import toast from 'react-hot-toast';

const ListRoom = () => {

    const { axios, getToken, user } = useAppContext()
    const [rooms, setRooms] = React.useState([])

    const fetchRooms = async () => {
        try {
            const { data } = await axios.get('/api/rooms/owner', { headers:
{ Authorization: `Bearer ${await getToken()}` } })
            if (data.success) {
                setRooms(data.rooms)
            }
            else {
                toast.error(data.message)
            }
        } catch (error) {
            toast.error(error.message)
        }
    }

    const toggleAvailability = async (roomId) => {
        const { data } = await axios.post("/api/rooms/toggle-availability",
{ roomId }, { headers: { Authorization: `Bearer ${await getToken()}` } })
        if (data.success) {
            toast.success(data.message)
            fetchRooms()
        } else {
```

```
            toast.error(data.message)
        }
    }


    useEffect(() => {
        if (user) {
            fetchRooms()
        }
    }, [user])

    return (
        <div>
                 <Title  align='left'  font='outfit'  title='Room  Listings' subTitle='View, edit, or manage all listed rooms. Keep the information up-to-date to provide the best experience for users.' />
            <p className='text-gray-500 mt-8'>Total Hotels</p>

             <div className='w-full max-w-3xl text-left border border-gray-300 rounded-lg max-h-80 overflow-y-scroll mt-3'>
                <table className='w-full' >
                    <thead className='bg-gray-50 '>
                        <tr>
                              <th className='py-3 px-4 text-gray-800 font-medium'>Name</th>
                              <th className='py-3 px-4 text-gray-800 font-medium max-sm:hidden'>Facility</th>
                              <th className='py-3 px-4 text-gray-800 font-medium'>Price / night</th>
                              <th className='py-3 px-4 text-gray-800 font-medium text-center'>Actions</th>
                        </tr>
                    </thead>
                    <tbody className='text-sm'>
```

```
                        {
                            rooms.map((item, index) => (
                                <tr key={index}>
                                    <td className='py-3 px-4 text-gray-700
border-t border-gray-300'>{item.roomType}</td>
                                    <td className='py-3 px-4 text-gray-400
border-t border-gray-300 max-sm:hidden'>{item.amenities.join(', ')}</td>
                                    <td className='py-3 px-4 text-gray-400
border-t border-gray-300'>{item.pricePerNight}</td>
                                        <td className='py-3 px-4  border-t
border-gray-300 text-center text-sm text-red-500'>
                                         <label className="relative inline-
flex items-center cursor-pointer text-gray-900 gap-3">
                                                <input type="checkbox"
className="sr-only  peer"  onChange={()  =>  toggleAvailability(item._id)}
checked={item.isAvailable} />
                                                 <div className="w-12 h-7 bg-
slate-300  rounded-full  peer  peer-checked:bg-blue-600  transition-colors
duration-200"></div>
                                                <span className="dot absolute
left-1 top-1 w-5 h-5 bg-white rounded-full transition-transform duration-
200 ease-in-out peer-checked:translate-x-5"></span>
                                        </label>
                                    </td>
                                </tr>
                            ))
                        }
                    </tbody>
                </table>
            </div>
        </div>
    )
}
export default ListRoom
```

Додаток Є. Лістинг файлу «AllRooms.jsx»

```
import { useState, useMemo } from 'react'
import { assets } from '../assets/assets'
import { useAppContext } from '../context/AppContext'
import StarRating from '../components/StarRating'
import { useSearchParams } from 'react-router-dom'

const CheckBox = ({ label, selected = true, onChange = () => { } }) => {
    return (
        <label className="flex gap-3 items-center cursor-pointer mt-2 text-
sm">
            <input type="checkbox" checked={selected} onChange={(e) =>
onChange(e.target.checked, label)} />
            <span className='font-light select-none'>{label}</span>
        </label>
    )
}
const RadioButton = ({ label, selected = true, onChange = () => { } }) => {
    return (
        <label className="flex gap-3 items-center cursor-pointer mt-2 text-
sm">
            <input type="radio" name="sortOption" checked={selected}
onChange={() => onChange(label)} />
            <span className="font-light select-none">{label}</span>
        </label>
    );
}

const AllRooms = () => {

    const [searchParams, setSearchParams] = useSearchParams();

    const { facilityIcons, navigate, rooms, currency } = useAppContext();
```

```
const [openFilters, setOpenFilters] = useState(false);

const [selectedFilters, setSelectedFilters] = useState({
    roomType: [],
    priceRange: [],
});
const [selectedSort, setSelectedSort] = useState('');

const roomTypes = [
    "Single Bed",
    "Double Bed",
    "Luxury Room",
    "Family Suite",
];

const priceRanges = [
    '0 to 500',
    '500 to 1000',
    '1000 to 2000',
    '2000 to 3000',
];

const sortOptions = [
    "Price Low to High",
    "Price High to Low",
    "Newest First"
];

const handleFilterChange = (checked, value, type) => {
    setSelectedFilters((prevFilters) => {
        const updatedFilters = { ...prevFilters };
        if (checked) {
            updatedFilters[type].push(value);
        } else {
```

```
                updatedFilters[type] = updatedFilters[type].filter(item =>
item !== value);
            }
            return updatedFilters;
        });
    }

    const handleSortChange = (sortOption) => {
        setSelectedSort(sortOption);
    }

    const matchesRoomType = (room) => {
                return    selectedFilters.roomType.length    ===    0    ||
selectedFilters.roomType.includes(room.roomType);
    };

    const matchesPriceRange = (room) => {
                return    selectedFilters.priceRange.length    ===    0    ||
selectedFilters.priceRange.some(range => {
            const [min, max] = range.split(' to ').map(Number);
            return room.pricePerNight >= min && room.pricePerNight <= max;
        });
    };

    const sortRooms = (a, b) => {
        if (selectedSort === 'Price Low to High') {
            return a.pricePerNight - b.pricePerNight;
        }
        if (selectedSort === 'Price High to Low') {
            return b.pricePerNight - a.pricePerNight;
        }
        if (selectedSort === 'Newest First') {
            return new Date(b.createdAt) - new Date(a.createdAt);
        }
```

```
        return 0;
    };

    const filterDestination = (room) => {
        const destination = searchParams.get('destination');
        if (!destination) return true;
                                                                    return
room.hotel.city.toLowerCase().includes(destination.toLowerCase());
    }

    const filteredRooms = useMemo(() => {
        return rooms
                            .filter(room    =>    matchesRoomType(room)    &&
matchesPriceRange(room) && filterDestination(room))
            .sort(sortRooms);
    }, [rooms, selectedFilters, selectedSort, searchParams]);

    const clearFilters = () => {
        setSelectedFilters({
            roomType: [],
            priceRange: [],
        });
        setSelectedSort('')
        setSearchParams({});
    }

    return (
           <div  className='flex  flex-col-reverse  lg:flex-row  items-start
justify-between pt-28 md:pt-35 px-4 md:px-16 lg:px-24 xl:px-32'>
            <div>
                {/* Main Title */}
                <div className="flex flex-col items-start text-left">
                           <h1  className='font-playfair  text-4xl  md:text-
[40px]'>Hotel Rooms</h1>
```

```
                    <p className='text-sm md:text-base text-gray-500/90 mt-2 max-w-174'>Take advantage of our limited-time offers and special packages to enhance your stay and create unforgettable memories.</p>
                </div>

                {filteredRooms.map((room) => (
                     <div key={room._id} className='flex flex-col md:flex-row items-start py-10 gap-6 border-b border-gray-300 last:pb-30 last:border-0'>
                        {/* Room Image */}
                           <img title='View Room Details' onClick={() => { navigate(`/rooms/${room._id}`); scrollTo(0, 0) }} src={room.images[0]} alt="hotel-img" className='max-h-65 md:w-1/2 rounded-xl shadow-lg object-cover cursor-pointer' />
                        <div className='md:w-1/2 flex flex-col gap-2'>
                                                <p  className='text-gray-500'>{room.hotel.city}</p>
                                                  <p  onClick={() => { navigate(`/rooms/${room._id}`); scrollTo(0, 0) }} className='text-gray-800 text-3xl font-playfair cursor-pointer' title='View Room Details'>{room.hotel.name}</p>
                            <div className='flex items-center'>
                                <StarRating />
                                <p className='ml-2'>200+ reviews</p>
                            </div>
                              <div className='flex items-center gap-1 text-gray-500 mt-2 text-sm'>
                                                <img  src={assets.locationIcon} alt="location-icon" />
                                <span>{room.hotel.address}</span>
                            </div>
                            {/* Room Amenities */}
                            <div className='flex flex-wrap items-center mt-3 mb-6 gap-4'>
```

```
                                {room.amenities.map((item, index) => (
                                    <div key={index} className='flex items-
center gap-2 px-3 py-2 rounded-lg bg-[#F5F5FF]/70'>
                                        <img src={facilityIcons[item]}
alt={item} className='w-5 h-5' />
                                        <p className='text-xs'>{item}</p>
                                    </div>
                                ))}
                            </div>
                            {/* Room Price per Night */}
                               <p className='text-xl font-medium text-gray-
700'>${room.pricePerNight} /night</p>
                        </div>
                    </div>
                ))}
            </div>

            {/* Filters */}
            <div className="bg-white w-80 border border-gray-300 text-gray-
600 max-lg:mb-8 min-lg:mt-16">
                <div className={`flex items-center justify-between px-5 py-
2.5 min-lg:border-b border-gray-300 ${openFilters && "border-b"}`}>
                        <p  className='text-base  font-medium  text-gray-
800'>FILTERS</p>
                    <div className='text-xs cursor-pointer'>
                         <span onClick={() => setOpenFilters(!openFilters)}
className='lg:hidden'>
                            {openFilters ? "HIDE" : "SHOW"}
                        </span>
                            <span  onClick={clearFilters} className='hidden
lg:block'>CLEAR</span>
                    </div>
                </div>
                   <div  className={`${openFilters  ?  "h-auto"  :  "h-0 lg:h-
```

```
auto"} overflow-hidden transition-all duration-700`}>
                    <div className='px-5 pt-5'>
                              <p  className='font-medium  text-gray-800  pb-
2'>Popular filters</p>
                         {roomTypes.map((room, index) => (
                                        <CheckBox  key={index}  label={room}
selected={selectedFilters.roomType.includes(room)}  onChange={(checked)  =>
handleFilterChange(checked, room, 'roomType')} />
                         ))}
                    </div>
                    <div className='px-5 pt-5'>
                         <p className='font-medium text-gray-800 pb-2'>Price
Range</p>
                         {priceRanges.map((range, index) => (
                                  <CheckBox key={index} label={`${currency}
${range}`}            selected={selectedFilters.priceRange.includes(range)}
onChange={(checked) => handleFilterChange(checked, range, 'priceRange')} />
                         ))}</div>
                    <div className="px-5 pt-5 pb-7">
                          <p className="font-medium text-gray-800 pb-2">Sort
By</p>
                         {sortOptions.map((option, index) => (
                                   <RadioButton  key={index}  label={option}
selected={selectedSort       ===       option}       onChange={()       =>
handleSortChange(option)} />
                         ))}
                    </div>
               </div>
          </div>
        </div>
    )
}

export default AllRooms;
```

Додаток Ж. Лістинг файлу «Home.jsx»

```
import React from 'react'
import Hero from '../components/Hero'
import FeaturedDestination from '../components/FeaturedDestination'
import ExclusiveOffers from '../components/ExclusiveOffers'
import Testimonial from '../components/Testimonial '
import NewsLetter from '../components/NewsLetter'
import RecommendedHotels from '../components/RecommendedHotels'

const Home = () => {

    return (
        <>
            <Hero />
            <RecommendedHotels />
            <FeaturedDestination />
            <ExclusiveOffers />
            <Testimonial />
            <NewsLetter/>
        </>
    )
}

export default Home
```

Додаток З. Лістинг файлу «MyBookings.jsx»

```
import React, { useEffect, useState } from 'react'
import Title from '../components/Title'
import { assets } from '../assets/assets'
import { useAppContext } from '../context/AppContext'
import toast from 'react-hot-toast'

const MyBookings = () => {

    const { axios, getToken, user } = useAppContext();
    const [bookings, setBookings] = useState([]);

    const fetchUserBookings = async () => {
        try {
                const { data } = await axios.get('/api/bookings/user', {
headers: { Authorization: `Bearer ${await getToken()}` } })
            if (data.success) {
                setBookings(data.bookings)
            }
            else {
                toast.error(data.message)
            }
        } catch (error) {
            toast.error(error.message)
        }
    }

    const handlePayment = async (bookingId) => {
        try {
                const { data } = await axios.post('/api/bookings/stripe-
payment', { bookingId }, { headers: { Authorization: `Bearer ${await
getToken()}` } })
            if (data.success) {
```

```
                window.location.href = data.url
            } else {
                toast.error(data.message)
            }
        } catch (error) {
            toast.error(error.message)
        }
    }

    useEffect(() => {
        if (user) {
            fetchUserBookings();
        }
    }, [user]);

    return (
          <div  className='py-28  md:pb-35  md:pt-32  px-4  md:px-16  lg:px-24
xl:px-32'>
             <Title title='My Bookings' subTitle='Easily manage your past,
current, and upcoming hotel reservations in one place. Plan your trips
seamlessly with just a few clicks' align='left' />
            <div className="max-w-6xl mt-8 w-full text-gray-800">
                 <div  className="hidden  md:grid  md:grid-cols-[3fr_2fr_1fr]
w-full border-b border-gray-300 font-medium text-base py-3">
                    <div className="w-1/3">Hotels</div>
                    <div className="w-1/3">Date & Timings</div>
                    <div className="w-1/3">Payment</div>
                </div>

                {bookings.map((booking) => (
                        <div  key={booking._id}  className="grid  grid-cols-1
md:grid-cols-[3fr_2fr_1fr]    w-full    border-b    border-gray-300    py-6
first:border-t">
                        <div className="flex flex-col md:flex-row">
```

```
                                <img className="min-md:w-44 rounded shadow
object-cover" src={booking.room.images[0]} alt="hotel-img" />
                                <div className="flex flex-col gap-1.5 max-
md:mt-3 min-md:ml-4">
                                <p className="font-playfair text-2xl">
                                    {booking.hotel.name}
                                      <span className="font-inter text-sm">
({booking.room.roomType})</span>
                                </p>
                                   <div className="flex items-center gap-1
text-sm text-gray-500">
                                            <img  src={assets.locationIcon}
alt="location-icon" />
                                    <span>{booking.hotel.address}</span>
                                </div>
                                   <div className="flex items-center gap-1
text-sm text-gray-500">
                                              <img  src={assets.guestsIcon}
alt="guests-icon" />
                                    <span>Guests: {booking.guests}</span>
                                </div>
                                          <p  className="text-base">Total:
${booking.totalPrice}</p>
                            </div>
                        </div>

                         <div  className="flex flex-row md:items-center
md:gap-12 mt-3 gap-8">
                            <div>
                                <p>Check-In:</p>
                                  <p className="text-gray-500 text-sm">{new
Date(booking.checkInDate).toDateString()}</p>
                            </div>
                            <div>
```

```
                                <p>Check-Out:</p>
                                  <p className="text-gray-500 text-sm">{new
Date(booking.checkOutDate).toDateString()}</p>
                            </div>
                        </div>

                         <div className="flex flex-col items-start justify-
center pt-3">
                            <div className="flex items-center gap-2">
                                    <div className={`h-3 w-3 rounded-full
${booking.isPaid ? "bg-green-500" : "bg-red-500"}`}></div>
                                <p className={`text-sm ${booking.isPaid ?
"text-green-500" : "text-red-500"}`}>
                                   {booking.isPaid ? "Paid" : "Unpaid"}
                                </p>
                            </div>
                            {!booking.isPaid && (
                                                <button  onClick={()=>
handlePayment(booking._id)}  className="px-4  py-1.5  mt-4  text-xs  border
border-gray-400  rounded-full  hover:bg-gray-50  transition-all  cursor-
pointer">
                                    Pay Now
                                </button>
                            )}
                        </div>
                    </div>
                ))}
            </div>
        </div>
    )
}

export default MyBookings
```

Додаток И. Лістинг файлу «RoomDetails.jsx»

```
import React, { useEffect, useState } from 'react'
import { assets, roomCommonData } from '../assets/assets'
import { useAppContext } from '../context/AppContext';
import { useParams } from 'react-router-dom';
import StarRating from '../components/StarRating';
import toast from 'react-hot-toast';

const RoomDetails = () => {
    const { id } = useParams();
      const { facilityIcons, rooms, getToken, axios, navigate } =
useAppContext();

    const [room, setRoom] = useState(null);
    const [mainImage, setMainImage] = useState(null);
    const [checkInDate, setCheckInDate] = useState(null);
    const [checkOutDate, setCheckOutDate] = useState(null);
    const [guests, setGuests] = useState(1);

    const [isAvailable, setIsAvailable] = useState(false);

    const checkAvailability = async () => {
        try {

            if (checkInDate >= checkOutDate) {
                  toast.error('Check-In Date should be less than Check-Out
Date')
                return;
            }

                const { data } = await axios.post('/api/bookings/check-
availability', { room: id, checkInDate, checkOutDate })
            if (data.success) {
```

```
                if (data.isAvailable) {
                    setIsAvailable(true)
                    toast.success('Room is available')
                } else {
                    setIsAvailable(false)
                    toast.error('Room is not available')
                }
            } else {
                toast.error(data.message)
            }
        } catch (error) {
            toast.error(error.message)
        }
    }

    const onSubmitHandler = async (e) => {
        try {
            e.preventDefault();
            if (!isAvailable) {
                return checkAvailability();
            } else {
                 const { data } = await axios.post('/api/bookings/book', {
room: id, checkInDate, checkOutDate, guests, paymentMethod: "Pay At Hotel"
}, { headers: { Authorization: `Bearer ${await getToken()}` } })
                if (data.success) {
                    toast.success(data.message)
                    navigate('/my-bookings')
                    scrollTo(0, 0)
                } else {
                    toast.error(data.message)
                }
            }
        } catch (error) {
            toast.error(error.message)
```

```
        }
    }

    useEffect(() => {
        const room = rooms.find(room => room._id === id);
        room && setRoom(room);
        room && setMainImage(room.images[0]);
    }, [rooms]);

    return room && (
        <div className='py-28 md:py-35 px-4 md:px-16 lg:px-24 xl:px-32'>

            {/* Room Details */}
            <div className='flex flex-col md:flex-row items-start md:items-
center gap-2'>
                                <h1   className='text-3xl   md:text-4xl   font-
playfair'>{room.hotel.name}      <span      className='font-inter      text-
sm'>({room.roomType})</span></h1>
                <p className='text-xs font-inter py-1.5 px-3 text-white bg-
orange-500 rounded-full'>20% OFF</p>
            </div>
            <div className='flex items-center gap-1 mt-2'>
                <StarRating />
                <p className='ml-2'>200+ reviews</p>
            </div>
            <div className='flex items-center gap-1 text-gray-500 mt-2'>
                <img src={assets.locationIcon} alt='location-icon' />
                <span>{room.hotel.address}</span>
            </div>

            {/* Room Images */}
            <div className='flex flex-col lg:flex-row mt-6 gap-6'>
                <div className='lg:w-1/2 w-full'>
                        <img  className='w-full rounded-xl shadow-lg object-
```

```
cover'
                        src={mainImage} alt='Room Image' />
                </div>

                <div className='grid grid-cols-2 gap-4 lg:w-1/2 w-full'>
                       {room?.images.length > 1 && room.images.map((image,
index) => (
                                          <img  key={index}  onClick={()  =>
setMainImage(image)}
                              className={`w-full rounded-xl shadow-md object-
cover cursor-pointer ${mainImage === image && 'outline-3 outline-orange-
500'}`} src={image} alt='Room Image' />
                    ))}
                </div>
            </div>

            {/* Room Highlights */}
              <div className='flex flex-col md:flex-row md:justify-between
mt-10'>
                <div className='flex flex-col'>
                             <h1  className='text-3xl  md:text-4xl  font-
playfair'>Experience Luxury Like Never Before</h1>
                     <div className='flex flex-wrap items-center mt-3 mb-6
gap-4'>
                        {room.amenities.map((item, index) => (
                              <div key={index} className='flex items-center
gap-2 px-3 py-2 rounded-lg bg-gray-100'>
                                   <img src={facilityIcons[item]} alt={item}
className='w-5 h-5' />
                                <p className='text-xs'>{item}</p>
                            </div>
                        ))}
                    </div>
                </div>
```

```
                {/* Room Price */}
                                        <p    className='text-2xl    font-
medium'>${room.pricePerNight}/night</p>
            </div>

            {/* CheckIn CheckOut Form */}
                <form  onSubmit={onSubmitHandler}  className='flex  flex-col
md:flex-row  items-start  md:items-center  justify-between  bg-white  shadow-
[0px_0px_20px_rgba(0,0,0,0.15)] p-6 rounded-xl mx-auto mt-16 max-w-6xl'>
                 <div className='flex flex-col flex-wrap md:flex-row items-
start md:items-center gap-4 md:gap-10 text-gray-500'>
                    <div className='flex flex-col'>
                            <label  htmlFor='checkInDate'  className='font-
medium'>Check-In</label>
                                                <input   onChange={(e)   =>
setCheckInDate(e.target.value)}   id='checkInDate'   type='date'   min={new
Date().toISOString().split('T')[0]}   className='w-full   rounded   border
border-gray-300  px-3  py-2  mt-1.5  outline-none'  placeholder='Check-In'
required />
                    </div>
                           <div  className='w-px  h-15  bg-gray-300/70  max-
md:hidden'></div>
                    <div className='flex flex-col'>
                            <label  htmlFor='checkOutDate'  className='font-
medium'>Check-Out</label>
                                                <input   onChange={(e)   =>
setCheckOutDate(e.target.value)}       id='checkOutDate'       type='date'
min={checkInDate} disabled={!checkInDate} className='w-full rounded border
border-gray-300  px-3  py-2  mt-1.5  outline-none'  placeholder='Check-Out'
required />
                    </div>
                           <div  className='w-px  h-15  bg-gray-300/70  max-
md:hidden'></div>
                    <div className='flex flex-col'>
```

```
                                <label htmlFor='guests' className='font-
medium'>Guests</label>
                            <input onChange={(e) => setGuests(e.target.value)}
value={guests} id='guests' type='number' className='max-w-20 rounded border
border-gray-300 px-3 py-2 mt-1.5 outline-none' placeholder='0' required />
                        </div>
                    </div>
                        <button type='submit' className='bg-primary hover:bg-
primary-dull active:scale-95 transition-all text-white rounded-md max-md:w-
full max-md:mt-6 md:px-25 py-3 md:py-4 text-base cursor-
pointer'>{isAvailable ? "Book Now" : "Check Availability"}</button>
                </form>

                {/* Common Specifications */}
                <div className='mt-25 space-y-4'>
                    {roomCommonData.map((spec, index) => (
                        <div key={index} className='flex items-start gap-2'>
                                    <img  className='w-6.5'  src={spec.icon}
alt={`${spec.title}-icon`} />
                            <div>
                                <p className='text-base'>{spec.title}</p>
                                            <p  className='text-gray-
500'>{spec.description}</p>
                            </div>
                        </div>
                    ))}
                </div>

                <div className='max-w-3xl border-y border-gray-300 my-15 py-10
text-gray-500'>
                    <p>Guests will be allocated on the ground floor according
to availability. You get a comfortable Two bedroom apartment has a true
city feeling. The price quoted is for two guest, at the guest slot please
mark the number of guests to get the exact price for groups. The Guests
```

```
will be allocated ground floor according to availability. You get the
comfortable two bedroom apartment that has a true city feeling.</p>
            </div>

            <div className='flex flex-col items-start gap-4'>
                <div className='flex gap-4'>
                    <img className='h-14 w-14 md:h-18 md:w-18 rounded-full'
src={room.hotel.owner.image} alt='Host' />
                    <div>
                              <p  className='text-lg  md:text-xl'>Hosted  by
{room.hotel.name}</p>
                        <div className='flex items-center mt-1'>
                            <StarRating />
                            <p className='ml-2'>200+ reviews</p>
                        </div>
                    </div>
                </div>
                 <button className='px-6 py-2.5 mt-4 rounded text-white bg-
primary hover:bg-primary-dull transition-all cursor-pointer'>
                    Contact Now
                </button>
            </div>
        </div>
    )
}

export default RoomDetails
```

Додаток І. Лістинг файлу «bookingController.js»

```
import transporter from "../configs/nodemailer.js";
import Booking from "../models/Booking.js";
import Hotel from "../models/Hotel.js";
import Room from "../models/Room.js";
import stripe from "stripe";

const checkAvailability = async ({ checkInDate, checkOutDate, room }) => {

  try {
    const bookings = await Booking.find({
      room,
      checkInDate: { $lte: checkOutDate },
      checkOutDate: { $gte: checkInDate },
    });

    const isAvailable = bookings.length === 0;
    return isAvailable;

  } catch (error) {
    console.error(error.message);
  }
};

export const checkAvailabilityAPI = async (req, res) => {
  try {
    const { room, checkInDate, checkOutDate } = req.body;
        const isAvailable = await checkAvailability({ checkInDate,
checkOutDate, room });
    res.json({ success: true, isAvailable });
  } catch (error) {
    res.json({ success: false, message: error.message });
  }
```

```
};

export const createBooking = async (req, res) => {
  try {

    const { room, checkInDate, checkOutDate, guests } = req.body;

    const user = req.user._id;

    const isAvailable = await checkAvailability({
      checkInDate,
      checkOutDate,
      room,
    });

    if (!isAvailable) {
       return res.json({ success: false, message: "Room is not available"
});
    }

    const roomData = await Room.findById(room).populate("hotel");
    let totalPrice = roomData.pricePerNight;



    const checkIn = new Date(checkInDate);
    const checkOut = new Date(checkOutDate);
    const timeDiff = checkOut.getTime() - checkIn.getTime();
    const nights = Math.ceil(timeDiff / (1000 * 3600 * 24));

    totalPrice *= nights;

    const booking = await Booking.create({
      user,
      room,
```

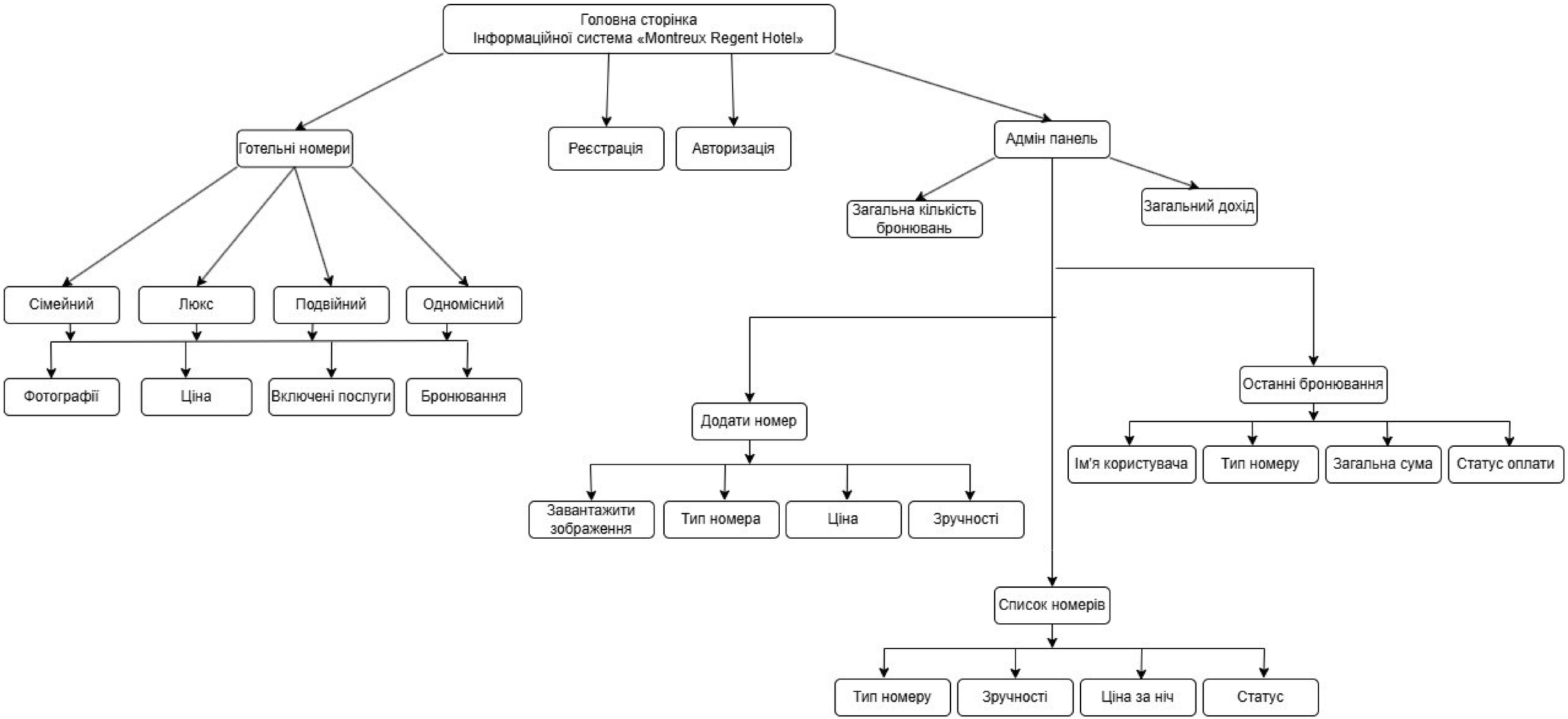
```
      hotel: roomData.hotel._id,
      guests: +guests,
      checkInDate,
      checkOutDate,
      totalPrice,
    });

    const mailOptions = {
      from: process.env.SENDER_EMAIL,
      to: req.user.email,
      subject: 'Hotel Booking Details',
      html: `
        <h2>Your Booking Details</h2>
        <p>Dear ${req.user.username},</p>
        <p>Thank you for your booking! Here are your details:</p>
        <ul>
          <li><strong>Booking ID:</strong> ${booking.id}</li>
          <li><strong>Hotel Name:</strong> ${roomData.hotel.name}</li>
          <li><strong>Location:</strong> ${roomData.hotel.address}</li>
                                                <li><strong>Date:</strong>
${booking.checkInDate.toDateString()}</li>
            <li><strong>Booking Amount:</strong>  ${process.env.CURRENCY ||
'$'} ${booking.totalPrice} /night</li>
        </ul>
        <p>We look forward to welcoming you!</p>
        <p>If you need to make any changes, feel free to contact us.</p>
      `,
    };

    await transporter.sendMail(mailOptions);

    res.json({ success: true, message: "Booking created successfully" });

  } catch (error) {
```

```
    console.log(error);

    res.json({ success: false, message: "Failed to create booking" });
  }
};

export const getUserBookings = async (req, res) => {
  try {
    const user = req.user._id;
       const bookings = await Booking.find({ user }).populate("room
hotel").sort({ createdAt: -1 });
    res.json({ success: true, bookings });
  } catch (error) {
    res.json({ success: false, message: "Failed to fetch bookings" });
  }
};

export const getHotelBookings = async (req, res) => {
  try {
    const hotel = await Hotel.findOne({ owner: req.auth.userId });
    if (!hotel) {
      return res.json({ success: false, message: "No Hotel found" });
    }
        const bookings = await Booking.find({ hotel: hotel._id
}).populate("room hotel user").sort({ createdAt: -1 });

    const totalBookings = bookings.length;

       const totalRevenue = bookings.reduce((acc, booking) => acc +
booking.totalPrice, 0);

    res.json({ success: true, dashboardData: { totalBookings, totalRevenue,
bookings } });
  } catch (error) {
```

```
    res.json({ success: false, message: "Failed to fetch bookings" });
  }
};

export const stripePayment = async (req, res) => {
  try {

    const { bookingId } = req.body;

    const booking = await Booking.findById(bookingId);
    const roomData = await Room.findById(booking.room).populate("hotel");
    const totalPrice = booking.totalPrice;

    const { origin } = req.headers;

    const stripeInstance = new stripe(process.env.STRIPE_SECRET_KEY);

    const line_items = [
      {
        price_data: {
          currency: "usd",
          product_data: {
            name: roomData.hotel.name,
          },
          unit_amount: totalPrice * 100,
        },
        quantity: 1,
      },
    ];

    const session = await stripeInstance.checkout.sessions.create({
      line_items,
      mode: "payment",
      success_url: `${origin}/loader/my-bookings`,
```

```
      cancel_url: `${origin}/my-bookings`,
      metadata: {
        bookingId,
      },
    });
    res.json({ success: true, url: session.url });

  } catch (error) {
    res.json({ success: false, message: "Payment Failed" });
  }
}
```

Схема структурна клієнтської частини інформаційної системи «Montreux Regent Hotel



						2026.КВР.122.423.16.00.00 СС	
Зм.	Арк.	№ докум.	Підпис	Дата	Розробка інформаційної системи управління готельним фондом та бронюванням «Montreux Regent Hotel» Схема структурна клієнтської частини	Літ.	Маса	Масштаб
Розроб.		Процик Б.О.						
Перевір.		Слободян Р.О.				Аркуш	Аркушів 1	
Т.контр.								
Рецензент						ВСП ТФК ТНТУ КН-423		
Н.контр.		Приймак В.А.				м. Тернопіль		
Затв.								

Копіював Формат А1

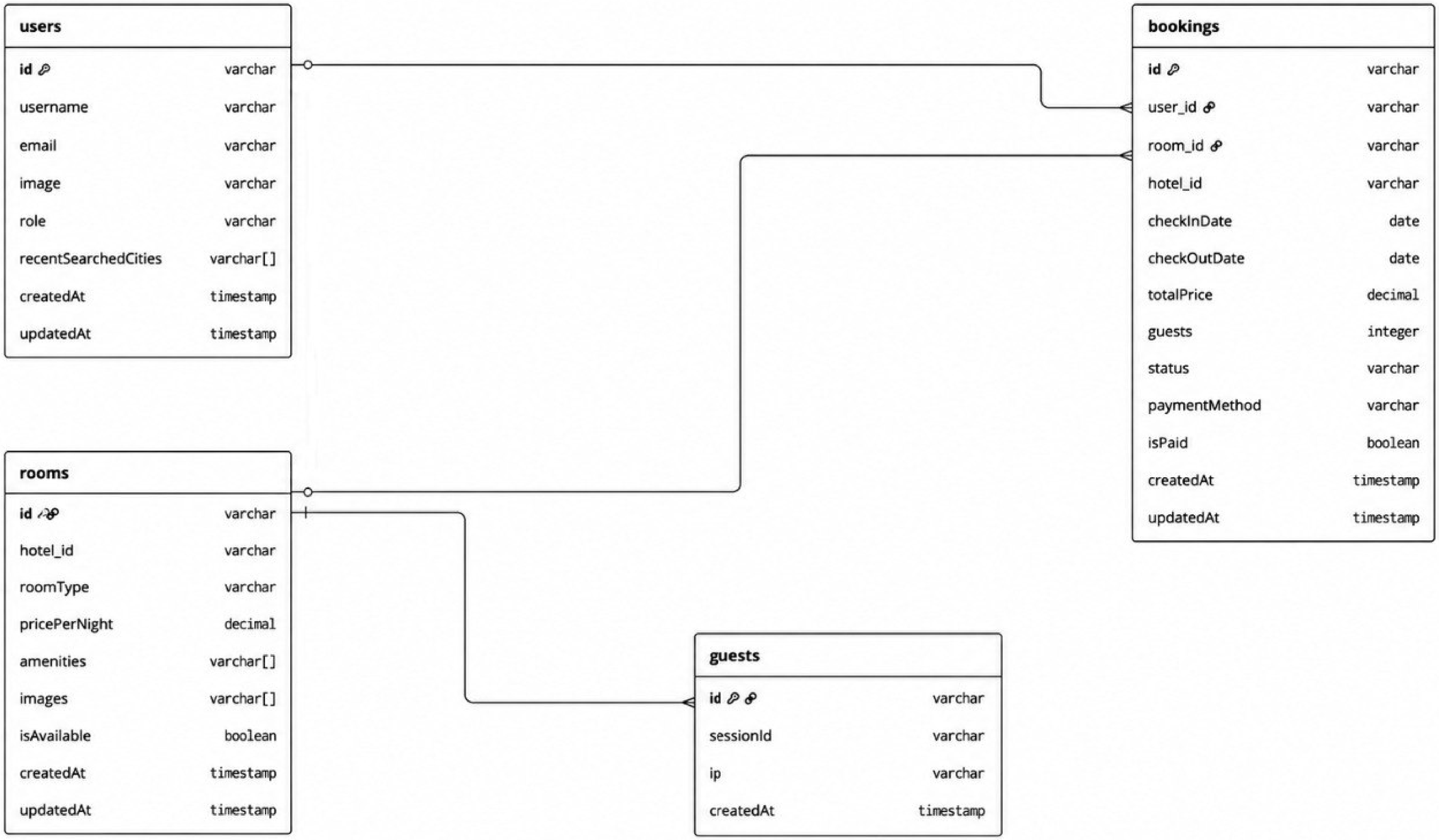
UML-діаграма варіантів використання інформаційної системи «Montreux Regent Hotel

Зм.	Арк.	№ докум.	Підпис	Дата	2026.КВР.122.423.16.00.00 ДВ	
Розроб.		Процик Б.О.			Розробка інформаційної системи управління готельним бізнесом та бронюванням «Montreux Regent Hotel» UML-діаграма варіантів використання	Літ. / Маса / Масштаб
Перевір.		Слободян Р.О.				
Т.контр.						Аркуш / Аркушів 1
Рецензент						ВСП ТФК ТНТУ КН-423 м. Тернопіль
Н.контр.		Приймак В.А.				
Затв.						

Копіював

Формат А1

ER-діаграма бази даних інформаційної системи «Montreux Regent Hotel



Зм.	Арк.	№ докум.	Підпис	Дата	2026.КВР.122.423.16.00.00 БД	
Розроб.		Процик Б.О.			Розробка інформаційної системи управління готельним фондом та бронюванням «Monfreux Regent Hotel» ER-діаграма бази даних	Літ. / Маса / Масштаб
Перевір.		Слободян Р.О.				
Т.контр.						Аркуш / Аркушів 1
Рецензент						ВСП ТФК ТНТУ КН-423 м. Тернопіль
Н.контр.		Прийомак В.А.				
Затв.						

Копіював Формат А1

Таблиця техніко-економічних показників

№ п/п	Показник	Одиниці вимірювання	Значення
1	Мова програмування	–	JavaScript
2	Стек технологій клієнтської частини	–	React JS, React Router, Axios, Tailwind, Vite
3	Стек технологій серверної частини	–	Express JS, Clerk, Brevo
4	Система керування базою даних	–	Mongo DB
5	Менеджер стану	–	Redux, createContext
6	Менеджер пакетів	год	112
7	Собівартість	грн.	71 025
8	Плановий прибуток	грн.	41 748
9	ЧТВ	грн.	112 503
10	Термін окупності	рік	2.2