БАКАЛАВРСЬКА

РОБОТА

БР.ІП – 21.00.00.000 ПЗ

Група ІП-22-1

Питюк Богдан

Івано-Франківський національний технічний університет нафти і газу
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

Питюк Богдан Ігорович

(прізвище, ім'я, по батькові)

УДК 004.42

БАКАЛАВРСЬКА РОБОТА

Розробка веб-орієнтованої інформаційної системи бронювання номерів для готельного комплексу

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121– Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня Питюк Б.І
(підпис, ініціали та прізвище здобувача)

Науковий керівник к.т.н. доцент. Бандура Вікторія Валеріївна
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

доц. БандураВ.В
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2026

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. Кафедрою ***ІП***

Доц. В.В. Бандура

“____” _____2026 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ
Питюк Богдан Ігорович

(прізвище, ім’я, по батькові)

1. Тема проекту (роботи) Розробка веб-орієнтованої інформаційної системи бронювання номерів для готельного комплексу

керівник проекту (роботи) Бандура Вікторія Валеріївна д.т.н., доцент

2. Строк подання студентом проекту (роботи) 10 червня 2026р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

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

1. Аналіз теорії конструювання програмного забезпечення
2. Проєкування бази даних системи
3. Опис бізнес-логіки
4. Представлення інтерфейсу програмного продукту
5. Тестування програмного продукту

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

1. Структурна схема взаємодії компонентів веб-системи (рис. 1.6, ст. 25)
2. Структурна схема Чистої архітектури вебсистеми (рис. 2.1, ст. 27)
3. UML-діаграма прецедентів системи бронювання (рис. 2.2, ст. 30)
4. UML-діаграма послідовностей (рис.2.3, ст 33)
5. Логічна ER-діаграма бази даних (рис. 2.4, ст. 46)

1. ***Консультанти розділів проекту (роботи)***

Розділ	***Консультант***	***Підпис, дата***	
		Завдання видав	***Завдання прийняв***

2. ***Дата видачі завдання*** _____

Керівник _____ _____
(підпис) (розшифровка підпису)

Завдання прийняв до виконання _____ _____
(підпис) (розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	***Назва етапів дипломного проекту (роботи)***	**Строк виконання етапів проекту**	**Примітка**
1	Визначення та обґрунтування теми роботи	15.03.2026	Виконано
2	Огляд існуючих концепцій, рішень та сервісів в даній області	05.04.2026	Виконано
3	Побудова моделі або алгоритму власного рішення	25.04.2026	Виконано
4	Документування реалізації власного оригінального рішення вибраними засобами	15.05.2026	Виконано
5	Оформлення пояснювальної записки бакалаврської роботи	10.06.2026	Виконано

Студент – дипломник _____ _____
(підпис) (розшифровка підпису)

Керівник роботи _____ _____
(підпис) (розшифровка підпису)

АНОТАЦІЯ

Бакалаврська робота містить 100 сторінок рисунків 23, список використаних джерел найменувань, 4 додатки.

Метою роботи є розробка інформаційної системи для бронювання номерів готелю, яка спрямована на автоматизацію процесів взаємодії між адміністрацією готелю та клієнтами, забезпечення зручності вибору номерів та оптимізацію процесу управління готельним фондом за допомогою сучасних веб-технологій.

Об'єкт дослідження: процеси проектування та функціонування інформаційної веб-системи для готельного комплексу, спрямованої на автоматизацію операційного управління та взаємодію з клієнтами.

Результати дослідження: Створено інформаційну веб-систему з використанням React та ASP.NET Core, яка дозволяє автоматизувати бронювання готельних номерів, забезпечуючи високу продуктивність та захист даних.

В першому розділі аналізуються сучасні підходи до автоматизації готельного бізнесу, обґрунтовується вибір технологічного стеку для розробки та аналізуються існуючі аналоги систем бронювання.

В другому розділі проведено детальний аналіз вимог до системи, визначено межі функціональності, побудовано концептуальну модель даних та описано архітектуру майбутнього застосунку.

В третьому розділі описано проектування інтерфейсу користувача, реалізацію бізнес-логіки системи та процес забезпечення якості шляхом комплексного тестування.

Висновки: результати розробки, підтверджено відповідність системи технічному завданню та визначено перспективи подальшого масштабування функціоналу для задоволення потреб готельного господарства.

КЛЮЧОВІ СЛОВА: ІНФОРМАЦІЙНА СИСТЕМА, БРОНЮВАННЯ ГОТЕЛЮ, REACT, ASP.NET CORE, ВЕБ-ДОДАТОК, БАЗА ДАНИХ, АВТОМАТИЗАЦІЯ, ТЕСТУВАННЯ, КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС, БЕЗПЕКА ДАНИХ.

ABSTRACT

The bachelor's thesis contains 100 pages, 23 figures, a list of references consisting of sources, and 2 appendices.

The aim of the work is to develop an information system for hotel room booking, which is intended to automate the interaction processes between hotel administration and clients, ensure convenience in room selection, and optimize hotel fund management using modern web technologies.

Object of study: the design and operation processes of an information web system for a hotel complex, aimed at automating operational management and customer interaction.

Research results: An information web system has been created using React and ASP.NET Core, which allows for the automation of hotel room bookings, ensuring high performance and data security.

The first chapter analyzes modern approaches to automating the hotel business, justifies the choice of the technology stack for development, and evaluates existing booking system analogs.

The second chapter provides a detailed analysis of system requirements, defines the scope of functionality, constructs a conceptual data model, and describes the system's architecture.

The third chapter describes the design of the user interface, the implementation of the system's business logic, and the quality assurance process through comprehensive testing.

The conclusions summarize the development results, confirm the system's compliance with the technical specifications, and identify prospects for further functional scaling to meet the needs of the hotel industry.

KEYWORDS: INFORMATION SYSTEM, HOTEL RESERVATION, REACT, ASP.NET CORE, WEB APPLICATION, DATABASE, AUTOMATION, TESTING, USER INTERFACE, DATA SECURITY.

ЗМІСТ

ВСТУП **8**

РОЗДІЛ 1. АНАЛІЗ ПРЕМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ.10

1.1. Дослідження ринку систем онлайн-бронювання. 10

1.2 Аналіз існуючих програмних рішень та архітектурних підходів 12

1.3 Формування вимог та постановка задачі на розробку 19

1.4 Висновки по розділу 25

РОЗДІЛ 2. ПРОЕКТУВАННЯ АРХІТЕКТУРИ ВЕБ-СИСТЕМИ 27

2.1. Концептуальне проєктування та розробка структурної схеми вебсистеми 27

2.2. Аналіз та обґрунтування технологічного стеку 33

2.3. Архітектура клієнт-серверної взаємодії. 37

2.4. Проєктування логічної та фізичної моделі бази даних 39

2.5. Стратегія ешелонованого захисту та управління доступом 46

2.6 Висновки по розділу 50

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ 52

3.1. Розробка серверної частини (Backend). 52

3.2. Програмне конструювання клієнтського середовища (Frontend) 55

3.3. Тестування програмного забезпечення та верифікація системи 66

3.4 Висновок по розлілу 70

ВИСНОВОК **72**

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ **74**

ДОДАТОК

БІБЛІОГРАФІЧНА ДОВІДКА

ВСТУП

У сучасному світі інформаційні технології стають невід'ємною частиною всіх сфер людської діяльності, і готельний бізнес не є винятком. Стрімкий розвиток цифрової економіки, зміна споживчих вподобань та зростання вимог до швидкості й якості обслуговування змушують представників індустрії гостинності шукати нові шляхи taco toot oht оптимізації операційної діяльності. В умовах високої конкуренції та динамічності ринку, автоматизація процесів бронювання та управління готельним фондом стає не просто перевагою, а необхідною умовою виживання та розвитку. Це підкреслює потребу у створенні ефективних програмних інструментів, які б забезпечили надійну, швидку та зручну взаємодію між готелем та його клієнтами.

Актуальність теми зумовлена необхідністю модернізації процесів бронювання готельних номерів шляхом впровадження спеціалізованої веб-системи. Незважаючи на існування глобальних агрегаторів (таких як Booking.com, Expedia або Hotels.com), багато готелів потребують власних, гнучких рішень, які дозволяли б уникнути високих комісій, зберігати контроль над даними клієнтів та надавати персоналізовані сервіси безпосередньо через офіційний веб-ресурс. В умовах сучасної економічної нестабільності, можливість прямої взаємодії з гостем, забезпечення безперебійної роботи системи в режимі 24/7 та оптимізація завантаженості номерного фонду стають критично важливими факторами ефективності бізнесу.

Метою роботи є розробка інформаційної системи для бронювання номерів готелю, яка сприятиме підвищенню ефективності управління готельним фондом та забезпечить зручний інтерфейс для клієнтів через впровадження сучасних веб-технологій.

Завданнями дослідження є вивчення сучасних тенденцій в автоматизації готельного бізнесу, визначення ключових потреб як адміністрації готелю, так і клієнтів, виявлення недоліків існуючих рішень, проектування та розробка

програмного комплексу, а також проведення тестування системи для забезпечення її надійності та безпеки.

Об’єктом дослідження є процеси розробки програмного забезпечення для автоматизації діяльності готелю, що включають аналіз вимог, проектування архітектури, реалізацію бізнес-логіки та забезпечення безпеки даних користувачів.

Предметом дослідження є методи, технології та інструменти, що застосовуються на етапах життєвого циклу розробки системи бронювання, зокрема: проектування архітектури клієнт-серверного застосунку (React, ASP.NET Core), моделювання структури бази даних, розробка механізмів автентифікації та валідації даних, а також методи тестування системи. В рамках дослідження буде здійснено аналіз особливостей управління готельним фондом, визначено вимоги до інтерфейсу користувача та реалізовано механізми динамічного оновлення статусу номерів, що дозволить забезпечити інноваційний підхід до організації сервісу.

Методи дослідження: системний аналіз процесів бронювання, моделювання об'єктів даних, функціональне та інтеграційне тестування програмного забезпечення, методи верифікації безпеки вхідних даних та аналіз швидкодії серверних запитів.

Розроблена система передбачає можливість перегляду актуальної інформації про номери, оформлення онлайн-бронювання з вибором дат, введення та валідацію персональних даних клієнта, а також адміністрування системи для керівництва готелю. Очікується, що впровадження розробленого рішення дозволить готелю автоматизувати рутинні завдання персоналу, зменшити кількість помилок при реєстрації гостей та підвищити рівень довіри клієнтів завдяки зручному сервісу.

РОЗДІЛ 1. АНАЛІЗ ПРЕМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Дослідження ринку систем онлайн-бронювання.

Сучасний вектор модернізації світового сектору гостинності безпосередньо пов'язаний із глобальною трансформацією бізнес-моделей та перенесенням операційної діяльності у цифрове середовище. Удосконалення технологічних процесів у готельному бізнесі за умов високої конкурентної активності вимагає обов'язкового застосування інноваційних інформаційних комплексів. Головним чинником забезпечення комерційної стійкості готелю стає формування безперервного та максимально доступного каналу взаємодії з потенційною аудиторією, що безпосередньо реалізується через розгортання систем електронного резервування послуг [1].

Сучасна архітектура цифрового ринку автоматизації готельного сегмента розподіляється на два ключові структурні напрями:

1. Глобальні дистриб'юторські мережі та спеціалізовані інтернет-агрегатори, які виконують роль інформаційних посередників між готельним підприємством та кінцевим споживачем послуг.

2. Автономні або хмарні платформи управління майном готелю, що містять інтегровані функціональні модулі для організації безпосереднього продажу номерного фонду через офіційні веб-ресурси компанії.

Дослідження галузевих тенденцій демонструє, що тривале та монопольне використання виключно зовнішніх агенцій або жорстких комерційних платформ призводить до виникнення критичної проблеми, відомої як ефект прив'язки до постачальника (Vendor Lock-in). Готельні комплекси стають цілковито залежними від закритих алгоритмів та фінансової політики сторонніх брендів. Головним технологічним бар'єром при цьому стає неможливість гнучкої інтеграції кастомних програмних інтерфейсів (API), що унеможливлює побудову унікальної цифрової інфраструктури. Це визначає високу наукову та практичну актуальність створення та впровадження індивідуальних веб-орієнтованих

інформаційних рішень для прямої комунікації, які дозволяють повністю контролювати потік клієнтських даних, кастомізувати сервіси під потреби гостей та залишати фінансові потоки всередині бізнес-структури [2,3].

Для формулювання раціонального переліку функціональних вимог до розроблюваного програмного забезпечення було виконано критичний огляд найбільш поширених ринкових продуктів-аналогів. Серед них детально розглянуто як масштабні посередницькі платформи, так і спеціалізовані системи керування.

Платформа Booking.com займає позицію одного з найбільших світових посередницьких сервісів. Дане рішення надає користувачам розвинені інструменти пошуку, багатокритеріальні фільтри та консолідовану базу відгуків. Проте для окремого готельного об'єкта цей ресурс є жорсткою зовнішньою надбудовою.

Система Mews представляє собою інноваційну хмарну платформу управління готелем, яка поєднує внутрішню операційну діяльність із механізмами онлайн-резервування. Вона автоматизує значний масив щоденних процедур, але її використання пов'язане із суттєвим фінансовим навантаженням через абонентську модель оплати. Крім того, її базова конфігурація характеризується певною архітектурною закритістю, що створює бар'єри при необхідності кастомізації інтерфейсу користувача під специфічні корпоративні стандарти, а також викликає труднощі при спробах підключення унікальних локальних платіжних систем або засобів державної фіскалізації.

Проведений детальний огляд ринкової ситуації та архітектурних особливостей існуючого програмного забезпечення дозволяє виділити комплекс спільних недоліків, притаманних готовим комерційним рішенням та зовнішнім агрегаторам:

1. Повна підпорядкованість комерційній та технічній політиці сторонніх розробників, що супроводжується ризиками непередбачуваної зміни тарифних планів або умов обслуговування.

2. Втрата цифрового суверенітету та загрози інформаційній безпеці через необхідність обробки та довгострокового збереження конфіденційних відомостей клієнтів на віддалених серверах інших компаній.

3. Низький рівень адаптивності архітектури, що унеможливлює інтеграцію специфічних внутрішніх кастомних API для взаємодії з унікальним обладнанням готелю, наприклад, системами електронних замків чи локальними ресторанними комплексами.

4. Обмеженість візуальних та функціональних засобів формування унікального клієнтського досвіду (UX), який має повністю відображати фірмову стилістику та маркетингову стратегію конкретного готельного бренду.

З огляду на це, проектування та розробка власної автономної веб-орієнтованої інформаційної системи для бронювання кімнат є повністю обґрунтованим технічним та економічним рішенням. Такий підхід дозволить готельному комплексу нівелювати загрози прив'язки до сторонніх вендорів, суттєво підвищити відсоток прямих інтернет-продажів, забезпечити оперативне керування номерним фондом без часових затримок, а також гарантувати абсолютний рівень захисту комерційної інформації та персональних даних користувачів [4,5].

1.2 Аналіз існуючих програмних рішень та архітектурних підходів

З метою формування концептуальних засад проектування високоефективного веб-орієнтованого інформаційного комплексу для бронювання місць у готелях виникає потреба у проведенні критичного огляду наявних програмних продуктів, представлених на світовому та вітчизняному ринках технологій гостинності. Системне вивчення наявних аналогів дозволяє виокремити позитивні й негативні сторони сучасних інженерних підходів до автоматизації операційного циклу, визначити оптимальний набір користувацьких функцій, обґрунтувати вибір майбутнього стеків технологій та запобігти

виникненню стандартних логічних та інфраструктурних помилок під час розробки власного програмного забезпечення [6].

Наявні на сьогодні ІТ-рішення для цифровізації готельного сектора та забезпечення дистанційного резервування номерного фонду класифікуються за кількома базовими архітектурними категоріями: глобальні розподілені маркетплейси та інформаційні посередники, хмарні екосистеми управління майном із вбудованими клієнтськими інтерфейсами бронювання, а також локальні автономні модулі та форми для інтеграції у структуру сторонніх сайтів. Кожен окремий клас програмного забезпечення характеризується власною топологією баз даних, логікою обробки запитів та ступенем взаємодії між кінцевим споживачем послуг і оператором готелю.

Першим і найбільш поширеним типом систем є глобальні цифрові платформи дистрибуції, типовим зразком яких виступає міжнародний ресурс Booking.com. Цей комплекс функціонує в рамках моделі дворівневого цифрового майданчика, що консолідує відомості про тисячі засобів розміщення та забезпечує взаємодію з мільйонами користувачів. Архітектура Booking.com спирається на потужну географічно розподілену серверну базу, яка здатна виконувати потокову обробку та синхронізацію великих масивів інформації без часових затримок [7].

Для кінцевого споживача програмний інтерфейс цього ресурсу надає розвинені інструменти багатокритеріального пошуку, систему гнучких фільтрів на основі просторових та вартісних показників, а також інтегровану базу оцінювання та верифікації. Відвідувач має змогу детально вивчити параметри кожної кімнати, ознайомитися з медіа-контентом та здійснити транзакцію через захищений платіжний сервіс. Загальний вигляд користувацького інтерфейсу на етапі аналізу категорій та параметрів номерів наведено на рисунку 1.2.

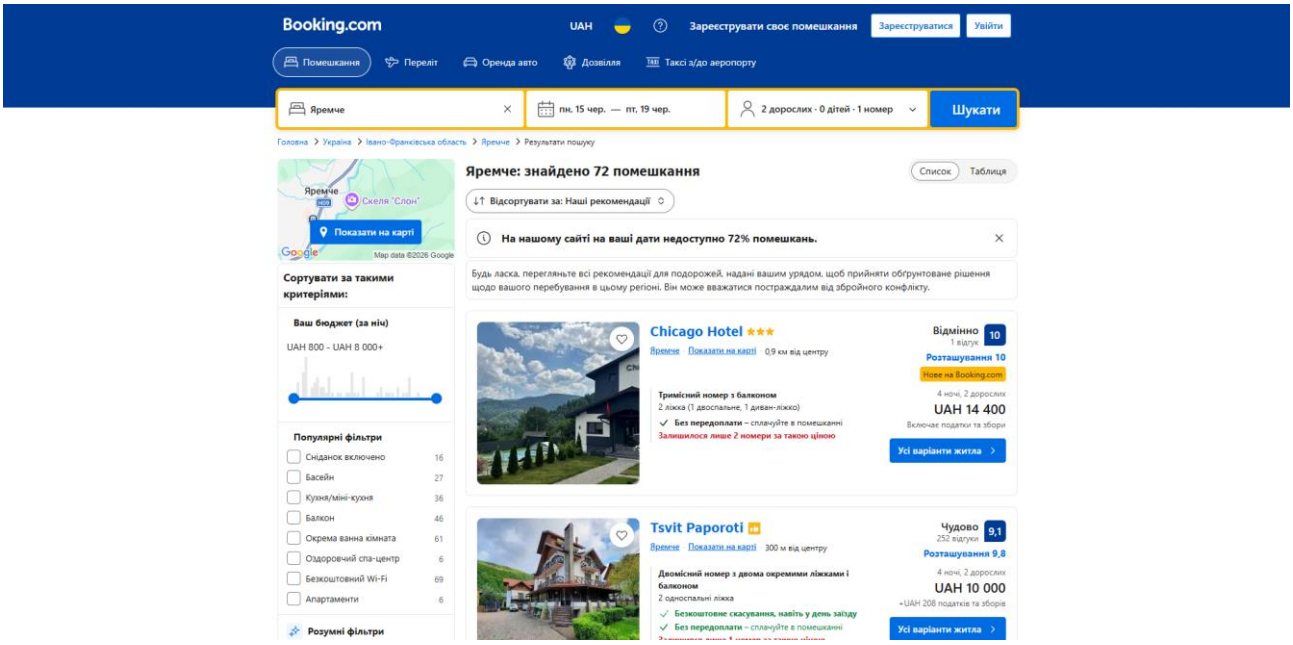

Рисунок 1.2 – Інтерфейс вибору номерів та фільтрації на платформі Booking.com

Попри високий рівень інженерної реалізації, Booking.com накладає суттєві обмеження на операційну діяльність окремого готельного об'єкта. Внутрішній персонал змушений застосовувати ізольовану панель керування для ручного оновлення відомостей про вільні місця та поточні тарифи. Головний архітектурний недолік для готелю полягає в тому, що дана платформа є повністю закритим зовнішнім середовищем. Готельний комплекс позбавлений безпосереднього доступу до бази даних клієнтів, не має можливості адаптувати інтерфейс під власні стандарти дизайну та стикається з жорсткою прив'язкою до сторонніх алгоритмів. До того ж, налаштування обміну даними між внутрішніми системами готелю та платформою Booking.com вимагає розгортання додаткових проміжних модулів, що суттєво ускладнює підсумкову конфігурацію інформаційного середовища підприємства [8].

Другим значущим класом технологічних рішень є інтегровані хмарні платформи автоматизації готельного бізнесу, що постачаються за моделлю програмного забезпечення як послуги. Одним із сучасних представників цього напряму є міжнародна система Mews. На відміну від відкритих інтернет-агрегаторів, Mews створюється насамперед як внутрішній інструмент для

ведення операційного обліку готелю, який додатково комплектується модулями зовнішньої взаємодії, зокрема власним інструментом онлайн-резервування для встановлення на веб-сторінки засобу розміщення.

Основою архітектури Mews є використання хмарних обчислювальних потужностей, що гарантує можливість віддаленого контролю за процесами з будь-якого пристрою через мережу інтернет. Головне робоче середовище адміністратора системи функціонує у формі динамічного календарного графіка, де відображається поточний стан кімнат, заплановані заїзди та сервісні роботи. Приклад реалізації панелі керування адміністратора хмарної системи для моніторингу та розподілу доступного номерного фонду зображено на рисунку 1.3.

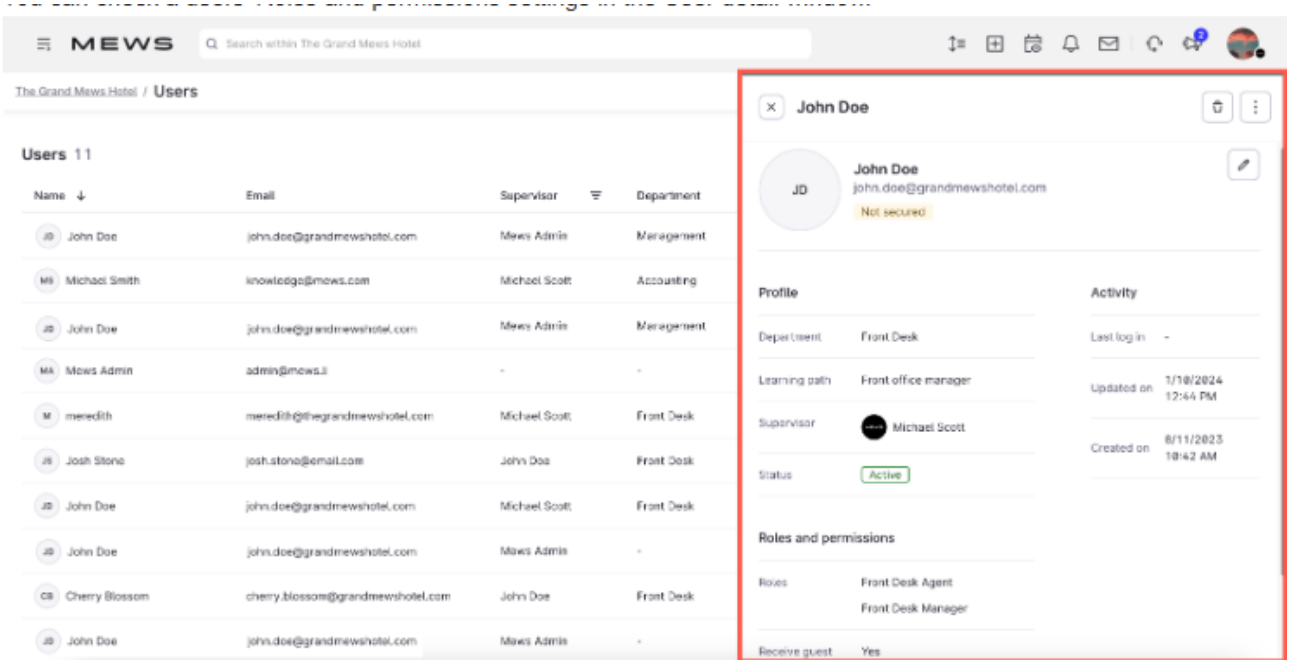

Рисунок 1.3 – Адміністративна панель хмарної системи Mews для управління номерним фондом

Функціональний елемент прямого бронювання від Mews підключається до офіційного сайту готелю за допомогою спеціальних програмних скриптів. Це дозволяє здійснювати продаж номерів без залучення зовнішніх посередників. Проте детальний розбір експлуатації подібних глобальних систем виявляє

наявність серйозних технічних бар'єрів. Інтерфейс клієнтської форми, попри можливість базового коригування стилів, підпорядкований жорстким внутрішнім шаблонам і не дозволяє реалізувати специфічні сценарії взаємодії. Крім того, інтеграція таких систем із локальними державними реєстрами та касовими апаратами часто ускладнюється через специфіку побудови архітектури ядра системи [9].

З метою вивчення можливостей більшої адаптації під потреби конкретного підприємства доцільно розглянути спеціалізовані гнучкі модулі бронювання, які орієнтовані на ширшу інтеграцію. Серед рішень такого типу виділяється платформа Beds24, яка надає інструменти для створення автономних форм замовлень та їхнього гнучкого вбудовування у різноманітні веб-ресурси.

Система Beds24 фокусується на постачанні адаптивних модульних блоків. Користувач може інтегрувати у свій сайт компактний елемент пошуку, який з'єднується безпосередньо з ядром системи та забезпечує високу швидкість передачі даних, підтримуючи при цьому коректне відображення на мобільних пристроях. Приклад візуального представлення віджета онлайн-резервування, інтегрованого у структуру стороннього веб-сайту, показано на рисунку 1.4.

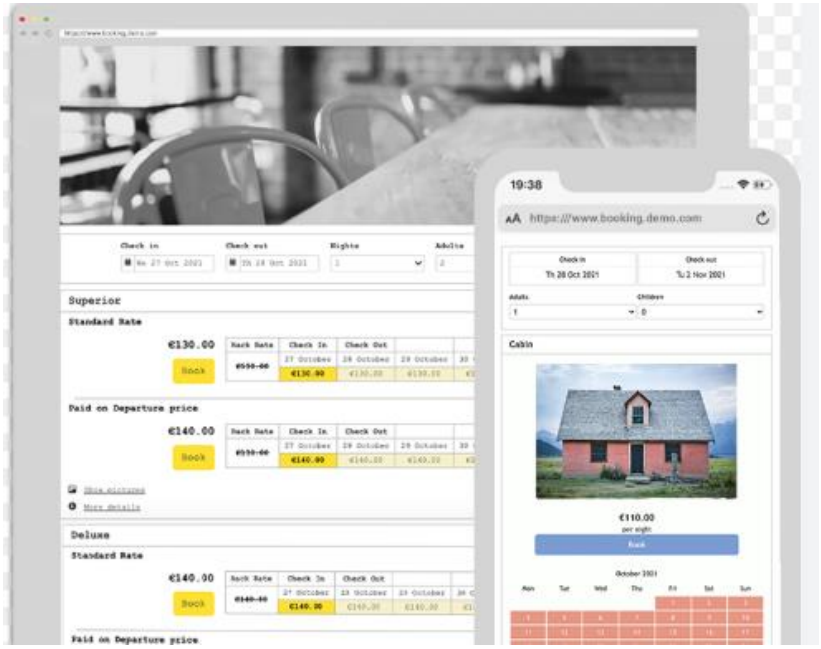

Рисунок 1.4 – Інтерфейс віджета онлайн-бронювання модуля Beds24 на сайті готелю

Основним плюсом застосування подібних модульних компонентів є швидкість їхнього розгортання та наявність базових механізмів обміну даними. Проте цей підхід супроводжується вагомим недоліком: архітектурна побудова форми та послідовність кроків фіксації замовлення обмежені внутрішніми стандартами розробника. У випадку, коли готельний комплекс має потребу в побудові унікального користувацького досвіду із впровадженням складних бізнес-процесів, базові можливості таких систем виявляються недостатніми через закритість вихідного коду для внесення глибоких структурних змін.

Ще одним поширеним варіантом організації онлайн-продажів є використання спеціалізованих регіональних інтернет-платформ, серед яких на вітчизняному просторі відомий сервіс Hotels24. Цей ресурс поєднує можливості локального агрегатора пропозицій та постачальника готових інформаційних сервісів для автоматизації взаємодії. На відміну від світових лідерів, Hotels24 має чітку орієнтацію на внутрішній споживчий сегмент.

Робоче середовище платформи Hotels24 зосереджене на забезпеченні можливості оперативного зіставлення цінових параметрів, деталізації переліку сервісів та спрощення процедури підтвердження наміру заселення. Сторінку відображення результатів пошуку та порівняльного аналізу доступних варіантів проживання в межах сервісу Hotels24 представлено на рисунку 1.5.

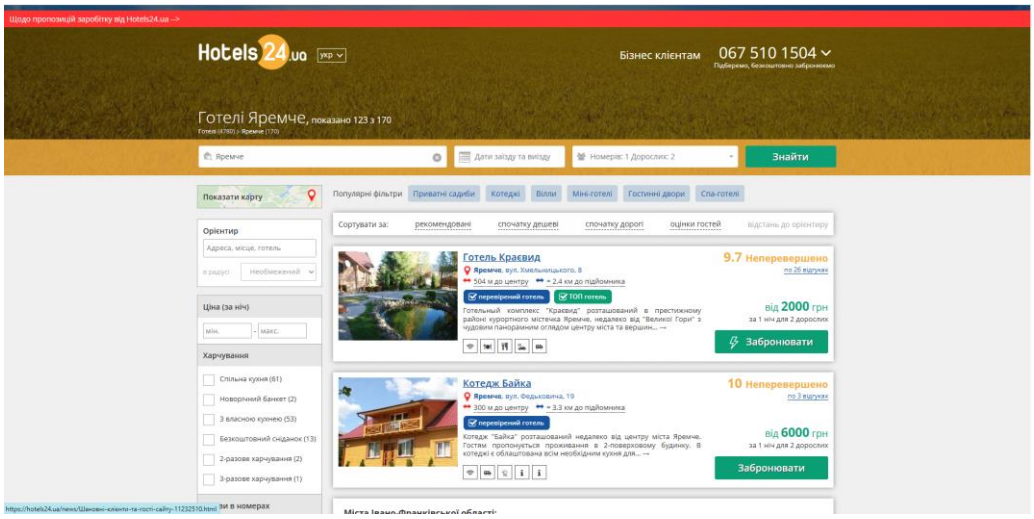

Рисунок 1.5 – Сторінка результатів пошуку готелів на вітчизняному сервісі Hotels24

Аналіз архітектури взаємодії з Hotels24 доводить, що для готельного комплексу робота через подібний інструмент означає постійне перебування в рамках стороннього інформаційного поля. Понад те, процеси синхронізації відомостей про стан номерного фонду під час пікових навантажень можуть виконуватися з певним запізненням, що суттєво підвищує ймовірність створення помилкових подвійних броней на один і той самий час.

Підбиваючи підсумки аналізу наявного програмного забезпечення та інженерних підходів, можна зробити висновок, що жоден із розглянутих базових варіантів не здатний у повній мірі вирішити завдання автономного готельного комплексу, який орієнтується на повну незалежність та унікальність власного цифрового бренду [10].

Застосування зовнішніх посередницьких майданчиків призводить до втрати прямого контакту з цільовою аудиторією та обмежує аналіз поведінкових факторів користувачів. Впровадження великих закордонних хмарних систем супроводжується значними інфраструктурними витратами та складнощами під час адаптації до локальних вимог обліку. Використання готових уніфікованих віджетів (на зразок Beds24) критично обмежує гнучкість проектування, створюючи проблему жорсткої залежності від розробника (Vendor Lock-in) та унеможливлюючи модернізацію веб-ресурсу під індивідуальні функціональні потреби бізнесу через відсутність доступу до кастомних API.

З огляду на це, найбільш раціональним технічним шляхом є створення власної веб-орієнтованої інформаційної системи. Спроектований програмний продукт повинен базуватися на ізольованій архітектурі, мати повністю адаптивний інтерфейс, реалізовувати прямий обмін даними з власною базою без затримок, підтримувати гнучкі інструменти динамічного пошуку та забезпечувати безпечну взаємодію з локальними фінансовими сервісами. Це дозволить готелю оптимізувати внутрішні технологічні процеси та суттєво підвищити якість обслуговування клієнтів [11].

1.3 Формування вимог та постановка задачі на розробку

Постановка задачі є визначальним етапом проектування будь-якої інформаційної системи, оскільки вона формалізує вимоги замовника, окреслює межі проектування та визначає критерії, за якими буде оцінюватися успішність реалізації програмного продукту. На основі проведеного аналізу ринку та дослідження існуючих аналогів, у цьому підпункті детально формулюється технічне завдання на розробку автономної веб-орієнтованої інформаційної системи бронювання номерів для готельного комплексу.

Метою дипломної роботи є проектування, програмна реалізація та впровадження веб-орієнтованої інформаційної системи, яка забезпечує автоматизацію процесів пошуку, фільтрації, бронювання та оплати готельних номерів кінцевими користувачами, а також надає адміністрації готелю інструменти для ефективного керування номерним фондом, тарифами та клієнтською базою в реальному часі [12].

Об'єктом розробки є процес інформаційного супроводу та операційного управління діяльністю готельного комплексу в сегменті взаємодії з клієнтами через інтернет-простір.

Предметом розробки є методи, алгоритми, програмні засоби, архітектурні рішення та бази даних, що використовуються для створення автономного веб-додатка з функціями динамічного онлайн-резервування.

Для досягнення поставленої мети в межах дипломної роботи необхідно вирішити такі науково-технічні та практичні завдання:

1. Розробити концептуальну та логічну моделі бази даних для збереження інформації про категорії номерів, кімнати, статуси завантаженості, цінові тарифи, користувачів та історію замовлень.
2. Спроектувати гнучку клієнт-серверну архітектуру веб-системи, яка забезпечить швидку обробку запитів, високу відмовостійкість та мінімальний час відгуку.

3. Розробити інтуїтивно зрозумілий, адаптивний та сучасний інтерфейс користувача для клієнтської частини системи, оптимізований под мобільні пристрої та персональні комп'ютери.

4. Реалізувати бізнес-логіку серверної частини системи, включаючи алгоритми динамічної фільтрації масивів даних, перевірки доступності номерів на задані дати та запобігання виникненню подвійних бронювань.

5. Створити захищену адміністративну панель для менеджменту готелю з інтерактивною системою відображення зайнятості кімнат та інструментами аналітичної звітності.

6. Інтегрувати систему з зовнішніми сервісами електронних платежів та автоматичними сповіщеннями користувачів.

Вимоги до функціональної структури проектованої системи формуються на основі розподілу прав доступу та сценаріїв взаємодії різних категорій користувачів. У системі має бути реалізовано щонайменше три основні ролі: незареєстрований або зареєстрований гість, адміністратор (менеджер) готелю, та системний адміністратор [13].

Для ролі клієнта (гостя) інформаційна система повинна забезпечувати виконання таких ключових функцій:

— перегляд загальної інформації про готельний комплекс, переліку послуг та контактних даних;

— динамічний пошук доступних номерів за заданими датами заїзду та виїзду, а також кількістю дорослих гостей і дітей;

— багаторівнева фільтрація результатів пошуку за вартістю, категорією комфортності (стандарт, люкс, апартаменти) та наявністю додаткових зручностей (кондиціонер, сніданок, вид на море тощо);

— перегляд детальної картки кожного номера з фотогалереєю, текстовим описом та актуальною ціною;

— реєстрація нового користувача та авторизація в особистому кабінеті;

— оформлення бронювання з введенням персональних даних або автоматичним їх заповненням з профілю;

— вибір способу оплати (повна передоплата, гарантоване бронювання карткою або оплата при заселенні);

— перегляд історії власних бронювань та їх поточних статусів (підтверджено, скасовано, очікує оплати) в особистому кабінеті;

— можливість самостійного скасування бронювання відповідно до правил готелю.

Для ролі адміністратора (менеджера) готелю система повинна надавати розширений набір інструментів керування в межах робочого місця:

— візуальне відображення стану всього номерного фонду у вигляді інтерактивного календаря-шахматки;

— можливість швидкого створення бронювання в ручному режимі (наприклад, для клієнтів, які звернулися телефоном);

— зміна статусів номерів (вільний, заброньований, заселений, на ремонті, прибирання);

— керування тарифними планами, встановлення сезонних цін, знижок та спеціальних пропозицій на певні періоди;

— робота зі списком клієнтів, перегляд їхньої картки та історії взаємодії;

— генерація базових звітів за обраний період (рівень завантаженості готелю, фінансові надходження, популярність окремих категорій номерів);

— зміна статусів бронювань та фіксація платежів, що надійшли поза веб-системою (готівка, банківський переказ).

Для ролі системного адміністратора мають бути доступні функції технічного обслуговування платформи:

— керування обліковими записами співробітників готелю та призначення їм відповідних прав доступу;

— редагування довідників системи (додавання нових категорій номерів, описів зручностей, типів харчування);

— моніторинг системних логів та журналів дій користувачів для виявлення потенційних помилок або спроб несанкціонованого доступу;

— налаштування глобальних параметрів системи, таких як інтеграційні ключі платіжних систем та шаблони електронних листів сповіщень.

Окрім функціональних вимог, до веб-орієнтованої системи висувається комплекс жорстких нефункціональних вимог, які гарантують її надійність, безпеку та зручність використання в умовах реальної експлуатастатичної або динамічної роботи [14].

Вимоги до інтерфейсу та ергономіки передбачають створення адаптивного дизайну за принципом спочатку для мобільних, оскільки більша частина сучасного інтернет-трафіку припадає на смартфони. Інтерфейс повинен коректно відображатися в усіх сучасних веб-браузерах без спотворення елементів керування. Навігація сайтом має бути максимально спрощеною, щоб клієнт міг завершити процес бронювання не більше ніж за три-чотири кроки.

Вимоги до продуктивності та надійності визначають, що час відгуку системи на стандартні пошукові запити користувачів не повинен перевищувати півтори-дві секунди. Система повинна стабільно працювати в режимі цілодобового доступу (24/7) з коефіцієнтом готовності не менше 99.8%. При виникненні критичних помилок на боці сервера користувачу повинно виводитися інформативне повідомлення без порушення цілісності даних у базі даних. Програмна архітектура має передбачати можливість вертикального та горизонтального масштабування у разі різкого збільшення кількості одночасних сесій під час туристичного сезону.

Вимоги до інформаційної безпеки є критично важливими, оскільки система оперує персональними даними громадян та фінансовою інформацією. Увесь мережевий трафік між браузером користувача та сервером повинен шифруватися за допомогою захищеного протоколу HTTPS з використанням сучасних сертифікатів. Паролі користувачів у базі даних мають зберігатися виключно в захешованому вигляді із використанням стійких криптографічних алгоритмів. Система повинна містити механізми захисту від типових веб-вразливостей, таких як SQL-ін'єкції, міжсайтовий скриптинг та підробка міжсайтових запитів. Безпосередня обробка даних банківських карток повинна

бути винесена на сторону сертифікованого платіжного провайдера, що звільняє готель від необхідності проходження складної сертифікації безпеки.

Для успішного проектування та функціонування системи необхідно також чітко визначити структуру вхідних та вихідних даних. Схематичне представлення інформаційних потоків та архітектури баз даних буде детально розписано у наступних розділах, проте загальний перелік даних формується вже на етапі постановки задачі.

Вхідними даними для системи є:

— інформація про параметри пошуку від користувача (дати, кількість місць);

— персональні дані клієнта при реєстрації або бронюванні (ім'я, прізвище, контактний телефон, адреса електронної пошти);

— текстовий та графічний контент, що завантажується адміністратором (описи номерів, тарифи, фотографії);

— технічні відповіді від сторонніх сервісів, зокрема верифікаційні токени від платіжного шлюзу про успішність транзакції.

— Вихідними даними системи є:

— динамічно сформовані веб-сторінки з результатами пошуку номерів та актуальними цінами;

— підтвердження бронювання у вигляді унікального цифрового коду, що виводиться на екран та дублюється на електронну пошту клієнта;

— зміна станів об'єктів у базі даних (оновлений календар зайнятості кімнат на шахматці адміністратора);

— аналітичні графіки, таблиці та текстові звіти для керівництва готельного комплексу, згенеровані адміністративною панеллю.

Таким чином, сформована постановка задачі чітко окреслює межі та вимоги до розробки веб-орієнтованої інформаційної системи бронювання номерів. Реалізація зазначеного функціоналу дозволить створити сучасний, гнучкий та безпечний програмний продукт, який повністю вирішить проблему

залежності готелю від сторонніх платформ-агрегаторів, оптимізує внутрішні бізнес-процеси підприємства та забезпечить користувачам високий уровень сервісу під час онлайн-взаємодії. Сформульовані вимоги стануть основою для подальшого вибору технологічного стеку розробки, проектування структури бази даних та написання програмного коду системи [15].

Для наочного представлення взаємодії між розроблюваними модулями, різними категоріями користувачів та зовнішніми сервісами було сформовано загальну структурну схему інформаційної системи. Вона відображає архітектурну логіку розподілу інформаційних потоків та взаємозв'язок між клієнтською частиною, веб-сервером та базою даних (рисунок 1.6).

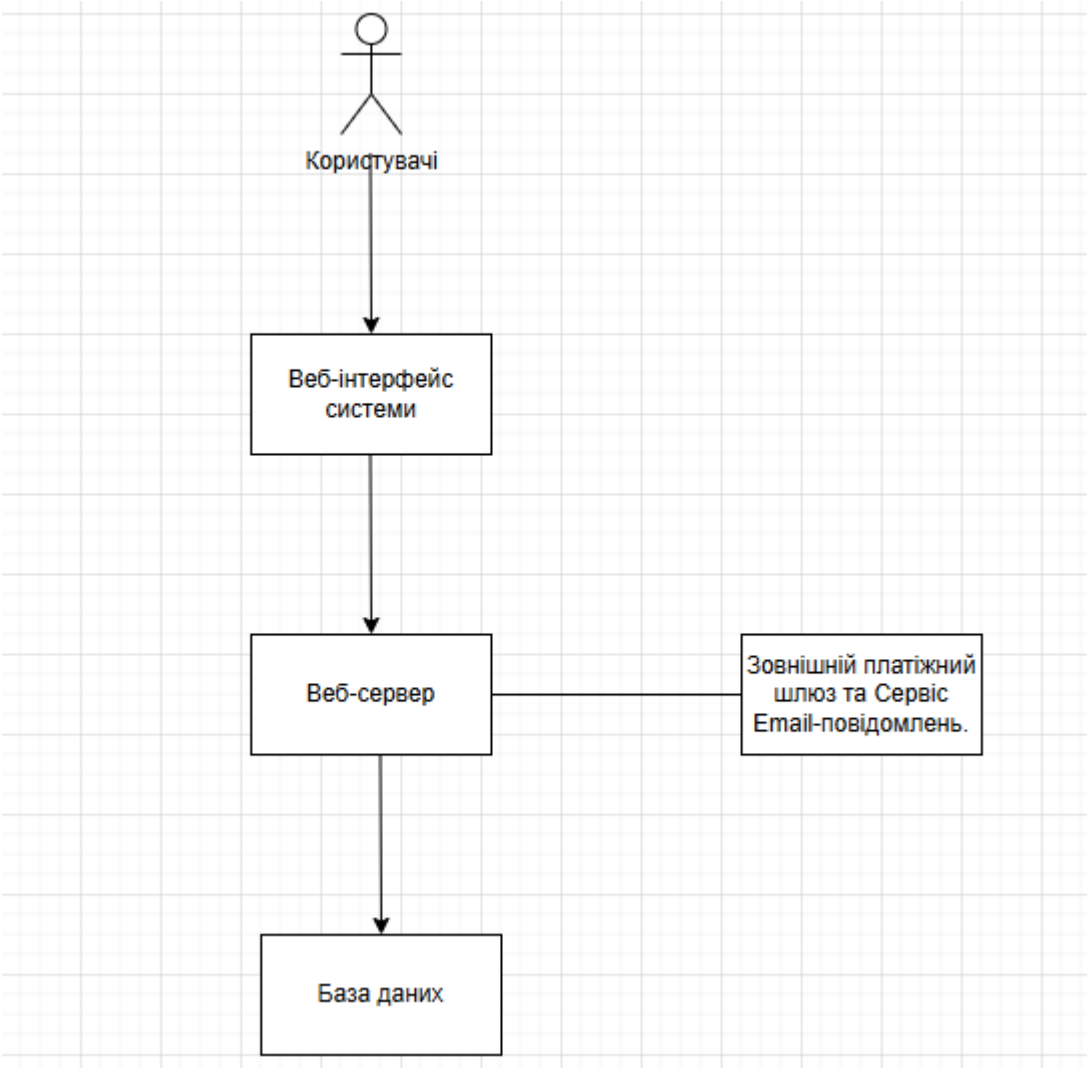

Рисунок 1.6 – Структурна схема взаємодії компонентів веб-системи

1.4 Висновки по розділу

У межах першого розділу було здійснено комплексний аналіз предметної області та досліджено сучасний стан ринку систем онлайн-бронювання в індустрії гостинності. Критичний огляд існуючих комерційних аналогів і глобальних інтернет-агрегаторів дозволив виявити низку суттєвих технологічних та економічних недоліків, притаманних готовим рішенням. Серед них ключовими є ефект прив'язки до постачальника (Vendor Lock-in), значні фінансові витрати на оплату посередницьких послуг чи абонентських плат, обмежені можливості кастомізації інтерфейсу користувача під фірмові стандарти, а також втрата цифрового суверенітету через необхідність збереження конфіденційної інформації клієнтів на сторонніх віддалених серверах.

На основі виявлених проблем було повністю обґрунтовано доцільність розробки та впровадження власної автономної веб-орієнтованої інформаційної системи. Створення такого програмного продукту з ізольованою архітектурою є раціональним технічним кроком, що дозволить готельному комплексу повністю нівелювати залежність від сторонніх вендорів, суттєво підвищити відсоток прямих інтернет-продажів, забезпечити оперативне керування номерним фондом у режимі реального часу без часових затримок, а також гарантувати максимальний рівень захисту комерційної інформації та персональних даних користувачів.

У результаті дослідження було успішно сформульовано технічне завдання та поставлено задачу на розробку. Функціональну структуру майбутньої системи чітко розмежовано відповідно до трьох основних ролей користувачів, а саме: гостя, адміністратора (менеджера) готелю та системного адміністратора. Для кожного типу користувачів визначено вичерпний набір інструментів, який охоплює весь спектр операційного циклу — від динамічного пошуку й багатофакторної фільтрації апартаментів для клієнтів до інтерактивного календаря-шахматки, керування тарифами й генерації модулів аналітичної звітності для менеджменту готелю.

Окрему увагу приділено формуванню комплексу жорстких нефункціональних вимог, які визначають критерії надійності, безпеки та ергономіки системи. Зокрема, обґрунтовано необхідність створення адаптивного дизайну за принципом Mobile First, встановлено часові ліміти відгуку серверної частини на запити користувачів, а також визначено вимоги до криптографічного захисту даних, використання захищеного протоколу HTTPS та безперервності роботи платформи. Окреслена структура вхідних і вихідних даних разом із загальною схемою взаємодії компонентів веб-системи становить повноцінне теоретичне та практичне підґрунтя для переходу до наступних етапів розробки, що включають вибір технологічного стеку, проектування бази даних та програмне конструювання системи.

РОЗДІЛ 2. ПРОЕКТУВАННЯ АРХІТЕКТУРИ ВЕБ-СИСТЕМИ

2.1. Концептуальне проєктування та розробка структурної схеми вебсистеми

Основою створення надійного, стійкого до високих навантажень та безпечного програмного забезпечення є грамотне архітектурне планування на початкових етапах життєвого циклу розробки. Для проектування інформаційної системи готельного комплексу було імплементовано сучасні принципи Чистої архітектури (Clean Architecture), розробленої Робертом Мартіном. Головна перевага цього підходу полягає в дотриманні принципу розділення відповідальності (Separation of Powers) та забезпеченні максимальної ізоляції ключових бізнес-правил від зовнішніх фреймворків, баз даних, інтерфейсів користувача та сторонніх інтеграцій. Це робить програмний продукт гнучким, тестованим та стійким до технологічних змін у майбутньому.

Згідно з обраною парадигмою, проектована веб-система концептуально розділена на чотири концентричні рівні (шари), де залежності спрямовані виключно всередину — від зовнішніх шарів до центрального ядра:

— Доменний рівень (Domain Layer): Центральне ядро системи, яке є абсолютно ізольованим. Воно містить основні сутності предметної області (наприклад, Guest, Room, Reservation, Tariff), переліки (Enums) та базову бізнес-логіку (Domain Validation), яка залишається незмінною незалежно від того, які технології чи бази даних використовуються на зовнішніх рівнях [16].

— Рівень застосунку (Application Layer): Визначає конкретні сценарії використання системи (Use Cases). Цей шар містить інтерфейси (Interfaces), обробники команд та запитів (CQRS-патерн), а також сервіси, що координують взаємодію між доменними сутностями. Він оперує абстракціями і не знає про фізичну реалізацію бази даних чи особливості відображення інтерфейсу.

— Інфраструктурний рівень (Infrastructure Layer): Відповідає за фізичну взаємодію із зовнішнім світом. Саме тут реалізується конкретний доступ до

реляційної СУБД за допомогою ORM-фреймворку, імплементуються сервіси генерації токенів безпеки (JWT), логування, а також модулі для роботи з файловими сховищами медіаконтенту (наприклад, фотографій готельних номерів).

— Рівень представлення (Presentation Layer): Зовнішній шар взаємодії з користувачем. Він складається з програмного інтерфейсу Web API (контролерів C#) та клієнтського односторінкового застосунку (SPA), реалізованого на базі бібліотеки React. Завдання цього рівня — виключно прийом вхідних запитів від користувача, передача їх на рівень застосунку та повернення результату обробки у форматі JSON.

Така розв'язана архітектура гарантує, що клієнтська частина повністю ізольована від методів збереження даних, а внутрішня бізнес-логіка не залежить від особливостей UI-компонентів React. На рисунку 2.1 наведено структурну схему взаємодії рівнів Чистої архітектури розроблюваної інформаційної системи.

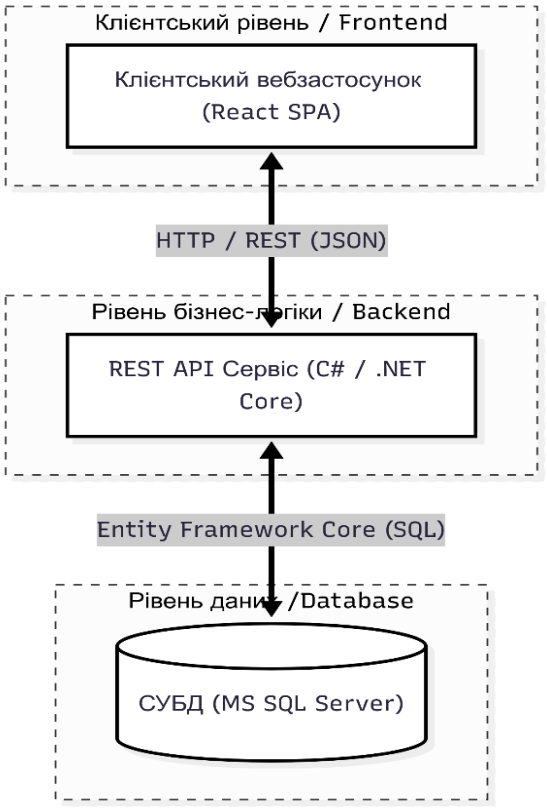

Рисунок 2.1 – Структурна схема Чистої архітектури вебсистеми

Важливим аспектом концептуального проєктування розробленої системи є суворе дотримання принципів об'єктно-орієнтованого дизайну SOLID. Впровадження цих принципів на рівні архітектури дозволило уникнути проблеми «жорсткого зчеплення» (tight coupling) компонентів:

1. Принцип єдиної відповідальності (SRP): Кожен клас і модуль системи виконує лише одну задачу. Наприклад, контролери рівня представлення відповідають виключно за прийом HTTP-запитів, тоді як валідація бізнес-правил винесена в окремі сервісні класи рівня застосунку.

2. Принцип відкритості/закритості (OCP): Архітектура системи дозволяє додавати новий функціонал (наприклад, нові типи онлайн-оплат чи нові категорії номерів) шляхом написання нового коду, без необхідності змінювати вже існуючий та протестований базовий код доменного рівня.

3. Принцип інверсії залежностей (DIP): Це ключовий принцип, на якому тримається Чиста архітектура. Високорівневі модулі (бізнес-логіка) не залежать від низькорівневих (база даних, файлова система). Натомість обидва рівні залежать від абстракцій (інтерфейсів), які реєструються в IoC-контейнері (Inversion of Control) під час запуску системи.

Для забезпечення ефективного та безпечного обміну даними між ізольованими шарами архітектури імплементовано патерн Data Transfer Object (DTO). Оскільки сутності доменного рівня (Entities) часто містять конфіденційну інформацію або навігаційні властивості, які не повинні потрапляти до клієнтського React-застосунку, система автоматично мапить (перетворює) ці сутності у полегшені об'єкти DTO перед їх відправкою через веб-інтерфейс. Це також мінімізує обсяг трафіку між клієнтом та сервером, підвищуючи загальну швидкодію системи при повільному інтернет-з'єднанні [17].

Управління життєвим циклом об'єктів та доступом до даних на інфраструктурному рівні реалізовано за допомогою класичних архітектурних шаблонів Repository (Репозиторій) та Unit of Work (Одиниця роботи). Патерн «Репозиторій» виступає абстракцією над Entity Framework Core, приховуючи від рівня бізнес-логіки складність формування SQL-запитів. Він надає

стандартизований інтерфейс у вигляді колекцій об'єктів для виконання CRUD-операцій з базою даних MS SQL Server. Своєю чергою, патерн «Одиниця роботи» гарантує транзакційну цілісність: усі зміни, внесені до кількох репозиторіїв у межах одного бізнес-процесу (наприклад, створення запису про гостя, бронювання номера та генерація рахунку), накопичуються в пам'яті сервера і зберігаються в базі даних одночасно єдиною транзакцією. Якщо на будь-якому з етапів виникає помилка, система виконує автоматичний відкат (Rollback), запобігаючи виникненню частково збережених або пошкоджених даних [18].

Крім того, на етапі концептуального проєктування були враховані аспекти мережевої безпеки та масштабованості. На межі між рівнем представлення та зовнішнім середовищем налаштовано політику CORS (Cross-Origin Resource Sharing), яка жорстко регламентує, з яких саме доменів дозволено надсилати запити до API сервера, блокуючи потенційні міжсайтові атаки. Також впроваджено механізми Rate Limiting (обмеження частоти запитів), що захищає вебсистему від автоматизованих DDoS-атак та брутфорсу (підбору паролів адміністратора).

Таким чином, використання вищезазначених архітектурних патернів перетворює систему з простого набору скриптів на корпоративне програмне забезпечення високого рівня, готове до подальшого масштабування, інтеграції з мікросервісами та безболісної підтримки у процесі експлуатації [19].

Для візуалізації функціональних вимог та чіткого визначення меж проектованої системи було застосовано методологію об'єктно-орієнтованого моделювання UML. На етапі концептуального аналізу розроблено діаграму прецедентів (Use Case Diagram), яка деталізує ключові сценарії взаємодії між системою та двома головними акторами (рисунок 2.2).

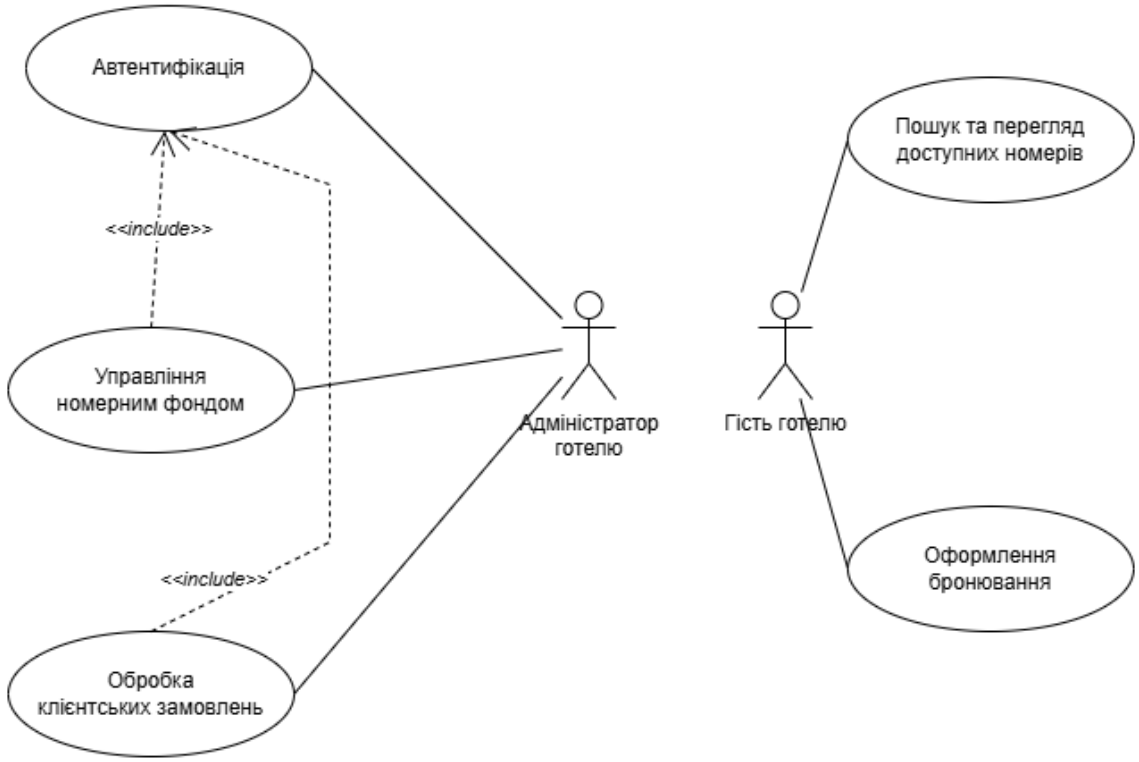

Рисунок 2.2 – UML-діаграма прецедентів системи бронювання

Згідно з розробленою діаграмою прецедентів, у межах функціонування системи чітко виділено два ключові сценарії взаємодії для актора «Гість»:

1. Ініціалізація пошукового запиту: Клієнт формує комплексні критерії відбору (бажаний таймфрейм проживання, категорія комфортності номера, гранична місткість), після чого система виконує динамічну вибірку доступних варіантів. Алгоритм автоматично відсікає об'єкти, які вже зарезервовані на ці дати іншими користувачами, або знаходяться на технічному обслуговуванні.

2. Формування транзакції резервування: Після вибору оптимального апартаменту клієнт передає свої персональні та контактні дані для оформлення замовлення. Система одразу ініціює тимчасове блокування обраного номера на період обробки транзакції, створює відповідний фінансовий запис у реляційній базі даних та генерує електронний ваучер-підтвердження, який гарантує закріплення номера за гостем.

Для актора «Адміністратор» передбачено розширений спектр повноважень, що охоплює сценарії глибокої модерації та управління щоденною операційною діяльністю готельного комплексу:

1. Управління станами номерного фонду: Надання авторизованих повноважень для виконання повного циклу CRUD-операцій. Це включає створення нових карток номерів, систематичне оновлення їхніх технічних та візуальних характеристик, гнучку зміну цінових тарифів відповідно до сезону, а також тимчасове виведення об'єктів з експлуатації на період проведення ремонтних робіт чи прибирання.

2. Модерація операційної діяльності: Здійснення безперервного оперативного аудиту вхідних заявок, управління станами замовлень (очікування транзакції, успішно оплачено, скасовано клієнтом, фактично заселено). Додатково реалізовано адміністрування системи користувацьких відгуків шляхом їх обов'язкової премодерації перед публікацією на сайті з метою уникнення спаму та нецензурної лексики.

Більш детальна динаміка взаємодії внутрішніх компонентів системи під час виконання критично важливого бізнес-процесу — створення бронювання номера — моделюється за допомогою діаграми послідовностей (Sequence Diagram), яку наведено на рисунку 2.3.

Життєвий цикл цього запиту починається з асинхронної валідації введених користувачем полів безпосередньо на стороні клієнтського React-застосунку. Після успішної перевірки формується структурований пакет даних у форматі JSON, який через захищений HTTP-протокол маршрутизується до відповідного API-контролера на рівні представлення. Далі серверна частина, побудована на платформі .NET Core, перенаправляє цей запит на рівень застосунку (Application Layer), де через інтегрований ORM-фреймворк здійснюється сувора транзакційна перевірка доступності номера в СУБД. Такий архітектурний підхід дозволяє повністю виключити ризик виникнення стану гонки (Race Condition) у базі даних при одночасному надсиланні запитів від кількох незалежних користувачів на один і той самий об'єкт. Після успішного комміту транзакції

рівень представлення повертає клієнту відповідний статус-код завершення операції та динамічно оновлює візуальний стан інтерфейсу [20].

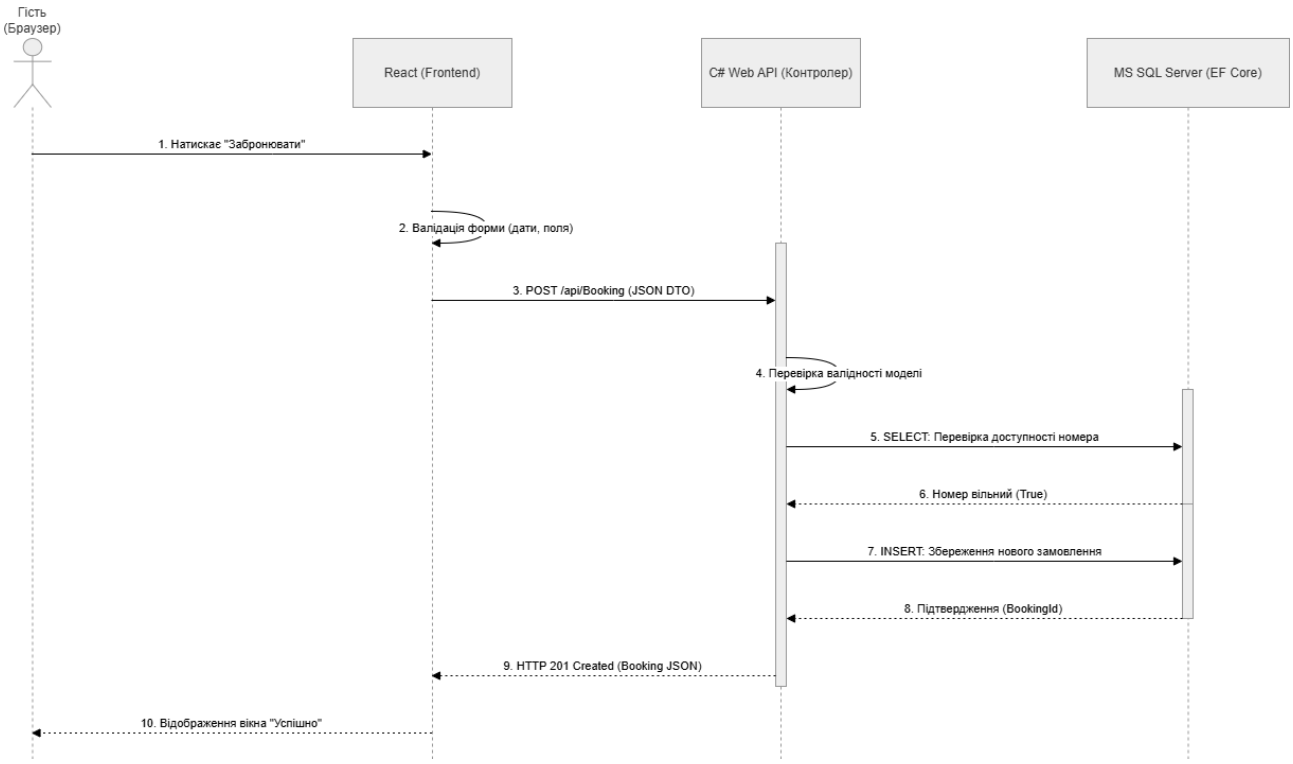

Рисунок 2.3 – UML-діаграма послідовностей

2.2. Аналіз та обґрунтування технологічного стеку

Для практичної реалізації спроєктованої архітектури було проведено глибокий порівняльний аналіз сучасних інструментальних засобів розробки. Вибір технологій здійснювався з огляду на архітектурні вимоги до відмовостійкості системи, швидкості розробки (Time-to-Market), безпеки персональних даних та продуктивності при високих навантаженнях. Оскільки система побудована на принципах Чистої архітектури (Clean Architecture), обраний стек має забезпечувати чітке розмежування обов'язків між фронтендом, бекендом та сховищем даних.

Серверна платформа та інфраструктура (Backend) У якості фундаменту для побудови серверної логіки обрано екосистему .NET Core та об'єктно-орієнтовану мову програмування C#. Під час проєктування розглядалися альтернативні

рішення, зокрема екосистема мови Java (фреймворк Spring) та мова Go (Golang). Мова Go є потужним інструментом для побудови мікросервісів завдяки легковажним горутинам, проте для систем зі складною реляційною бізнес-логікою, де наявні жорсткі зв'язки між сутностями предметної області («клієнт–замовлення–кімната») та багатоетапні транзакції, .NET Core забезпечує значно вищий рівень абстракції "з коробки" [21].

Переваги платформи ASP.NET Core для даного проєкту:

— Високопродуктивний сервер Kestrel: Вбудований кросплатформний вебсервер Kestrel використовує асинхронні цикли подій та оптимізований механізм обробки потоків I/O, що дозволяє бекенду витримувати десятки тисяч одночасних підключень без деградації продуктивності.

— Система Middleware (Конвеєр обробки): Архітектура ASP.NET Core базується на послідовному ланцюжку проміжних програмних засобів, що дозволяє елегантно інтегрувати механізми перехоплення винятків (Global Exception Handling), логування запитів та перевірку JWT-токенів автентифікації на найбільш ранніх етапах життєвого циклу HTTP-запиту.

— Сувора типізація та ООП: Статична типізація мови C# мінімізує появу логічних помилок, виявляючи архітектурні розбіжності ще на етапі компіляції коду, що є критично важливим при розрахунку вартості бронювань та застосуванні динамічних тарифів.

Рівень клієнтського інтерфейсу (Frontend) Для розробки користувацького інтерфейсу (Frontend) застосовано клієнтську JavaScript-бібліотеку React (розробка компанії Meta). Вибір здійснювався між React, Angular та Vue.js. На противагу фреймворку Angular, який нав'язує жорстку внутрішню структуру і часто є занадто важким для вебзастосунків такого обсягу, React забезпечує розробнику високу гнучкість у виборі інструментів для маршрутизації (React Router) та управління станом. Технічне обґрунтування вибору React:

— Механізм Virtual DOM: Замість прямого оновлення важкого об'єктного дерева документа (DOM) браузера, React використовує алгоритм узгодження (Reconciliation). Всі зміни стану інтерфейсу (наприклад, зміна дат

заїзду) спочатку застосовуються до віртуальної копії, обчислюється мінімальна різниця, після чого відбувається точкове оновлення екрану. Це забезпечує максимальну плавність роботи інтерфейсів бронювання.

— Функціональні компоненти та React Hooks: Використання хуків (useState, useEffect, useContext) дозволяє писати чистий, легко читабельний код без необхідності створення громіздких класових компонентів. Вся бізнес-логіка UI інкапсулюється всередині незалежних модулів, які можна перевикористовувати.

Рівень бази даних та ORM Як фізичне сховище даних (Database Layer) інтегровано реляційну систему управління базами даних Microsoft SQL Server. СУБД повністю відповідає критеріям ACID (Атомарність, Узгодженість, Ізольованість, Довговічність), що є обов'язковою вимогою для інформаційних систем, пов'язаних із фінансовими операціями. Реляційна структура найкраще підходить для збереження нормалізованих даних (до рівня 3NF), виключаючи дублювання інформації про клієнтів чи готельні номери [22].

Для взаємодії серверного коду з базою даних використовується ORM-фреймворк Entity Framework Core (EF Core):

— Парадигма Code-First: Цей підхід дозволяє розробляти структуру таблиць виключно мовою C# за допомогою сутностей (Entities). Усі зміни в структурі БД контролюються через механізм міграцій (Migrations), що дозволяє відстежувати історію змін схеми бази даних та легко розгортати систему на нових серверах.

— LINQ-запити: Замість написання сирих SQL-скриптів, розробник оперує строго типізованими колекціями даних. EF Core автоматично транслює LINQ-вирази в оптимізований T-SQL код, одночасно параметризуючи змінні, що повністю нівелює ризики атак типу SQL-ін'єкцій.

Допоміжні інструменти та інфраструктура (DevOps) Забезпечення стабільного життєвого циклу розробки (SDLC) вимагає впровадження сучасних інструментів автоматизації:

— Система контролю версій Git: Використовується для безпечного збереження вихідного коду, розгалуження функціоналу (Branching) та ведення історії розробки.

— Swagger (OpenAPI): Автоматизований інструмент для стандартизованого документування та візуалізації RESTful API. Завдяки інтеграції в екосистему ASP.NET Core, він динамічно генерує специфікацію контракту API на основі рефлексії коду та атрибутів маршрутизації. На базі цієї специфікації розгортається інтерактивна вебсторінка (Swagger UI), що надає зручний графічний інтерфейс для перегляду ендпоінтів та структур HTTP-запитів і відповідей. Це дозволяє розробникам тестувати серверну логіку без використання сторонніх клієнтів чи написання клієнтського коду, що підвищує прозорість архітектури та значно прискорює інтеграцію Frontend і Backend рівнів системи.Узагальнені критерії вибору технологічного стеку для розробки інформаційної системи систематизовано в таблиці 2.1.

Таблиця 2.1

Порівняння критерій вибору технологічного стеку

Архітектурний рівень	Обрана технологія	Ключові альтернативи	Визначальний фактор вибору
Серверний рівень (API)	C# / .NET Core	Go (Golang), Java (Spring)	Глибока інтеграція з ORM, висока продуктивність завдяки Kestrel, швидка розробка транзакційної логіки
	React.js	Angular, Svelte	Компонентна ізоляція, швидкодія завдяки Virtual DOM, використання React Hooks для управління станом
Рівень даних (СУБД)	MS SQL Server	MySQL, PostgreSQL	Гарантована ACID-сумісність, висока надійність, нативна інтеграція з екосистемою Microsoft
Механізм доступу (ORM)	Entity Framework Core	Dapper, Hibernate	Використання технології LINQ запитів, система міграцій БД
Система документування	Swagger (OpenAPI)	Postman (ручне тестування)	Автоматична генерація інтерактивної документації для REST API "з коробки"

Завдяки синергії обраних технологій створено надійний інженерний фундамент. Відсутність прямого доступу клієнта до бази даних, використання

проміжних шарів Чистої архітектури та асинхронна обробка запитів забезпечують високий рівень безпеки та швидкодії майбутнього програмного продукту [23].

2.3. Архітектура клієнт-серверної взаємодії.

Взаємодія між клієнтською частиною (виконаною на базі бібліотеки React) та серверною частиною (реалізованою на платформі .NET Core) побудована за принципами архітектурного стилю REST (Representational State Transfer). Основним протоколом передачі даних виступає HTTP, а універсальним форматом обміну повідомленнями є об'єктна нотація JSON (JavaScript Object Notation). Такий підхід забезпечує високу швидкість обробки запитів, платформонезалежність взаємодії та зручність масштабування вебсистеми. Оскільки клієнтський застосунок та серверна часть функціонують як незалежні вузли і під час розробки запускаються на різних мережевих портах, важливою складовою архітектури є налаштування політики спільного використання ресурсів з різних джерел — CORS (Cross-Origin Resource Sharing) [18]. На рівні конфігурації бекенду встановлено відповідні дозволи, які регламентують, які саме домени мають право відправляти HTTP-запити до програмного інтерфейсу, що є базовим рівнем захисту від міжсайтових підробок запитів [24].

Ключовим архітектурним рішенням на рівні обміну даними є використання патерну DTO (Data Transfer Object). Серверна частина не відправляє клієнту «сирі» сутності бази даних (моделі об'єктно-реляційного відображення Entity Framework), а трансформує їх у спеціалізовані класи передачі даних (наприклад, RoomDto, BookingDto, CreateRoomDto). Застосування цього патерну дозволяє вирішити відразу кілька важливих завдань: по-перше, приховати чутливу або внутрішню інформацію (службові ідентифікатори, паролі чи дані інших користувачів); по-третє, уникнути проблеми циклічних залежностей під час серіалізації об'єктів у JSON; по-третє, суттєво мінімізувати обсяг мережевого трафіку, передаючи лише ті поля, які дійсно необхідні для формування

користувацького інтерфейсу на фронтенді [25].

Маршрутизація системи реалізована за допомогою чітко структурованих кінцевих точок (Endpoints), кожна з яких відповідає за певну бізнес-логіку та використовує стандартні HTTP-методи. Зокрема, у розробленій системі використовуються такі базові маршрути для управління даними:

— POST /api/Auth/register — реєстрація нового користувача в системі зі збереженням хешованого пароля;

— POST /api/Auth/login — перевірка облікових даних та генерація сесійного токена доступу;

— GET /api/Room — отримання списку всіх номерів готелю з підтримкою параметрів клієнтської фільтрації;

— GET /api/Room/{id} — отримання детальної інформації про конкретний готельний номер за його ідентифікатором;

— POST /api/Room — кінцева точка для додавання нового номера до бази даних (маршрут із обмеженим доступом виключно для адміністратора);

— POST /api/Booking — ініціалізація та створення нового електронного бронювання авторизованим клієнтом готелю;

— GET /api/Booking — отримання загального масиву активних бронювань для подальшого виведення на дашборді адміністратора;

— POST /api/Review — публікація нового відгуку гостем готелю з первинним приховуванням до проходження модерації;

— PATCH /api/Review/{id}/approve — зміна статусу відгуку на підтверджений адміністратором (маршрут із обмеженим доступом).

Для захисту інформації та розмежування прав доступу в системі інтегровано механізм авторизації на базі стандарту JWT (JSON Web Tokens). Процес авторизації та взаємодії відбувається за наступним алгоритмом: після успішної перевірки облікових даних користувача бекенд генерує унікальний криптографічно підписаний JWT-токен. На стороні клієнтського застосунку цей токен зберігається у локальному сховищі браузера (localStorage). Під час виконання захищених операцій (наприклад, створення нового номера або

перегляд статистики) фронтенд автоматично додає збережений токен до заголовків HTTP-запиту у форматі Authorization: Bearer <токен>. Отримавши запит, серверна частина валідує токен, перевіряє його термін дії та наявність необхідних прав доступу (наприклад, перевіряє приналежність користувача до ролі «Admin»). Лише після успішної автентифікації та авторизації система дозволяє виконання запитуваної дії, що унеможливлює несанкціоноване втручання в роботу готельного комплексу

Невід'ємною складовою надійної клієнт-серверної архітектури є стандартизована обробка результатів виконання запитів. Серверна частина використовує загальноприйняті коди стану HTTP (HTTP Status Codes) для інформування клієнта про результат операції. У разі успішного зчитування даних повертається код 200 (OK), а при успішному створенні нового ресурсу (наприклад, нового бронювання) — код 201 (Created). Якщо дані, відправлені клієнтом, не проходять серверну валідацію, система повертає статус 400 (Bad Request) із детальним описом помилки у форматі JSON, що дозволяє React-застосунку коректно підсвітити невалідні поля форми. У випадках відсутності JWT-токена повертається статус 401 (Unauthorized), а при спробі доступу звичайного користувача до адміністративних маршрутів — статус 403 (Forbidden). Будь-які непередбачувані збої на рівні бази даних або бізнес-логіки перехоплюються глобальним обробником помилок (Global Exception Handler), який формує відповідь із кодом 500 (Internal Server Error), запобігаючи витоку чутливої системної інформації на клієнтську частину

2.4. Проєктування логічної та фізичної моделі бази даних

Ефективність функціонування, швидкість обробки запитів та стійкість до відмов створюваної вебсистеми безпосередньо залежать від якості проєктування її інформаційної архітектури. Зважаючи на потребу в забезпеченні суворої консистентності фінансових транзакцій та узгодженості даних, у ролі реляційного сховища інформації було обрано СКБД Microsoft SQL Server.

Реалізація шару доступу до даних (DAL) базується на використанні ORM-платформи Entity Framework Core з імплементацією підходу Code-First для автоматичного генерування та керування схемою через механізм міграцій

Відповідно до концепції предметно-орієнтованого проєктування (DDD), логічну структуру сховища декомпоновано на окремі агрегати — зв'язані групи сутностей, що забезпечують ізольоване виконання бізнес-правил готельного комплексу:

1. Субдомен автентифікації та контролю доступу (Access Management). Ключовим елементом цього блоку є сутність Accounts (Облікові записи), яка забезпечує збереження користувацьких профілів та перевірку прав доступу в системі. Вона пов'язана відношенням «один-до-багатьох» ($1:N$) із таблицями транзакцій бронювання та відгуків. Специфікацію атрибутів цієї сутності наведено в таблиці 2.2.

Таблиця 2.2

Параметри та атрибути сутності Accounts

Атрибут сутності	Тип даних (C#)	SQL-еквівалент	Призначення та обмеження (Fluent API)
AccountId	int	INT IDENTITY(1,1)	Первинний ключ (Primary Key),
EmailAddress	string	NVARCHAR(256)	Логін користувача.
SecurityHash	String	NVARCHAR(MAX)	Криптографічний зліпок пароля
AccessRole	string	VARCHAR(50)	Маркер авторизації

2. Субдомен керування номерним фондом (Accommodation). Центральне місце в архітектурі цього агрегату посідає сутність AccommodationUnits (Об'єкти розміщення), що описує технічні та комерційні параметри готельних номерів. Ця сутність має зв'язки типу «один-до-багатьох» із медіаконтентом, графіком доступності та відгуками клієнтів, а також зв'язок «багато-до-багатьох» ($M:N$) із довідниковою таблицею інфраструктурних зручностей. Специфікація полів AccommodationUnits подана в таблиці 2.3.

Таблиця 2.3

Специфікація полів сутності AccommodationUnits

Атрибут сутності	Тип даних (C#)	SQL-еквівалент	Призначення та обмеження (Fluent API)
UnitId	int	INT PRIMARY KEY	Унікальний ідентифікатор
Title	string	NVARCHAR(150)	Комерційне найменування номера
Details	string	NVARCHAR(MAX)	Деталізована текстова специфікація умов
StandardRate	decimal	MONEY	Базова вартість за добу
MinStayNights	int	int	Обмеження на тривалість
AdditionalGuestFee	Decimal	MONEY	Вартість розміщення додаткової особи
StandardCapacity	Int	INT	Нормативна кількість гостей у номері
MaxGuests	int	INT	Гранично допустиме навантаження на номер
SquareMeters	int	INT	Фізична площа приміщення

Для збереження та структуризації статичного графічного контенту розроблено залежну сутність MediaAssets, опис якої наведено в таблиці 2.4.

Таблиця 2.4

Схема реляційних атрибутів сутності MediaAssets

Атрибут сутності	Тип даних (C#)	SQL-еквівалент	Призначення та обмеження (Fluent API)
AssetId	int	INT IDENTITY	Первинний ключ
FilePath	string	NVARCHAR(500)	URL-посилання на збережений файл (Blob Storage)
UnitId	int	INT	Зовнішній ключ (Foreign Key) до AccommodationUnits
MediaType	String	VARCHAR(20)	Формат контенту (Image / Video)
IsThumbnail	bool	BIT	Прапорець головного зображення для прев'ю

3. Субдомен обробки транзакцій (Reservations). Сутність Reservations відстежує стан та етапи життєвого циклу оренди номерів. Дана таблиця виконує роль сполучного вузла між сутностями клієнтів (Accounts) та об'єктами розміщення (AccommodationUnits). Специфікацію її полів наведено в таблиці 2.5.

Таблиця 2.5

Технічний опис полів сутності Reservations

Атрибут сутності	Тип даних (C#)	SQL-еквівалент	Призначення та обмеження (Fluent API)
ReservationId	int	INT PRIMARY KEY	Первинний ключ транзакції
CheckInDate	DateTime	DATETIME2	Розрахунковий час прибуття клієнта
CheckOutDate	DateTime	DATETIME2	Розрахунковий час виїзду
AdultCount	int	INT	Кількість повнолітніх гостей
ChildCount	int	INT	Кількість неповнолітніх гостей
FinalAmount	decimal	MONEY	Загальна сума транзакції до сплати
ProcessStatus	string	VARCHAR(30)	Статус (Pending, Confirmed, Cancelled)
UnitId	int	INT	Foreign Key на об'єкт розміщення
AccountId	int	INT	Foreign Key на обліковий запис клієнта
ContactName	String	NVARCHAR(100)	Фактичне ім'я гостя (може відрізнятися від акаунта)
ContactPhone	string	VARCHAR(20)	Контактний телефон для зв'язку

4. Довідники та конфігураційні параметри бізнес-логіки. З метою забезпечення високого рівня гнучкості операційного керування, системної масштабованості архітектури та повного уникнення жорсткого кодування (hardcoding) ключових змінних предметної області, до структури реляційної бази даних було інтегровано спеціалізований блок допоміжних сутностей та

нормалізованих довідників. Цей структурний сегмент виступає ізольованим інструментом управління, що відповідає за тонке налаштування цінових політик, централізоване адміністрування вебплатформи та підтримку динамічної зміни параметрів середовища виконання. Завдяки цьому забезпечується висока адаптивність системи до мінливих умов бізнесу в режимі реального часу, що дозволяє коригувати поведінку додатка без втручання у вихідний код серверної частини та без потреби проведення повторного деплою:

— FacilityOptions (Інфраструктурні зручності): Базова довідникова таблиця, яка акумулює стандартизовані описи всіх доступних додаткових сервісів та зручностей готелю (наприклад, наявність системи клімат-контролю, високошвидкісного бездротового інтернету, міні-бару тощо). Вона включає первинний ключ OptionId (INT), текстову назву Title (NVARCHAR) та спеціальне поле VectorIcon (VARCHAR), яке зберігає ідентифікатор або шлях до векторного зображення. Це архітектурне рішення забезпечує динамічний та оптимізований рендеринг відповідних графічних іконок на фронтенд-частині застосунку (React), дозволяючи адміністраторам легко додавати нові зручності через панель управління без залучення розробників.

— GuestFeedbacks (Користувацькі відгуки): Сутність, що виконує роль цифрового архіву для зберігання суб'єктивних оцінок та розгорнутих текстових коментарів гостей, формуючи репутаційний профіль готелю. Структура таблиці складається з унікального ідентифікатора FeedbackId (INT), імені автора ReviewerName (NVARCHAR), самого відгуку Content (NVARCHAR(MAX)), числової оцінки Score (INT з суворим обмеженням рівня бази даних CHECK для валідації діапазону від 1 до 5), точного часу публікації PublishedAt (DATETIME2) та булевого прапорця IsModerated (BIT). Наявність останнього атрибута є критично важливою для реалізації бекенд-логіки обов'язкової попередньої модерації користувацького контенту, що гарантує надійну протидію автоматизованому спаму та публікації небажаної лексики.

— AvailabilitySchedules (Графік доступності номерного фонду): Ключова транзакційна сутність, що функціонує як глибока часова матриця. Вона

відображає деталізований стан кожного окремого апартаменту на будь-який заданий календарний день. Таблиця містить сурогатний ключ ScheduleId (INT), зовнішній ключ для зв'язку з конкретним номером UnitId (INT), цільову дату TargetDate (DATE), атрибут для реалізації механізмів динамічного сезонного ціноутворення DynamicPrice (MONEY) та спеціальний прапорець системного блокування IsLocked (BIT). Завдяки полю IsLocked операційний персонал отримує можливість гнучко керувати номерним фондом, маючи змогу превентивно вилучити номер із публічної вітрини продажів на певний період (наприклад, для проведення планового ремонту або під час технічного обслуговування).

— PricingRules (Алгоритми ціноутворення): Спеціалізована сутність, розроблена для підтримки гнучких маркетингових стратегій та автоматичного розрахунку каскадних знижок залежно від фактичних параметрів заповненості номера. До її складу входять зовнішній ідентифікатор номера UnitId, числовий ліміт кількості гостей, необхідний для тригера активації правила TargetGuestCount (INT), а також розмір дисконту DiscountValue (MONEY). Цей довідник дозволяє системі "на льоту" перераховувати фінальну вартість бронювання для різних груп клієнтів.

— SystemConfigurations (Глобальні налаштування системи): Ізольована конфігураційна таблиця, призначена для централізованого збереження глобальних параметрів середовища виконання, які впливають на поведінку всього застосунку. Показовим прикладом є наявність атрибута BookingHorizonLimit (DATE), який встановлює максимальну часову глибину (горизонт) бронювання, доступну для кінцевих користувачів. Такий підхід дає змогу керівництву готелю блокувати резервації на віддалені сезони до моменту затвердження нових актуальних прайс-листів.

Спроєктована реляційна структура бази даних є глибоко оптимізованою та суворо відповідає всім вимогам третьої нормальної форми (3NF). Такий рівень нормалізації гарантує відсутність транзитивних залежностей, унеможливлює виникнення аномалій модифікації та зводить до мінімуму дублювання

інформації. Для надійної підтримки реляційної цілісності безпосередньо на рівні СКБД, за допомогою декларативного інструментарію Fluent API було конфігуровано чіткі правила каскадного видалення та оновлення залежних записів (Cascade Delete). Це надійно запобігає засміченню бази даних «сирітськими» об'єктами при видаленні батьківських сутностей. Зважаючи на специфіку готельної системи та дуже високу частоту операцій читання, з метою максимізації загальної швидкодії під час виконання складних пошукових запитів та ресурсоємних операцій фільтрації, для критичних полів (зокрема дат бронювання, статусів та зовнішніх ключів) було спроєктовано та впроваджено систему некластерних індексів. Це рішення дозволяє суттєво зменшити час відгуку сервера при вибірці вільних номерів. Повна логічна та фізична схема зв'язків між описаними сутностями, специфікації типів даних, а також загальна архітектура сховища наочно наведені у вигляді ER-діаграми, що зображена на рисунку 2.4.

Рисунок 2.4 – Логічна ER-діаграма бази даних

2.5. Стратегія ешелонованого захисту та управління доступом

Оперування конфіденційними даними користувачів, збереження персональної інформації гостей (PII — Personally Identifiable Information) та обробка фінансових транзакцій у сфері HoReCa вимагають безкомпромісного підходу до проєктування контуру безпеки вебзастосунку. Для створюваної веборієнтованої інформаційної системи готельного комплексу було обрано та реалізовано комплексну стратегію ешелонованого захисту (Defense in Depth). Дана концепція є фундаментальним архітектурним патерном, який базується на створенні кількох незалежних, послідовних рівнів безпеки. Це гарантує, що у разі компрометації або успішного подолання зловмисником одного з бар'єрів, наступні рівні захисту заблокують просування атаки, ізолюють інфраструктуру та запобіжать несанкціонованому доступу до ядра системи й реляційного сховища даних

Рівень ідентифікації та криптографічної автентифікації на базі токенів

В основі побудови безпечної клієнт-серверної взаємодії розроблюваного вебзастосунку лежить механізм автентифікації без збереження стану сесії на сервері (Stateless Authentication). Такий підхід дозволяє суттєво підвищити масштабованість системи та знизити навантаження на оперативну пам'ять серверної частини, оскільки бекенд не зберігає сесійні файли чи записи у кеші про кожного активного користувача [26].

Для імплементації цієї логіки було інтегровано відкритий індустріальний стандарт криптографічних маркерів доступу — JSON Web Tokens (JWT). Архітектура процесу автентифікації та ідентифікації користувача функціонує за наступним багатоетапним алгоритмом:

— Ініціалізація запиту: Користувач (гість або адміністратор) надсилає свої автентифікаційні дані (пару логін/пароль) через захищену форму клієнтського React-застосунку за допомогою POST-запиту на відповідний ендпоінт API-сервера.

— Верифікація та генерація: Серверна частина на базі ASP.NET Core перевіряє відповідність отриманих даних записам у базі даних MS SQL Server. У разі успішної перевірки, спеціалізований сервіс генерації токенів створює рядок JWT, який складається з трьох частин, розділених крапками:

— Header (Заголовок): Містить метадані про тип токена (JWT) та алгоритм шифрування, який використовується для створення цифрового підпису (наприклад, HMAC SHA256).

— Payload (Корисне навантаження): Містить набір структурованих тверджень (Claims) про об'єкт користувача. Сюди інкапсулюються деперсоналізований ідентифікатор облікового запису (AccountId), електронна адреса, роль у системі, а також службові мітки часу: дата випуску маркеру (iat) та точний час закінчення терміну його дії (exp).

— Signature (Цифровий підпис): Формується шляхом гешування заголовка та корисного навантаження, закодованих у формат Base64Url, за допомогою секретного ключа (Secret Key), який відомий виключно серверній частині системи та зберігається в зашифрованих конфігураційних файлах середовища виконання.

— Передача та збереження: Згенерований маркер повертається клієнтській частині у форматі JSON-відповіді. Клієнтський SPA-застосунок кешує отриманий JWT-токен у захищеному сховищі браузера (Application Storage). Для підвищення рівня безпеки та запобігання атакам типу Session Hijacking, передача токена під час кожного наступного асинхронного HTTP-запиту здійснюється у заголовку авторизації Authorization у вигляді Bearer <token>.

— Валідація на бекенді: При отриманні запиту до захищених ресурсів, конвеєр обробки проміжного програмного забезпечення (Authentication Middleware) в ASP.NET Core автоматично видобуває токен із заголовка, десеріалізує його та проводить криптографічну перевірку підпису за допомогою секретного ключа. Якщо підпис є валідним, а часовий ліміт дії токена (Time-To-Live / TTL) не вичерпано, запит пропускається далі до шару бізнес-логіки. У разі

будь-якої невідповідності або спроби модифікації payload, сервер миттєво відхиляє запит, повертаючи HTTP-статус 401 Unauthorized.

Політико-орієнтована авторизація та розмежування привілеїв. Замість класичної, жорстко закодованої рольової моделі розмежування доступу (RBAC), яка є малогнучкою при масштабуванні функціоналу, в архітектуру системи було закладено парадигму політико-орієнтованої авторизації (Policy-Based Authorization). Це дозволяє абстрагувати правила доступу від конкретних назв ролей та будувати складні логічні ланцюжки перевірки прав на основі сукупності тверджень (Claims), що містяться в JWT-токені [27].

Усі суб'єкти взаємодії із системою розділені на чітко ізольовані домени довіри:

— Домен клієнта (Guest Policy): Надає доступ до базового пулу кінцевих точок API, що дозволяють керувати власним профілем, переглядати персональну історію фінансових транзакцій, створювати нові заявки на резервування та залишати відгуки після виселення з готелю. Клієнтський токен не має технічної можливості впливати на конфігурацію системи чи переглядати дані інших користувачів.

— Домен адміністратора (Admin Policy): Надає повний адміністративний доступ до внутрішньої панелі управління (Back-office). Дана політика вимагає наявності спеціального криптографічного маркера з високим рівнем довіри. Вона дозволяє здійснювати повний спектр CRUD-операцій над сутностями номерного фонду, змінювати цінові політики, керувати глобальними налаштуваннями календаря доступності та проводити премодерацію контенту користувачів.

Процес авторизаційної перевірки інтегровано безпосередньо у конвеєр обробки маршрутів за допомогою проміжного ПЗ (Authorization Middleware). Декорування контролерів або окремих екшенів на бекенді здійснюється за допомогою декларативних атрибутів (наприклад, [Authorize(Policy = "RequireAdminRole")]). Перевірка відповідності політикам відбувається на етапі перехоплення HTTP-запиту до моменту активації контролера та виділення ресурсів під виконання бізнес-логіки. Якщо маркер користувача не містить необхідних для даної політики прав, система блокує виконання транзакції та

повертає статус-код 403 Forbidden [28].

Рівень безпеки прикладного програмного забезпечення (Application-Level Security). Цей рівень захисту спрямований на безпосереднє нівелювання вразливостей у програмному коді застосунку та захист від поширених векторів кібератак, які класифікуються міжнародним консорціумом OWASP.

Протидія міжсайтовому скриптингу (XSS — Cross-Site Scripting). Для запобігання впровадженню шкідливого JavaScript-коду в інтерфейс застосунку та захисту кінцевих користувачів від викрадення їхніх токенів, архітектура фронтенду використовує вбудовані механізми безпеки бібліотеки React. Під час рендерингу компонентів через Virtual DOM, React автоматично виконує контекстне екранування (Data Sanitization) всіх змінних, що виводяться в інтерфейс. Будь-які текстові дані, отримані з бази даних чи вхідних форм, інтерпретуються виключно як рядкові літерали, а не як виконуваний код. Використання небезпечних методів прямого впровадження HTML-розмітки (таких як dangerouslySetInnerHTML) в межах проєкту суворо обмежено та контролюється статичними аналізаторами коду (лінтерами).

Нейтралізація спроб SQL-ін'єкцій (SQL Injection). Спроби руйнівного впливу на реляційну базу даних через впровадження сторонніх SQL-команд у текстові поля повністю нівелюються за рахунок використання ORM-системи Entity Framework Core на рівні доступу до даних. Архітектура бекенду повністю виключає використання динамічної конкатенації рядків для формування SQL-запитів.

Усі операції вибірки, фільтрації та модифікації даних описуються за допомогою строго типізованих виразів LINQ (Language Integrated Query). Під час виконання програми EF Core транслює ці вирази у параметризовані SQL-запити до MS SQL Server. На рівні СКБД параметри запиту чітко відокремлюються від тіла самої команди, тому будь-який вхідний рядок, що містить символи ін'єкцій (наприклад, ' OR '1'='1), сприймається виключно як текстове значення атрибута і не може змінити логічну структуру SQL-оператора [29].

Запобігання витоку метаданих та топології системи (Information Exposure). Для виключення можливості дослідження потенційними зловмисниками внутрішньої структури коду та топології серверної частини, в ASP.NET Core конвеєр було інтегровано глобальний перехоплювач виняткових ситуацій за допомогою паттерну Global Exception Handling Middleware.

У разі виникнення непередбачуваної критичної помилки на будь-якому рівні виконання програми (наприклад, збій зв'язку з базою даних або помилка ділення на нуль), цей компонент перехоплює виняток, логує детальний технічний стек трейс (Stack Trace) у захищені внутрішні файли логів на сервері (за допомогою Serilog) для подальшого аналізу розробниками. Клієнту ж повертається уніфікована, очищена від службової інформації відповідь у форматі стандарту RFC 7807 (Problem Details) із загальним статус-кодом 500 Internal Server Error. Це повністю приховує внутрішній устрій системи від зовнішнього спостерігача.

Криптографічний захист статичних даних. Збереження облікових записів у базі даних MS SQL Server вимагає високого рівня захисту від витоку інформації у разі фізичного компрометування файлів сховища. Зберігання паролів користувачів у відкритому або просто зашифрованому вигляді, який підлягає дешифруванню, суворо заборонено [30].

2.6 Висновки по розділу

У межах другого розділу було здійснено комплексне проєктування архітектури веб-орієнтованої інформаційної системи бронювання готельних номерів, що стало ключовим етапом інженерної розробки. В основу концептуального проєктування покладено принципи Чистої архітектури (Clean Architecture) та об'єктно-орієнтованого дизайну SOLID. Це дозволило забезпечити жорстке розмежування відповідальності, ізолювати центральне ядро бізнес-логіки від зовнішніх інтерфейсів і баз даних, а також створити гнучкий, масштабований програмний продукт, стійкий до майбутніх технологічних змін.

Для практичної реалізації розробленої архітектури було здійснено обґрунтований вибір технологічного стеку. В якості фундаменту серверної частини (Backend) обрано екосистему .NET Core та мову C#, які забезпечують високу швидкодію, сувору типізацію та потужні механізми транзакційної обробки запитів. Рівень клієнтського інтерфейсу (Frontend) спроєктовано на базі бібліотеки React, що гарантує високу реактивність застосунку завдяки механізму Virtual DOM та ізоляції компонентів. Взаємодія між цими рівнями побудована за архітектурним стилем REST, із застосуванням патерну DTO (Data Transfer Object) для оптимізації мережевого трафіку та приховування чутливої внутрішньої інформації системи.

Окрему увагу приділено проєктуванню інформаційної структури та безпеки системи. На основі підходу Code-First розроблено логічну та фізичну моделі реляційної бази даних під управлінням MS SQL Server. Модель декомпоновано на ключові предметні субдомени: управління номерним фондом, обробка транзакцій бронювання та контроль доступу. Для забезпечення надійності функціонування застосунку впроваджено стратегію ешелонованого захисту, основою якої стала система безсесійної автентифікації на базі JSON Web Tokens (JWT). Це рішення унеможливлює несанкціонований доступ до системи та гарантує надійний захист персональних і фінансових даних клієнтів. Загалом, розроблена архітектура утворює стійкий технологічний каркас, повністю готовий до етапу безпосереднього написання та тестування програмного коду.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1. Розробка серверної частини (Backend).

Процес програмного конструювання серверної частини інформаційної системи для автоматизації готельного комплексу базується на використанні кросплатформного фреймворку ASP.NET Core та об'єктно-орієнтованої мови програмування C#. Відповідно до попередньо обраної архітектурної стратегії, бекенд-вузол спроєктовано за моделлю RESTful API. Цей компонент виступає фундаментальним ядром системи, яке бере на себе відповідальність за централізовану обробку складної бізнес-логіки, маршрутизацію запитів та безпечну взаємодію з реляційним сховищем даних. Для досягнення високого рівня модульності, а також для спрощення подальшого масштабування та тестування, вихідний код серверної частини був декомпонований на чітко ізольовані логічні шари:

1. Проєктування доменного рівня (Domain Layer) Практична реалізація бекенду розпочалася з формування ядра системи — доменної моделі. На цьому етапі було розроблено набір базових класів-сутностей (Entities), що описують предметну область готельного бізнесу: Room (об'єкт розміщення), Booking (транзакція резервування), Guest (клієнт), Category (класифікація номерного фонду) та переліки на кшталт ReservationStatus (поточний стан замовлення). Головною особливістю цього шару є інкапсуляція найважливіших бізнес-правил безпосередньо всередині сутностей. Завдяки вбудованим валідаційним методам об'єкти самостійно контролюють власну цілісність, унеможливлюючи перехід у неконсистентний стан (наприклад, система фізично не дозволить створити екземпляр бронювання, де дата виїзду передує даті заїзду). Використання суворої статичної типізації мови C# на цьому рівні дозволило нейтралізувати цілий клас потенційних помилок, пов’язаних із розбіжностями форматів даних між серверною логікою та базою даних [31,32].

2. Реалізація рівня доступу до даних (Data Access Layer) На інфраструктурному рівні було розроблено механізм персистенції даних. Взаємодія з реляційною системою управління базами даних MS SQL Server імплементована через сучасний ORM-фреймворк Entity Framework Core. Зважаючи на наявність багатокрокових бізнес-процесів бронювання, архітектура цього шару була посилена застосуванням патерну Repository. Для кожної сутності створено окремий репозиторій, який абстрагує складність написання SQL-запитів і надає стандартизований інтерфейс для виконання CRUD-операцій (наприклад, вибірка вільних кімнат певної категорії). Для управління транзакційною цілісністю інтегровано архітектурний шаблон Unit of Work. Цей підхід дозволяє акумулювати декілька незалежних операцій (наприклад, реєстрацію нового профілю гостя та одночасне створення його першого бронювання) в один логічний контекст, який фіксується в базі даних як неподільна атомарна транзакція. У разі виникнення збою на будь-якому етапі, система виконує автоматичний відкат (Rollback), гарантуючи, що база даних ніколи не опиниться в частково оновленому, "пошкодженому" стані.

3. Розробка сервісного рівня (Application Layer) Найбільший обсяг обчислювальної бізнес-логіки зосереджений саме в межах сервісного шару. Тут імплементовано класи-обробники, які відповідають за реалізацію конкретних сценаріїв використання системи (Use Cases). Яскравим прикладом є сервіс BookingService та його ключовий асинхронний метод CreateBookingAsync, який інкапсулює критично важливий ланцюжок дій:

— Проведення транзакційної перевірки доступності обраного номера на запитані дати з використанням механізмів блокування в базі даних. Динамічна калькуляція фінальної вартості проживання, що враховує поточні сезонні тарифи, знижки та вартість додаткових послуг. Автоматична модифікація статусу об'єкта розміщення в загальному календарі готелю.

— Імітація інтеграції із зовнішніми SMTP-провайдерами для відправки електронного ваучера. Кожен із розроблених сервісів реєструється та

впроваджується в систему через вбудований механізм інверсії управління (Dependency Injection). Це архітектурне рішення забезпечує слабку зв'язність компонентів і дозволяє легко замінювати реальні сервіси на імітаційні об'єкти:

(mocks) під час написання Unit-тестів [33].

4. Маршрутизація та транспортний рівень REST API (Presentation Layer) Зовнішній шар серверної частини представлений API-контролерами (наприклад, RoomsController, ReservationsController). Згідно з найкращими практиками розробки, ці контролери виконують роль "тонкого" шару, делегуючи всі обчислення сервісам, а самі займаються виключно маршрутизацією HTTP-запитів та мапінгом відповідей. Обмін інформацією між клієнтом та сервером здійснюється виключно через спеціалізовані об'єкти передачі даних — DTO (Data Transfer Objects). Такий підхід гарантує безпеку системи, приховуючи від зовнішнього світу реальну топологію таблиць бази даних та запобігаючи витоку конфіденційної інформації. Для обмеження доступу до приватних ендпоінтів (наприклад, отримання фінансової звітності адміністратором) застосовується атрибут [Authorize]. Механізм авторизації функціонує на базі безсесійного стандарту криптографічних маркерів JSON Web Tokens (JWT).

5. Механізми обробки помилок та еволюція бази даних На етапі конфігурації конвеєра запитів було розроблено глобальне проміжне програмне забезпечення (Global Exception Middleware) для централізованої обробки виняткових ситуацій. Цей модуль автоматично перехоплює будь-які необроблені збої (аж до помилок виконання SQL-скриптів) і трансформує їх у безпечні, стандартизовані JSON-відповіді. Це унеможливлює випадкову відправку клієнту технічних деталей системи (Stack Traces), які можуть бути використані зловмисниками. Управління життєвим циклом структури бази даних реалізовано через систему міграцій EF Core. Кожна зміна C#-моделей конвертується у відповідний код міграції, що дозволяє синхронізувати схему бази даних між локальним середовищем розробника та бойовим сервером без ризику втрати інформації.

Технічні особливості та використані інструменти: Асинхронна модель: Для забезпечення високої пропускної здатності сервера всі операції вводу-виводу (звернення до БД, робота з мережею) імплементовано з використанням асинхронних патернів (async / await), що дозволяє серверу Kestrel не блокувати потоки обробки.

Декларативна валідація: Забезпечення цілісності вхідних даних (перевірка формату email, лімітів символів тощо) реалізовано за допомогою бібліотеки FluentValidation, що виносить правила валідації в окремі, легко читабельні класи. Моніторинг: Для оперативного контролю за станом системи інтегровано професійний інструмент структурованого логування Serilog, який фіксує всі системні події та винятки в режимі реального часу [34].

3.2. Програмне конструювання клієнтського середовища (Frontend)

Процес створення візуальної оболонки (фронтенду) для розроблюваної готельної інформаційної системи спирається на концепцію односторінкового вебдодатка (Single Page Application, SPA). Такий архітектурний підхід було обрано з метою досягнення найвищої швидкості відгуку інтерфейсу та формування безперервного користувацького досвіду (UX), який за своєю ергономікою та плавністю перемикань нагадує повноцінні десктопні чи мобільні програми. На противагу традиційним багатосторінковим сайтам, SPA-модель передбачає завантаження основного HTML-каркаса лише одноразово під час старту сеансу. Надалі всі трансформації візуальної частини, навігація між розділами та актуалізація контенту виконуються виключно динамічним шляхом, без потреби повного перезавантаження сторінки у браузері. Забезпечується це завдяки асинхронним запитам до серверного програмного інтерфейсу (REST API) та точковому оновленню необхідних елементів екрана.

Фундаментом для реалізації користувацького інтерфейсу стала популярна бібліотека React (на базі мови JavaScript), що використовує декларативний опис UI та застосовує технологію віртуального дерева документів (Virtual DOM) для

оптимізації ресурсоємних процесів рендерингу. Інструментом для збирання проєкту виступає середовище Vite, яке гарантує ефективне управління залежностями та підтримує механізм миттєвого гарячого перезавантаження (HMR) під час написання коду.

Фронтенд-додаток відрізняється суворо структурованою ієрархією модулів, розробленою з дотриманням парадигм ізоляції компонентів та принципу єдиної відповідальності (Separation of Concerns). Відповідно до наведеної схеми (рисунок 3.1), основна логіка кодової бази зосереджена в межах каталогу src. Робочий простір логічно поділено на кілька функціональних директорій. Папка assets призначена для зберігання оптимізованих графічних ресурсів, векторних зображень та глобальних файлів стилізації. У каталозі components згруповано універсальні UI-елементи (багаторазові форми, кнопки, модальні вікна), які можна перевикористовувати в різних сегментах системи, оскільки вони функціонують ізольовано від глобального контексту. Сторінки застосунку, які відповідають за унікальні URL-маршрути та управління станом дочірніх компонентів, винесені в каталог pages. Базова ініціалізація додатку та налаштування маршрутизації відбуваються у файлах App.jsx і main.jsx.

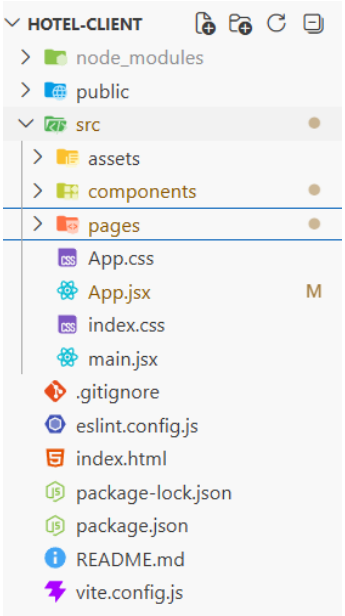

Рисунок 3.1 – Конфігурація каталогів та архітектурних модулів фронтенд-частини застосунку

Знайомство клієнта з цифровою платформою готелю починається з головної сторінки, яка відіграє роль інтерактивної вітрини. Аби максимально скоротити шлях користувача до цільової дії (оформлення броні), центральне місце в інтерфейсі (рисунок 3.2) відведено пошуковому модулю. Його основою виступає програмний віджет вибору дат у форматі календаря. Цей елемент підтримує виділення цілих діапазонів проживання і містить вбудовані алгоритми клієнтської валідації: система суворо перевіряє хронологічну послідовність (дата виїзду має бути пізнішою за заїзд), автоматично блокує минулі дні та враховує налаштовані ліміти на мінімальну тривалість перебування. Окремі селектори дають змогу вказати кількість дорослих і дітей для формування точного запиту на бекенд.

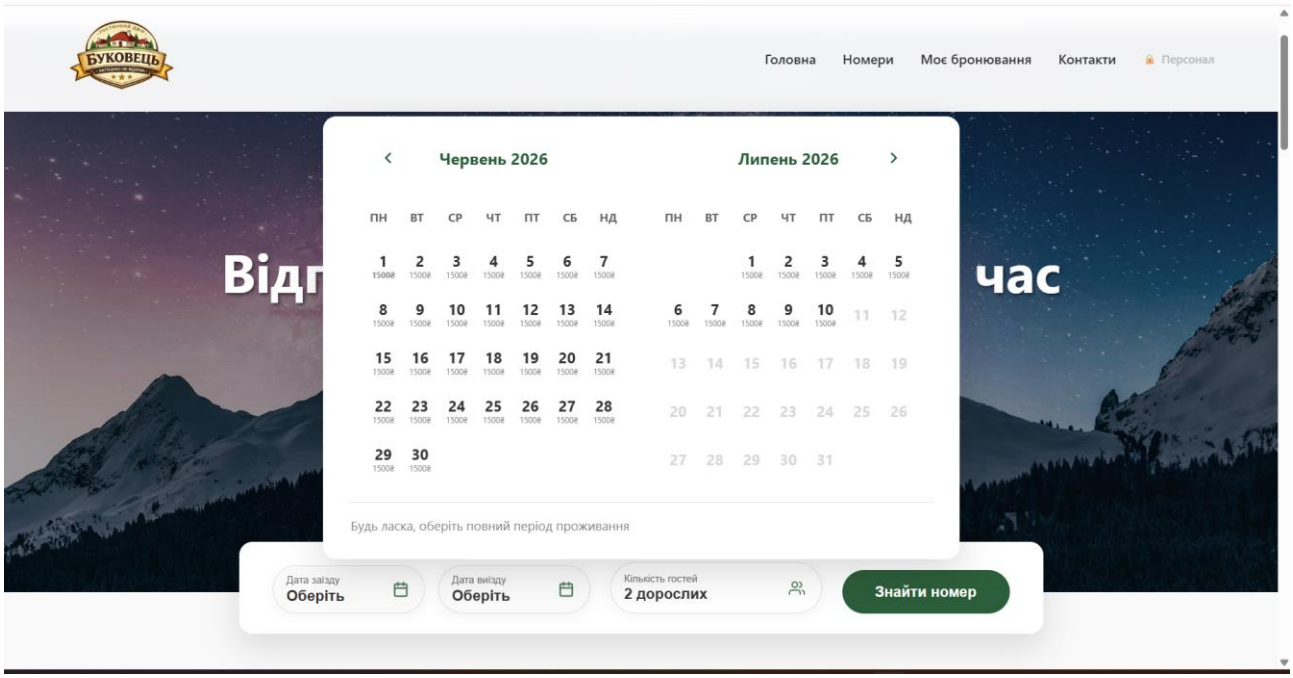

Рисунок 3.2 – Візуальна форма стартової сторінки з інтерактивним віджетом планування заїзду

Після того як серверна частина опрацьовує задані параметри, застосунок рендерить сторінку з релевантними пропозиціями розміщення. Список вільних апартаментів (рисунок 3.3) організований у форматі адаптивної сітки, що складається з карток номерів. Кожен такий об'єкт динамічно відображає ключові

відомості: медіаконтент, категорію комфорту, максимальну місткість та площу. Вартість проживання вираховується системою в режимі реального часу на основі тривалості часового інтервалу, вказаного у пошуку, що гарантує абсолютнуфінансову прозорість для клієнта.

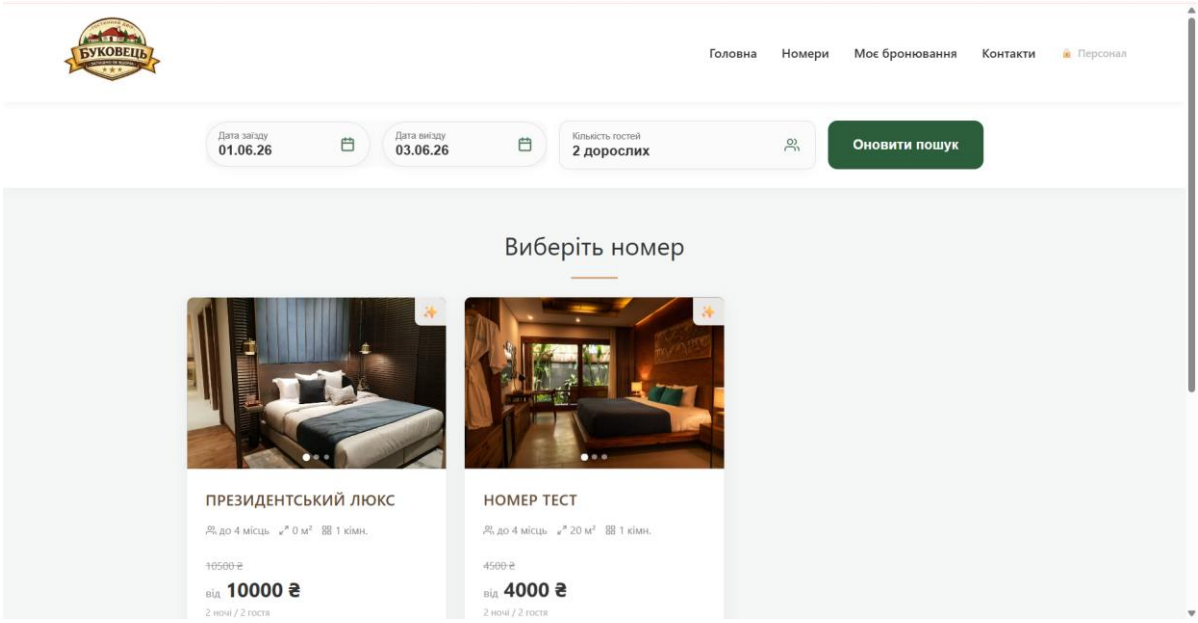

Рисунок 3.3 – Графічний екран пошукової видачі та картки доступного номерного фонду

Щоб ознайомитися з поглибленою інформацією про конкретний номер, користувачеві достатньо натиснути на відповідну картку. Для реалізації функції деталізації застосовано механізм модальних вікон (рисунок 3.4). Цей архітектурний підхід дає змогу виводити додаткові дані поверх поточного екрана, не ініціюючи маршрутизацію на нову сторінку, що зберігає результати пошуку в пам'яті браузера. Тут клієнт може переглянути розширену фотогалерею, ознайомитися з детальним переліком інфраструктурних зручностей (наприклад, клімат-контроль чи бездротовий інтернет) та миттєво перейти до підтвердження замовлення.

Після натискання на кнопку вибору система переводить гостя на етап реєстрації заявки (Checkout). Візуальний макет екрана оформлення (рисунок 3.5) поділений на дві колонки для оптимізації когнітивного сприйняття. У лівій зоні

розташована форма збору контактних даних клієнта, реалізована з використанням концепції «керованих компонентів» (Controlled Components) React. Це забезпечує синхронну перевірку вводу (наприклад, валідацію формату електронної пошти чи застосування маски для телефону), унеможливлюючи надсилання хибних запитів на сервер. Права частина інтерфейсу виконує функцію динамічного фінансового чека, візуалізуючи підсумкову калькуляцію вартості.

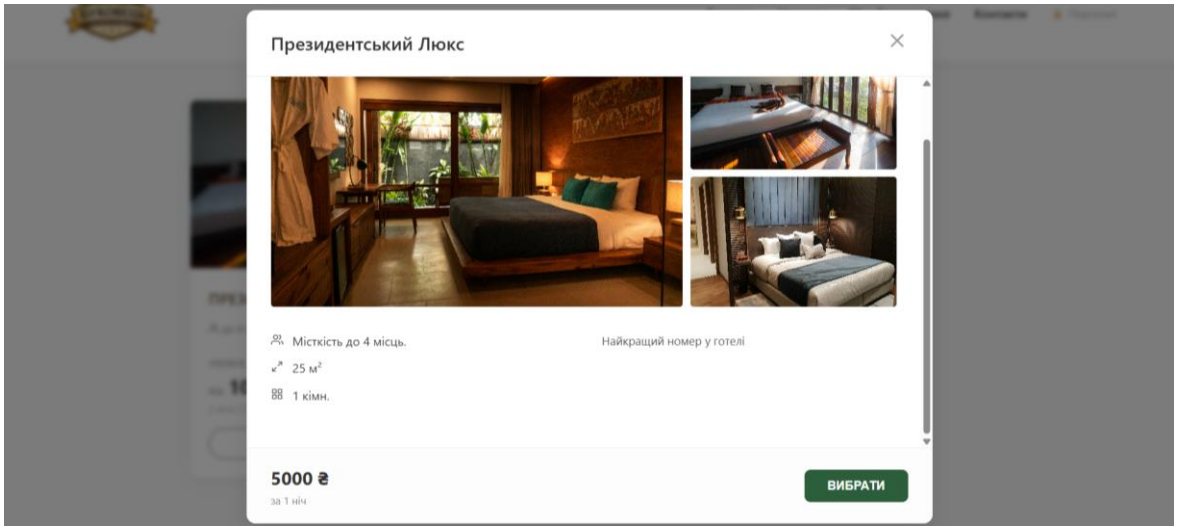

Рисунок 3.4 – Інтерфейсне вікно специфікації та інфраструктурних переваг обраного номера

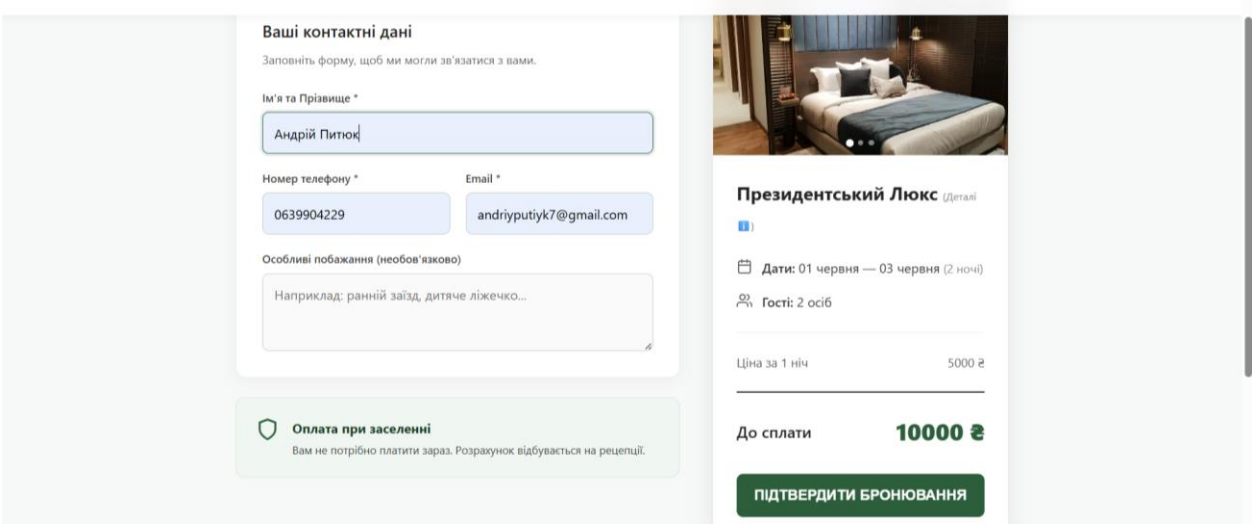

Рисунок 3.5 – Екранний інтерфейс реєстрації персональних даних та калькуляції транзакції

З метою покращення користувацького досвіду (UX) в систему інтегровано сервіс асинхронного відстеження бронювань, що працює без необхідності створення повноцінного акаунту. За допомогою цього інструменту (рисунок 3.6) гість може перевірити статус своєї заявки, використовуючи лише номер телефону та унікальний код бронювання. Фронтенд ініціює прямий GET-запит до API, зчитує інформацію з бази даних та виводить актуальний стан операції (наприклад, «Підтверджено адміністратором» або «Очікує оплати») разом з деталями заїзду.

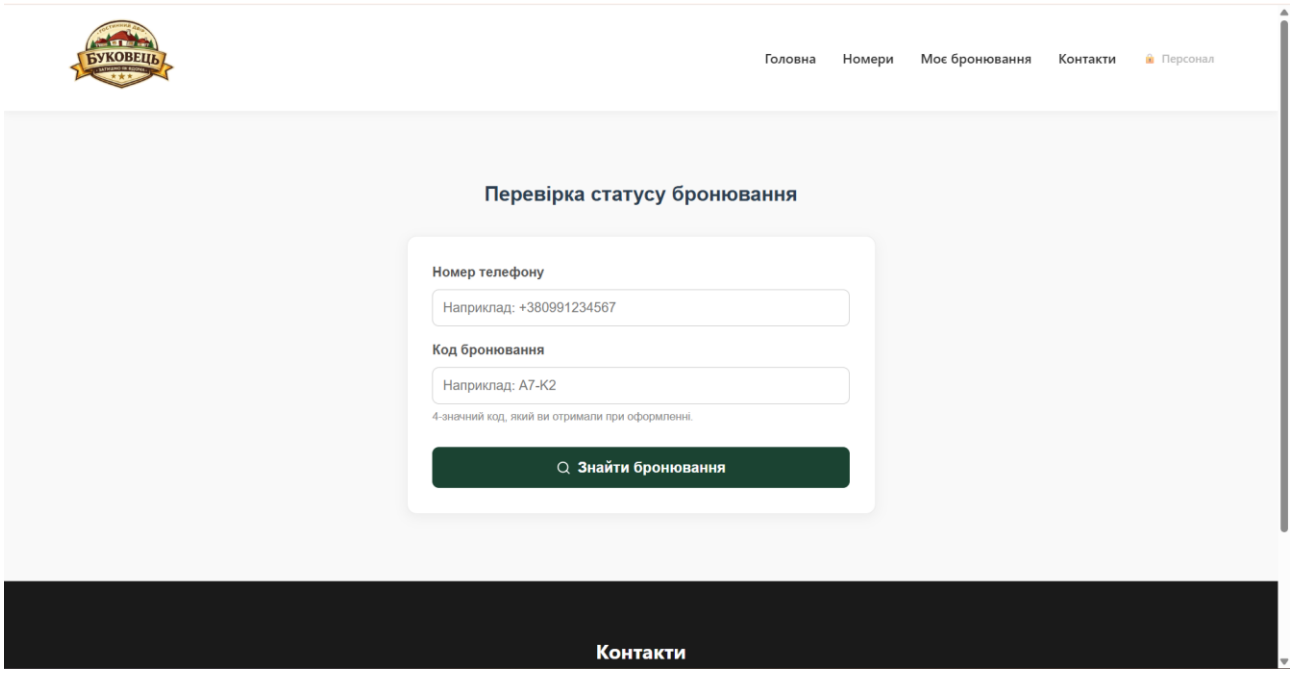

Рисунок 3.6 – Модуль самостійної перевірки поточного стану резервування за унікальним кодом

Адміністративний сегмент системи (Back-office) є повністю ізольованим від публічного доступу і вимагає проходження обов'язкової процедури автентифікації. Форма входу (рисунок 3.7) призначена для введення службових облікових даних. Після їх успішної верифікації на стороні сервера генерується спеціальний криптографічний JWT-маркер, який клієнтський застосунок надійно зберігає у пам'яті браузера. Це призводить до оновлення глобального стану авторизації та відкриває доступ до закритого функціоналу управління.

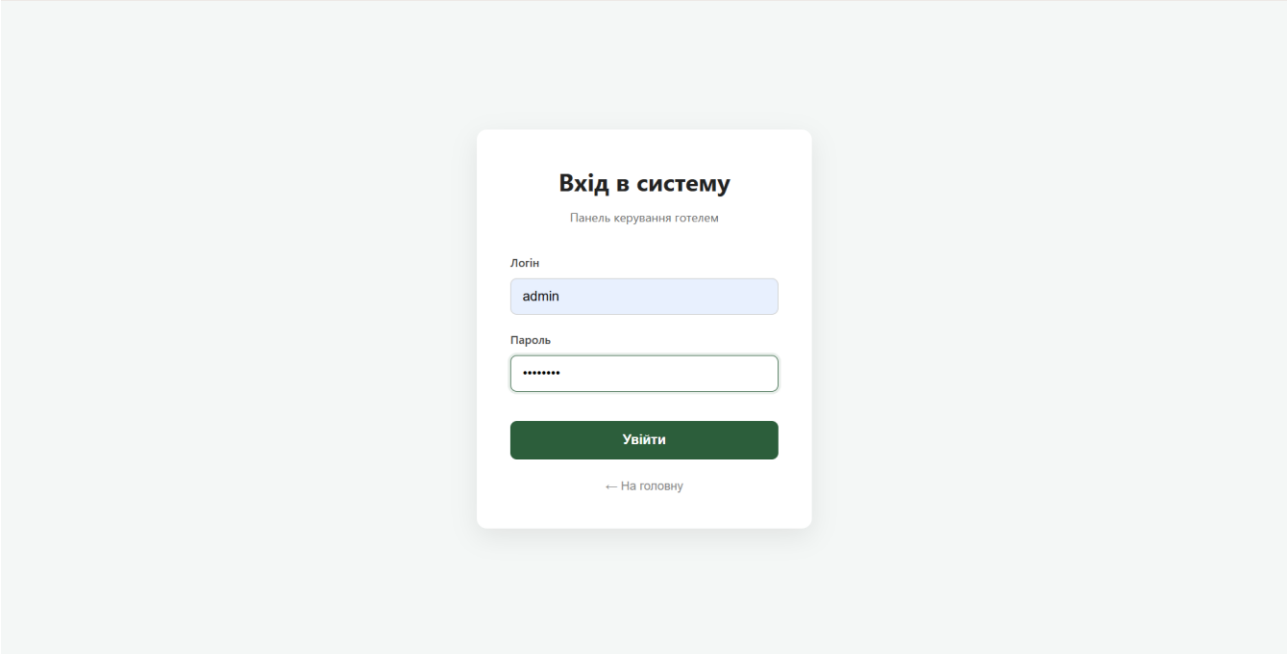

Рисунок 3.7 – Візуальний екран автентифікації для операційного та керівного персоналу

Головним робочим простором для операційного персоналу виступає дашборд, що забезпечує комплексний моніторинг ситуації в готелі. Основним UI-компонентом тут є складна інтерактивна матриця бронювань (рисунок 3.8), побудована за принципом теплової карти. Кольорове кодування (зелені клітинки позначають вільні дати, червоні — заброньовані) дозволяє персоналу миттєво оцінювати завантаженість номерного фонду.

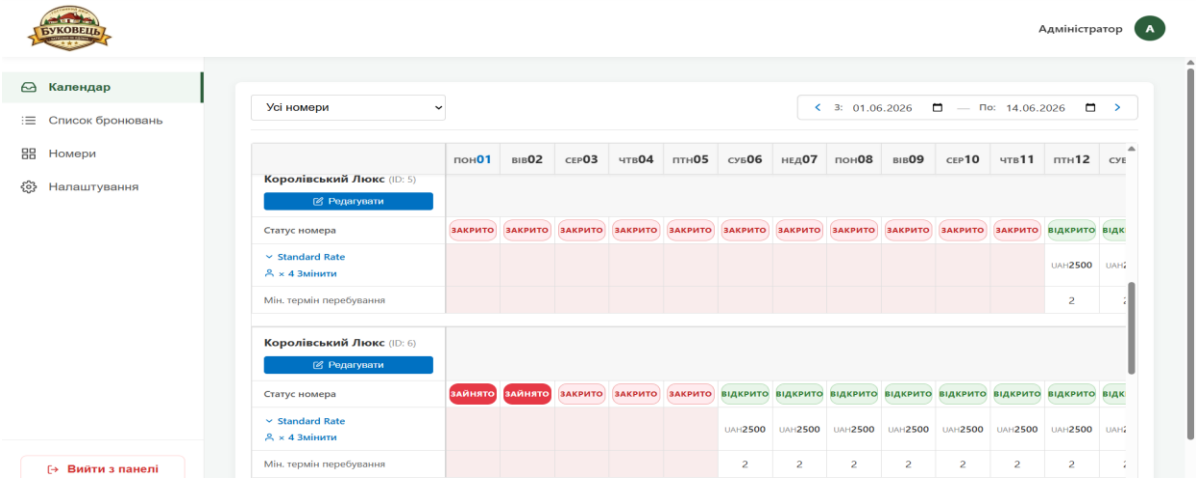

Рисунок 3.8 – Інтерактивна матриця завантаженості номерів в адміністративній панелі управління

Для ефективного адміністрування масиву вхідних заявок створено спеціалізований модуль управління транзакціями. В інформаційній таблиці (рисунок 3.9) акумулюються всі бронювання. Кожен запис структуровано за ключовими полями: ідентифікатор, дата створення, ім'я клієнта, обрані апартаменти, сума транзакції та її поточний статус. Для швидкої вибірки необхідної інформації модуль обладнано панеллю клієнтської фільтрації.

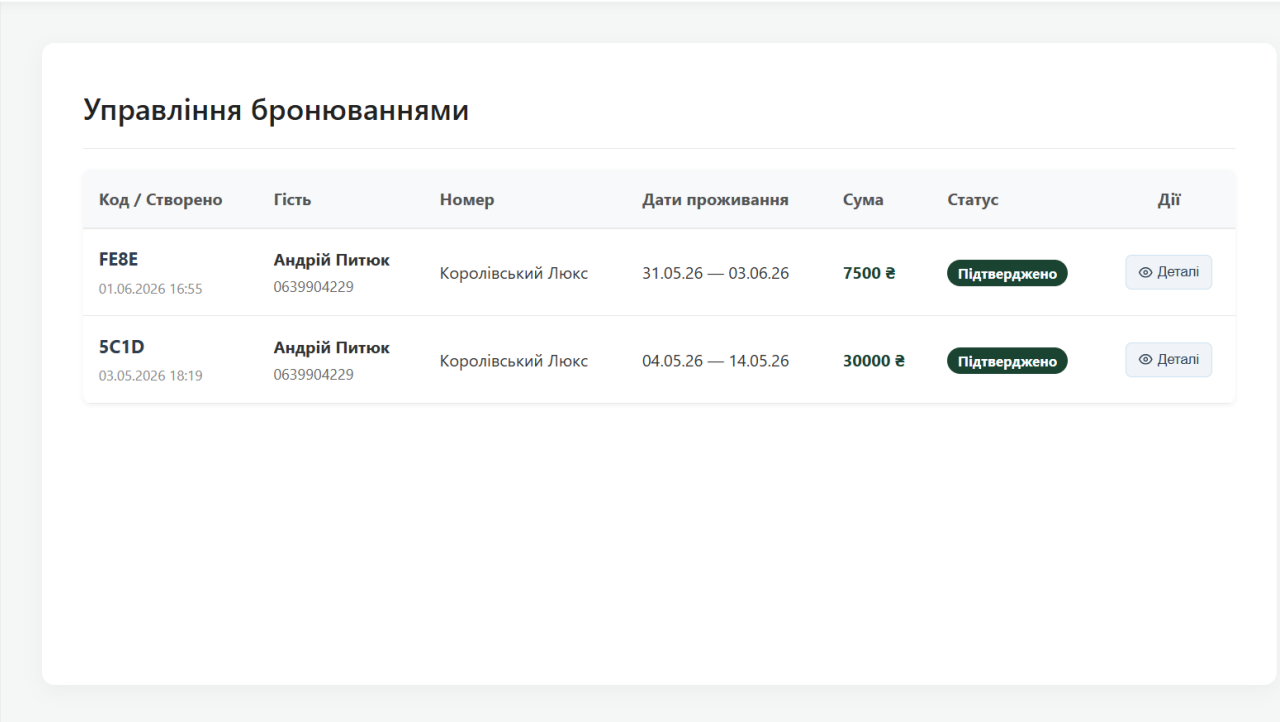

Рисунок 3.9 – Табличне представлення поточних транзакцій та заявок користувачів у системі

Якщо виникає потреба втрутитися в процес та змінити деталі замовлення, оператор використовує картку управління життєвим циклом броні. Цей інтерфейс (рисунок 3.10) рендериться у вигляді модального вікна при виборі конкретного рядка в таблиці. Він містить розширену інформацію про гостя та інструменти для управління статусом заявки. Наприклад, адміністратор може підтвердити або примусово скасувати бронювання. Будь-яка модифікація генерує REST-запит до сервера, що гарантує миттєву синхронізацію візуального стану з бекендом.

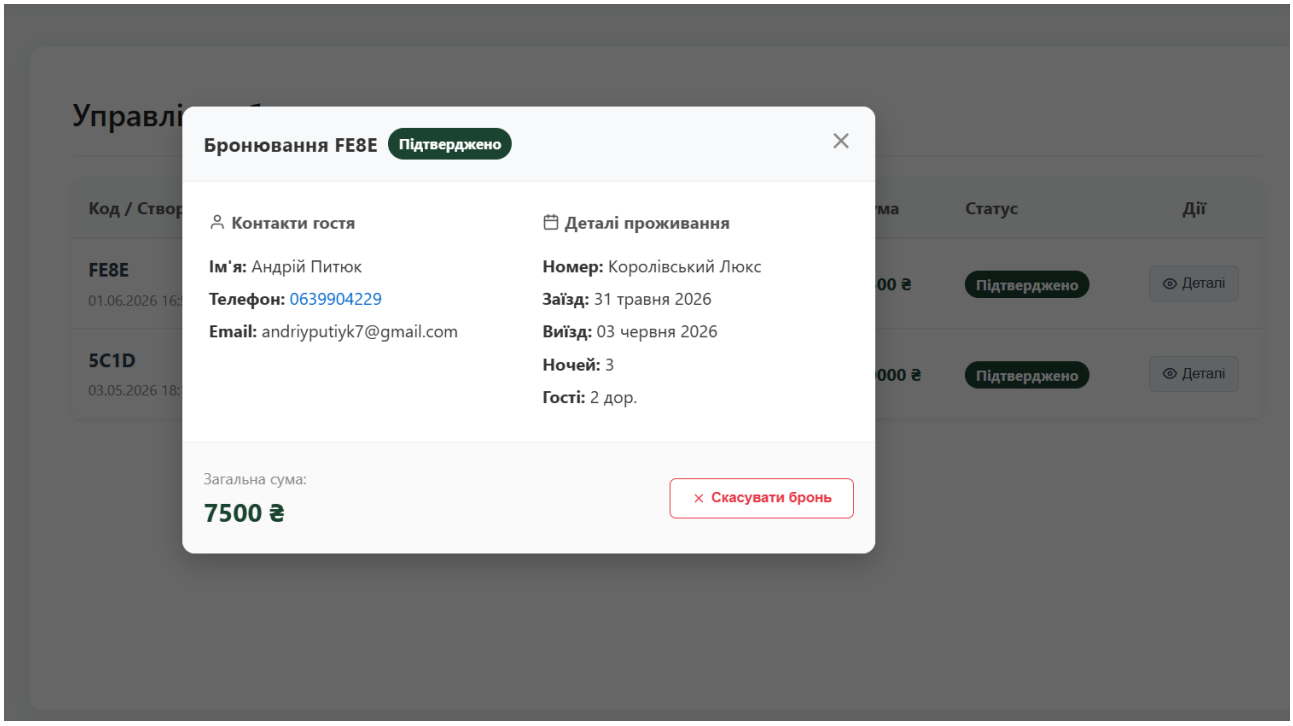

Рисунок 3.10 – Модальний інтерфейс коригування параметрів та життєвого циклу броні

Поза межами обробки клієнтських транзакцій, адміністративна панель містить інструментарій для гнучкого налаштування номерного фонду. У цьому розділі (рисунок 3.11) реалізовано графічний інтерфейс для управління категоріями номерів. Кожна сутність представлена інформаційною панеллю, де вказано базовий тариф, площу, ліміт місткості та наявні зручності. Адміністратор володіє правами на виконання повного спектра CRUD-операцій (створення, читання, оновлення, видалення об'єктів), що дозволяє повністю контролювати пропозиції готелю.

Для створення нових об'єктів розміщення або редагування існуючих передбачено спеціальну форму вводу даних. Архітектура цього компонента (рисунок 3.12) побудована за шаблоном «майстер налаштувань» (wizard), який розподіляє масив даних на кілька логічних екранів: основна конфігурація, інфраструктура та медіа-галерея. На початковому етапі вказуються текстові та цифрові змінні (назва, габарити, вартість). Розподіл процесу на кроки дозволяє суттєво знизити когнітивне навантаження на адміністратора. Окремо реалізовано

опцію керування видимістю, що дає змогу тимчасово сховати номер з публічної вітрини без видалення запису з бази.

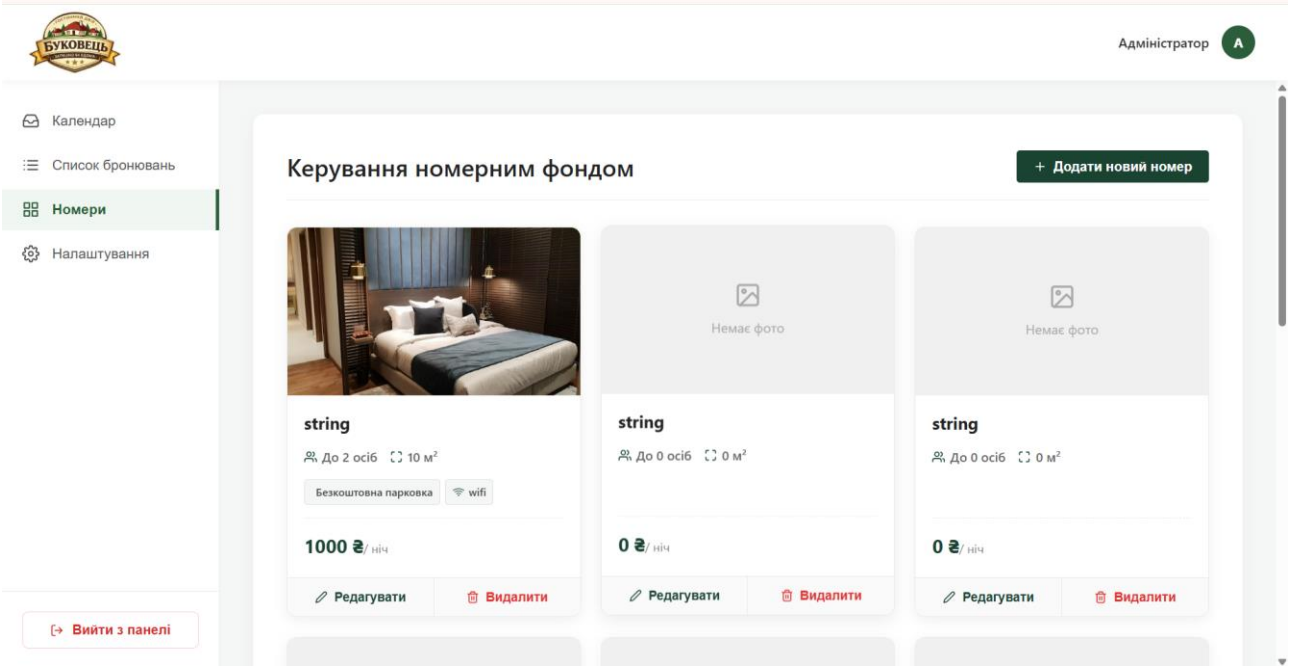

Рисунок 3.11 – Інтерфейс модуля адміністрування та редагування номерного фонду готелю

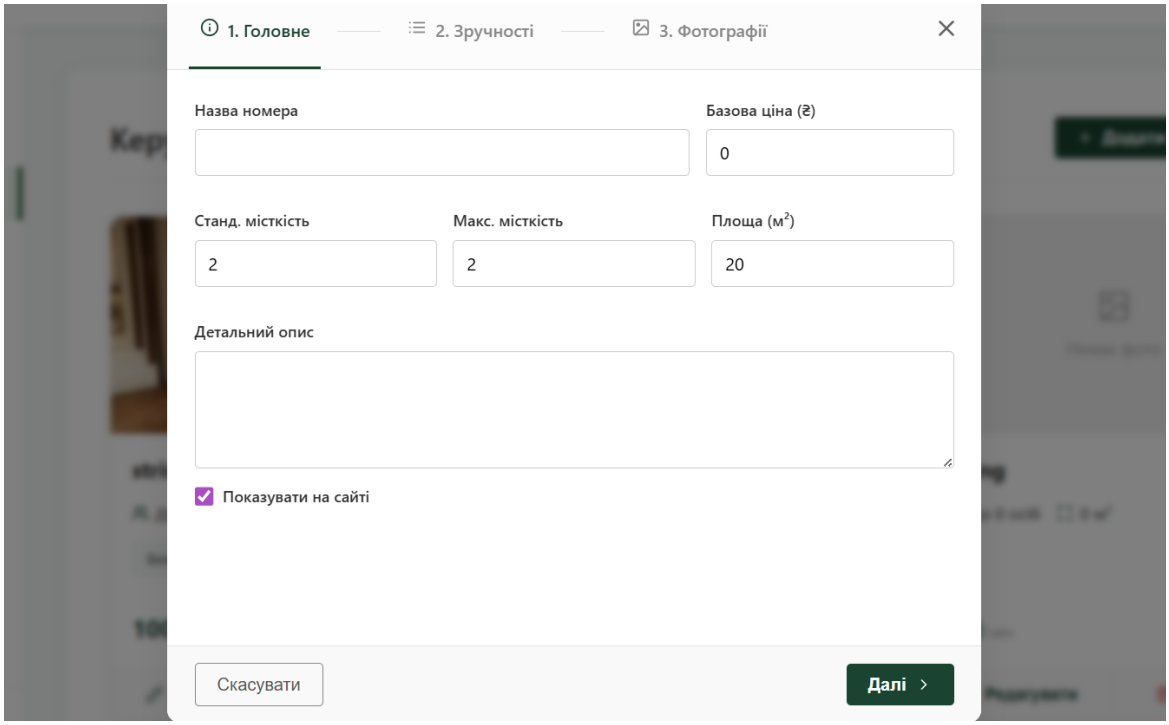

Рисунок 3.12 – Багатоекранна форма створення та детального налаштування параметрів номера

З метою контролю над високорівневою бізнес-логікою застосунку створено модуль управління системними параметрами. В даному інтерфейсі (рисунок 3.13) знаходяться налаштування, що визначають глобальну поведінку клієнтської частини. Ключовим інструментом є керування «горизонтом бронювання» — встановлення кінцевої календарної межі, до якої гостям дозволено бронювати номери онлайн. Це критично важлива функція для стратегічного планування, що дозволяє керівництву готелю вчасно змінювати цінову політику на майбутні сезони або блокувати продаж номерів під закриті заходи.

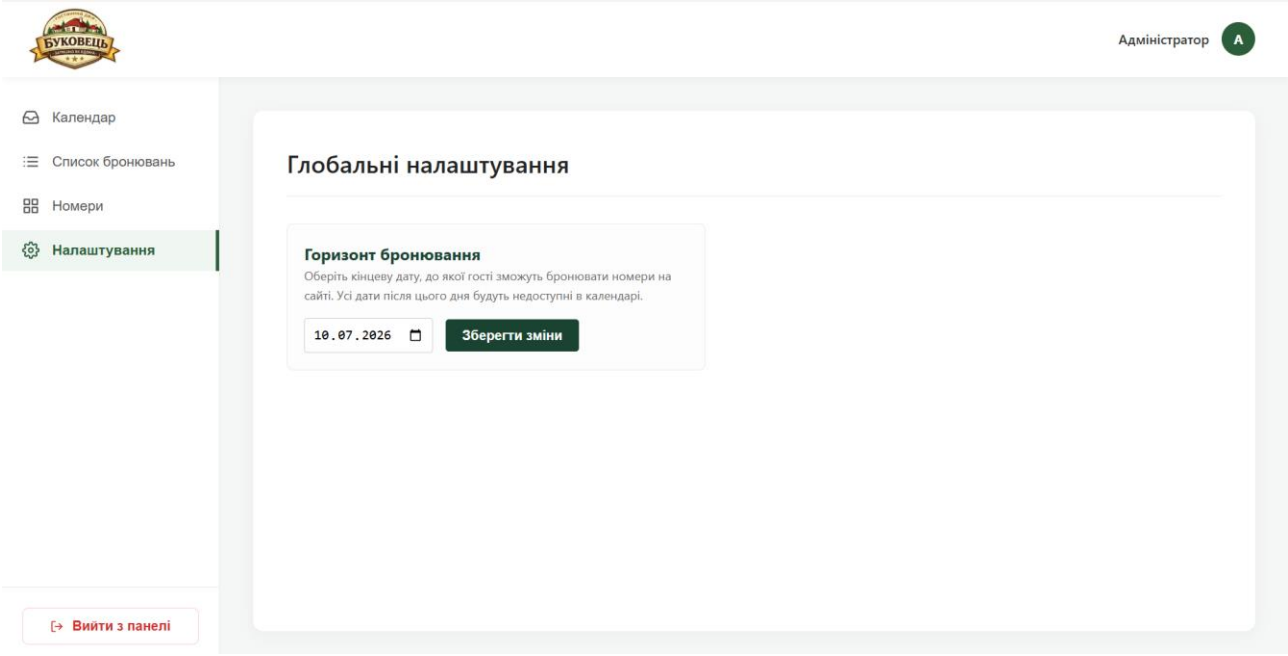

Рисунок 3.13 – Панель конфігурації глобальних параметрів та горизонту бронювання системи

В даному інтерфейсі знаходяться налаштування, що визначають глобальну поведінку клієнтської частини. Ключовим інструментом є керування «горизонтом бронювання» — встановлення кінцевої календарної межі, до якої гостям дозволено бронювати номери онлайн. Це критично важлива функція для стратегічного планування, що дозволяє керівництву готелю вчасно змінювати цінову політику на майбутні сезони або блокувати продаж номерів під закриті заходи.

3.3. Тестування програмного забезпечення та верифікація системи

Процедура тестування та гарантування якості (Quality Assurance) становить критично важливий етап у життєвому циклі створення програмного продукту. Цей процес дозволяє не лише підтвердити відповідність розробленої системи початковому технічному завданню, але й виявити приховані логічні дефекти до моменту введення продукту в комерційну експлуатацію. Для спроєктованої інформаційної системи готельного комплексу було імплементовано багаторівневу стратегію верифікації, що охоплює тестування як ізольованих серверних модулів, так і комплексного шляху користувача на клієнтському інтерфейсі. Враховуючи високу ціну системної помилки в індустрії гостинності (наприклад, втрата фінансової транзакції або збій у календарі доступності), левова частка ресурсів була спрямована на перевірку критичних бізнес-вузлів

Фундаментальним методом аудиту користувацького інтерфейсу стала методологія функціонального тестування «чорної скриньки» (Black-box testing). Цей підхід дозволив симулювати реальну поведінку кінцевих споживачів, абстрагуючись від внутрішньої програмної архітектури. Крім того, кожен етап інтеграції нового функціоналу супроводжувався глибоким регресійним тестуванням (Regression Testing). Повторна перевірка вже існуючих модулів гарантувала, що додавання нових інструментів (зокрема, панелі глобальних системних налаштувань) не порушило консистентність попередньо протестованих процесів, таких як калькуляція вартості проживання чи відображення матриці завантаженості [35].

Пріоритетним напрямком верифікації став аудит безпеки (Security Testing) адміністративного кластера. Оскільки back-office надає ексклюзивний доступ до фінансової звітності готелю та конфіденційних персональних даних клієнтів, механізм ідентифікації пройшов суворі стрес-тести. Було змодельовано серію атак типу «перебір словника» (Brute-force) та спроб авторизації з використанням навмисно пошкоджених чи застарілих облікових даних.

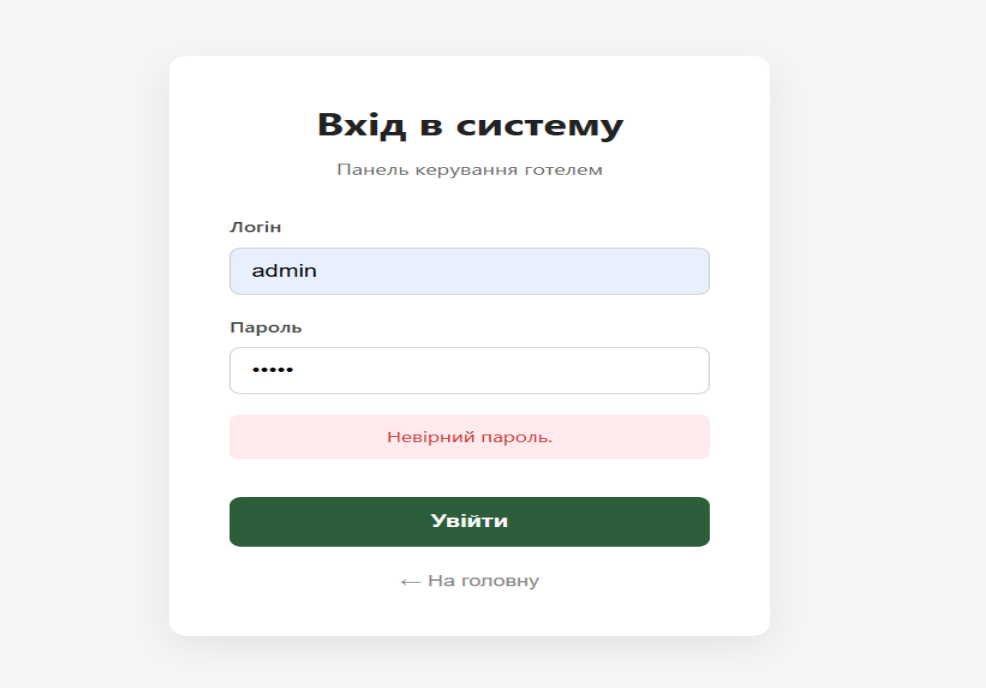

Рисунок 3.14 – Візуалізація відмови в доступі при спробі несанкціонованої автентифікації в панелі управління

Результати безпекового аудиту (рисунок 3.14) підтвердили високий рівень стійкості розробленого рішення. Серверна частина бекенду миттєво відхиляє будь-які нелегітимні HTTP-запити, повертаючи суворий статус-код 401 Unauthorized. При цьому система свідомо не розкриває технічних причин відмови (не уточнює, що саме є хибним — логін чи пароль), що є золотим стандартом захисту від атак типу «Username Enumeration» (перебір існуючих користувачів). Клієнтська частина React, своєю чергою, коректно перехоплює цей статус і виводить користувачеві безпечне інформаційне повідомлення без порушення структури інтерфейсу. Також було підтверджено механізм захисту внутрішніх маршрутів (Protected Routes): за відсутності валідного JWT-маркера у локальному сховищі браузера, спроба прямого переходу за адміністративним URL автоматично редиректить неавторизованого відвідувача на сторінку входу.

Важливим етапом стала інспекція механізмів клієнтської валідації на етапі оформлення резервації. Відсутність реєстрації вимагає від гостя ручного введення контактної інформації, що підвищує ризик людської помилки.

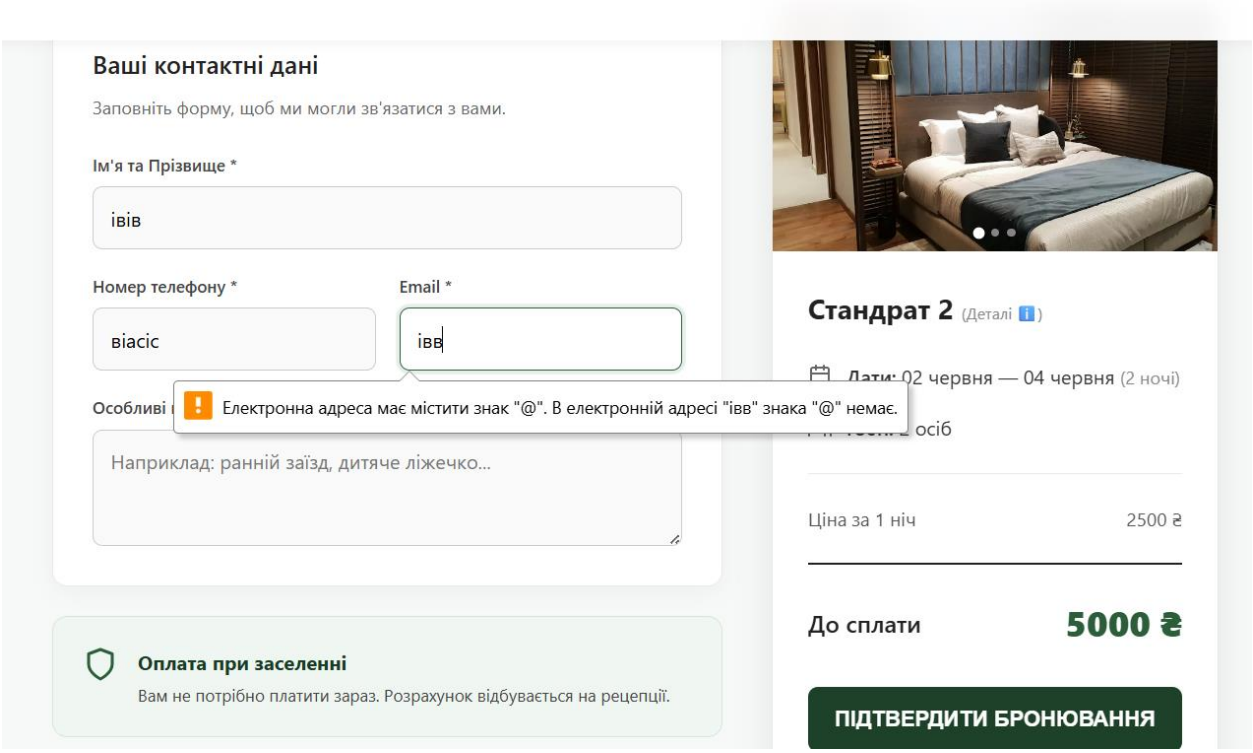

Рисунок 3.15 – Верифікація роботи алгоритмів клієнтської валідації під час заповнення персональних даних

Як демонструють результати перевірки (рисунок 3.15), впроваджена концепція «Fail-fast» працює бездоганно. Керовані форми React у режимі реального часу аналізують кожен введений символ. Система миттєво реагує на будь-яке відхилення від регулярних виразів (наприклад, введення літер у поле для номера телефону або відсутність символу @ в електронній пошті), підсвічуючи некоректні інпути маркерним кольором та виводячи текстові підказки. Такий підхід не дозволяє активувати кнопку фінального підтвердження до повного виправлення всіх помилок, що критично зменшує обсяг сміттєвого трафіку на сервер і знімає з персоналу готелю необхідність ручного коригування заявок.

Наскрізне тестування (End-to-End) цілісного циклу бронювання довело повну консистентність даних між клієнтом та сервером. Окрему увагу було приділено тестуванню конкурентності (Concurrency Testing). Для перевірки надійності транзакційної моделі було зімітовано умови високого пікового навантаження, коли кілька умовних користувачів одночасно ініціюють

бронювання одного й того ж номера на ідентичні дати. Завдяки правильно налаштованим рівням ізоляції транзакцій у MS SQL Server, система успішно заблокувала конфліктні запити, опрацювавши лише перший комміт, тим самим повністю виключивши фатальний для готелів ризик подвійного бронювання (Overbooking).

Додатково було проведено вичерпне кросбраузерне тестування макетів. Застосунок перевірявся на різних рушіях рендерингу (Blink для Google Chrome та Microsoft Edge, Gecko для Mozilla Firefox та WebKit для Apple Safari). Цей етап дозволив нівелювати дрібні розбіжності у позиціонуванні CSS-моделей Flexbox/Grid та гарантувати ідентичний, високоякісний користувацький досвід незалежно від програмних уподобань клієнта.

Інструментальний стек забезпечення якості. Для досягнення максимальної ефективності процесу тестування було задіяно професійний набір інструментів:

— Chrome Developer Tools (DevTools): Застосовувався для глибокого профілювання продуктивності React-компонентів, аналізу водоспаду мережевих запитів (вкладка Network), перевірки завантаження медіафайлів та налагодження клієнтського JavaScript-коду в режимі реального часу.

— Postman: Виступав головним середовищем для верифікації серверної логіки в обхід фронтенду. За його допомогою проводилося стрес-тестування RESTful API, верифікація коректності JWT-заголовків, а також перевірка точності JSON-відповідей відповідно до заданої DTO-схеми.

— Система контролю версій Git: Забезпечила надійний механізм управління багами (Bug Tracking). У разі виявлення критичної регресії під час тестування нового модуля, архітектура гілок (Branching) дозволяла розробнику миттєво виконати відкат (Rollback) до попередньої стабільної версії коду, що мінімізувало час простою під час налагодження.

3.4 Висновок по розлілу

У межах третього розділу було успішно здійснено програмну реалізацію та комплексне багаторівневе тестування веб-орієнтованої інформаційної системи для автоматизації готельного комплексу. Практичне конструювання серверної частини (Backend) на базі фреймворку ASP.NET Core та мови C# дозволило втілити розроблену архітектурну модель у вигляді відмовостійкого RESTful API. Завдяки чіткій декомпозиції бекенду на ізольовані шари, було послідовно реалізовано доменні сутності з вбудованою внутрішньою валідацією, шар доступу до даних за допомогою Entity Framework Core з інтеграцією патернів Repository та Unit of Work для забезпечення атомарності транзакцій, а також асинхронний сервісний рівень для обробки бізнес-сценаріїв. Безпеку транспортного рівня посилено впровадженням об'єктів передачі даних DTO, механізмом безсесійної JWT-авторизації, глобальним обробником винятків (Global Exception Middleware) для запобігання витоку системної інформації, декларативною валідацією через FluentValidation та структурованим логуванням за допомогою інструменту Serilog.

Клієнтське середовище системи (Frontend) було успішно розроблено як односторінковий вебдодаток (Single Page Application) за допомогою бібліотеки React та інструменту збирання Vite, що забезпечило високу плавність інтерфейсу та якісний користувацький досвід. Спроєктована кодова база отримала модульну структуру з розподілом на компоненти, сторінки та статичні активи. У межах публічної частини для гостей було реалізовано інтерактивний пошуковий віджет із хронологічною валідацією дат, адаптивну сітку карток номерного фонду, модальні вікна деталізації без перезавантаження сторінок, форму оформлення замовлення на основі керованих компонентів для моментальної перевірки введення, а також модуль асинхронного відстеження статусу заявок за кодом. Адміністративний сегмент системи (Back-office) було повністю ізольовано захищеними маршрутами, надаючи персоналу інструменти у вигляді інтерактивної матриці завантаженості номерів за принципом теплової карти,

таблиць модерації транзакцій, покрокових форм-майстрів (wizards) для керування номерним фондом та панелі налаштування глобальних параметрів, зокрема горизонту онлайн-бронювання.

Заключний етап роботи над розділом охоплював впровадження комплексної стратегії гарантування якості (Quality Assurance) за допомогою методів «чорної скриньки», регресійного, наскрізного та кросбраузерного тестування. Проведені стрес-тести та симуляції зловмисних дій підтвердили надійність контуру безпеки: захищені маршрути та бекенд ефективно блокують неавторизовані запити, повертаючи стандартизовані коди відповідей HTTP без розкриття вразливостей системи. На етапі оформлення замовлень алгоритми клієнтської валідації продемонстрували безпомилкову роботу за принципом «Fail-fast», мінімізуючи некоректний трафік. Найважливішим результатом тестування конкурентності стало підтвердження ефективності налаштованих рівнів ізоляції транзакцій у СКБД MS SQL Server, що дозволило повністю нівелювати стан гонки та виключити ризик подвійного бронювання номерів (Overbooking). Використання професійного інструментарію, що включав Chrome DevTools, Postman та систему Git, дозволило оперативно провести профілювання швидкодії, верифікувати REST-ендпоінти й зафіксувати повну відповідність створеного програмного продукту вихідним вимогам технічного завдання.

ВИСНОВОК

У даній бакалаврській роботі було успішно реалізовано процес проектування та програмної розробки інформаційної системи для садиби, що дозволило автоматизувати ключові бізнес-процеси бронювання та управління номерним фондом. Використання сучасного технологічного стеку, зокрема бібліотеки React для побудови інтерактивного інтерфейсу та ASP.NET Core для створення надійного серверного API, забезпечило високу продуктивність, масштабованість та безпеку розробленої системи. Усі етапи розробки проводилися у професійному інтегрованому середовищі JetBrains (зокрема WebStorm), що дозволило оптимізувати написання коду та забезпечити відповідність сучасним стандартам розробки.

Результатом роботи стало створення цілісного програмного продукту, що є логічним завершенням чотирирічного навчання в університеті та інтегрує знання, здобуті під час виконання курсових проектів з дисциплін програмної інженерії. У процесі виконання роботи було проведено глибокий аналіз предметної області, детально визначено ролі користувачів системи (клієнт та адміністратор) та окреслено її функціональні можливості, що забезпечує зручне управління бронюваннями в режимі реального часу.

Особливістю даного проекту стала його орієнтація на стандарти IT-індустрії. Для максимальної наближеності до реальних бізнес-процесів розробки були сформовані ключові проектні артефакти, що є обов’язковими для роботи бізнес-аналітика:

— Stakeholder register — дозволив ідентифікувати зацікавлені сторони та їхні впливи на проект;

— RACI matrix — забезпечив чіткий розподіл відповідальності на етапах розробки;

— Risk register — надав стратегію мінімізації потенційних загроз;

— KPI — дозволили оцінити ефективність функціонування системи та успішність реалізації бізнес-цілей.

Також було проведено комплексне тестування розробленого програмного забезпечення. Верифікація включала перевірку функціональних вимог (робота форми бронювання, логіка API, автентифікація) та нефункціональних характеристик (швидкість відгуку, захищеність від несанкціонованого доступу). У процесі аудиту коду було забезпечено відповідність стандартам «чистого коду» (Clean Code), а також розроблено детальний план рефакторингу, що гарантує високу підтримку та можливість подальшого розширення системи без загрози технічному боргу.

Підсумовуючи, можна стверджувати, що розроблена інформаційна система повністю відповідає поставленим вимогам. Вона є готовим інструментом, який здатний оптимізувати роботу садиби, покращити якість обслуговування клієнтів та підвищити ефективність управління ресурсами. Реалізований функціонал та застосовані підходи до проектування підтверджують високий рівень підготовки фахівця та готовність до вирішення складних інженерних завдань у сфері веб-розробки. У подальшому система може бути масштабована шляхом інтеграції платіжних шлюзів, модулів CRM або розширеної аналітики для власників бізнесу.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Андрейчук Ю. А., Мальська М. П. Інформаційні технології в туризмі, рекреації та готельно-ресторанному бізнесі : навч. посіб. Львів : ЛНУ імені Івана Франка, 2025. 284 с.
2. Арпуль О. В. Готельна справа: навч. посіб. Київ : Кондор, 2021. 312 с.
3. Головко О. М. Сучасні інформаційні системи управління готельним господарством. Київ : Ліра-К, 2022. 210 с.
4. Офіційний сайт сервісу онлайн-бронювання Expedia. URL: https://www.expedia.com/ (дата звернення: поточна).
5. How to Increase Direct Bookings at Your Hotel. SiteMinder Blog. URL: https://www.siteminder.com/r/direct-bookings/ (дата звернення: поточна).
6. Oracle Hospitality OPERA Cloud Property Management: огляд системи. URL: https://www.oracle.com/hospitality/opera-cloud-pms/ (дата звернення: поточна).
7. Програмне забезпечення для автоматизації готелів Bnovo. URL: https://bnovo.com/ (дата звернення: поточна).
8. Колесніков О. В. Основи проєктування інтерактивних веб-систем. Київ : Наукова думка, 2021. 195 с.
9. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. 2nd ed. Sebastopol : O'Reilly Media, 2020. 300 p.
10. Troelsen A., Japikse P. Pro C# 10 with .NET 6: Foundational Principles and Practices in Programming. 11th ed. New York : Apress, 2022. 1300 p.
11. Берко А. Ю., Верес О. М., Пасічник В. В. Моделі баз даних та знань : підручник. Львів : Магнолія-2006, 2024. 418 с.
12. Jin B., Sahni S., Shevat A. Designing Web APIs: Building APIs That Developers Love. Sebastopol : O'Reilly Media, 2018. 150 p.
13. Ларман К. Застосування UML і шаблонів проєктування. Вступ до об'єкто-орієнтованого аналізу та проєктування. Київ : Діалектика, 2019. 736 с.

14. React Router Official Documentation. URL: https://reactrouter.com/en/main (дата звернення: поточна).

15. Albahari J. C# 12 in a Nutshell: The Definitive Reference. Sebastopol : O'Reilly Media, 2023. 1060 p.

16. T-SQL Reference (Transact-SQL) for SQL Server. Microsoft Learn. URL: https://learn.microsoft.com/en-us/sql/t-sql/language-reference (дата звернення: поточна).

17. Entity Framework Core Official Documentation. Microsoft Learn. URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: поточна).

18. Fetch API Documentation. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (дата звернення: поточна).

19. OWASP API Security Top 10. URL: https://owasp.org/www-project-api-security/ (дата звернення: поточна).

20. RFC 7519: JSON Web Token (JWT). Internet Engineering Task Force (IETF). URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: поточна).

21. Лавріщева К. М. Програмна інженерія : підручник. Київ : Академперіодика, 2014. 404 с.

22. Мартін Р. Чистий код: Створення, аналіз і рефакторинг / пер. з англ. Харків : Фабула, 2019. 448 с.

23. Мартін Р. Чиста архітектура. Мистецтво розроблення програмного забезпечення / пер. з англ. Харків : Фабула, 2023. 368 с.

24. Пасічник В. В., Резніченко В. А. Організація баз даних та знань. Київ : BHV, 2011. 384 с.

25. Мальська М. П., Худо В. В. Готельний бізнес: теорія та практика : підручник. Київ : Центр учбової літератури, 2012. 392 с.

26. Сазерленд Д. Scrum. Навчіться робити вдвічі більше за менший час / пер. з англ. Харків : Клуб Сімейного Дозвілля, 2016. 280 с.

27. Myers G. J., Sandler C., Badgett T. The Art of Software Testing. 3rd ed. Hoboken : John Wiley & Sons, 2011. 240 p.

28. Miell C., Sayers A. Docker in Action. 2nd ed. Shelter Island : Manning Publications, 2019. 425 p.

29. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Reading : Addison-Wesley, 1994. 395 p.

30. Tanenbaum A. S., Wetherall D. J. Computer Networks. 5th ed. Boston : Pearson, 2011. 960 p.

31. Vite Official Documentation. URL: https://vite.dev/guide/ (дата звернення: поточна).

32. Postman API Platform Official Documentation. URL: https://learning.postman.com/docs/getting-started/introduction/ (дата звернення: поточна).

33. Chacon S., Straub B. Pro Git. 2nd ed. URL: https://git-scm.com/book/en/v2 (дата звернення: поточна).

34. React Developer Tools Documentation. URL: https://react.dev/learn/react-developer-tools (дата звернення: поточна).

35. Krug S. Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability. 3rd ed. Indianapolis : New Riders, 2014. 216 p.

ДОДАТКИ

ДОДАТОК А

Лістинг коду серверної частини

DataContext.cs

```
using HotelBookingAPI.Entities;
using HotelBookingSystem.Entities;
using Microsoft.EntityFrameworkCore;
namespace HotelBookingAPI.Data
{
   public class DataContext : DbContext
   {
      public DataContext(DbContextOptions<DataContext> options) : base(options)
      {
      }
      public DbSet<Room> Rooms { get; set; }
      public DbSet<RoomImage> RoomImages { get; set; }
      public DbSet<Amenity> Amenities { get; set; }
      public DbSet<Booking> Bookings { get; set; }
      public DbSet<Review> Reviews { get; set; }
      public DbSet<User> Users { get; set; }
      public DbSet<HotelSetting> HotelSettings { get; set; }

      public DbSet<RoomOccupancyRule> RoomOccupancyRules { get; set; }
      public DbSet<RoomAvailability> RoomAvailabilities { get; set; }

      protected override void OnModelCreating(ModelBuilder modelBuilder)
      {
         base.OnModelCreating(modelBuilder);
         // Налаштовуємо точність для грошей у Кімнатах
         modelBuilder.Entity<Room>()
            .Property(r => r.BasePrice)
            .HasColumnType("decimal(18, 0)");

         modelBuilder.Entity<Room>()

            .Property(r => r.ExtraPersonPrice)
            .HasColumnType("decimal(18, 0)");
         modelBuilder.Entity<Booking>()
            .Property(b => b.TotalPrice)
            .HasColumnType("decimal(18, 0)");

         modelBuilder.Entity<RoomAvailability>()
            .Property(ra => ra.CustomPrice)
            .HasColumnType("decimal(18, 0)");
         // Комбінація RoomId + Date має бути єдиною, щоб не було дублів дат для одного номера
         modelBuilder.Entity<RoomAvailability>()
            .HasIndex(ra => new { ra.RoomId, ra.Date })
```

```
            .IsUnique();
        // Налаштування унікальності: один номер = одне правило для конкретної кількості гостей
        modelBuilder.Entity<RoomOccupancyRule>()
            .HasIndex(r => new { r.RoomId, r.GuestCount })
            .IsUnique();
    }
  }
}
```

Booking.cs

```
namespace HotelBookingAPI.Entities
{
  public class Booking
  {
    // 1. Унікальний номер у базі даних (1, 2, 3...)
    // Використовується тільки програмістами і базою. Клієнт його не бачить.
    public int Id { get; set; }

    // Це "Зовнішній ключ" (Foreign Key). Він вказує, ЩО саме забронювали.
    // Наприклад: RoomId = 5 (це значить номер "Люкс").
    public int RoomId { get; set; }

    // Це "Навігаційна властивість".
    // Вона дозволяє нам у коді писати: booking.Room.Name
    // Тобто діставати назву кімнати прямо з бронювання.
    public Room? Room { get; set; }

    public string GuestName { get; set; } = string.Empty; //
    public string GuestPhone { get; set; } = string.Empty; // "+380..."
    public string GuestEmail { get; set; } = string.Empty; // Для чеків/листів
    // Унікальний код (PNR), який ми згенеруємо (наприклад "A7-K2").
    // Клієнт вводить Телефон + Цей код, щоб побачити своє бронювання.
    public string BookingCode { get; set; } = string.Empty;
    public DateTime CheckInDate { get; set; }  // Коли заїжджає
    public DateTime CheckOutDate { get; set; } // Коли виїжджає

    // Дата створення запису. Дуже важливо для адміна!
    // Щоб бачити: "Це бронювання впало 5 хвилин тому".
    public DateTime CreatedAt { get; set; } = DateTime.Now;
    public int Adults { get; set; }   // Дорослих
    public int Children { get; set; } // Дітей
    public string? Comments { get; set; } // "Потрібне дитяче ліжечко" (може бути пустим)
    // Фіксована сума. Ми розраховуємо її один раз при бронюванні і записуємо сюди.
    // Навіть якщо ціна номера потім зміниться, у цьому бронюванні сума залишиться старою (чесною).
    public decimal TotalPrice { get; set; }

    // Життєвий цикл броні:
    // "New" (щойно створили) -> "Confirmed" (адмін підтвердив) ->
```

```
        // "Completed" (виселилися) АБО "Cancelled" (скасували).
        public string Status { get; set; } = "New";
    }
}
```

BookingController.cs

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using HotelBookingAPI.Data;
using HotelBookingAPI.Entities;
using Microsoft.AspNetCore.Authorization;

namespace HotelBookingAPI.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class BookingController : ControllerBase
    {
        private readonly DataContext _context;

        public BookingController(DataContext context)
        {
            _context = context;
        }

        // 1. Створити бронювання (POST) - Доступно всім
        [HttpPost]
        public async Task<ActionResult<Booking>> CreateBooking(Booking booking)
        {
            // 1. Перевірка номера
            var room = await _context.Rooms.FindAsync(booking.RoomId);
            if (room == null) return NotFound("Такого номера не існує");

            // 2. Перевірка зайнятості
            var isBusy = await _context.Bookings.AnyAsync(b =>
                b.RoomId == booking.RoomId &&
                b.Status != "Cancelled" &&
                ((booking.CheckInDate >= b.CheckInDate && booking.CheckInDate < b.CheckOutDate) ||
                 (booking.CheckOutDate > b.CheckInDate && booking.CheckOutDate <= b.CheckOutDate) ||
                 (booking.CheckInDate <= b.CheckInDate && booking.CheckOutDate >= b.CheckOutDate)));

            if (isBusy) return BadRequest("Цей номер вже зайнятий на вибрані дати!");

            // 3. Розрахунки
            var days = (booking.CheckOutDate - booking.CheckInDate).Days;
            if (days < room.MinBookingDays) return BadRequest($"Мінімум {room.MinBookingDays} дні");

            decimal price = room.BasePrice;
```

```
    int extraPeople = (booking.Adults + booking.Children) - room.BaseCapacity;
    if (extraPeople > 0) price += (extraPeople * room.ExtraPersonPrice);

    booking.TotalPrice = price * days;

    // 4. Генерація коду
    booking.BookingCode = Guid.NewGuid().ToString().Substring(0, 4).ToUpper();
    booking.CreatedAt = DateTime.Now;
    booking.Status = "Confirmed";

    _context.Bookings.Add(booking);
    await _context.SaveChangesAsync();

    return Ok(booking);
}
[HttpGet("check-status")]
public async Task<ActionResult<Booking>> CheckBookingStatus(string phone, string bookingCode)
{
    var booking = await _context.Bookings
        .Include(b => b.Room)
        .Include(b => b.Room.Images)
        .FirstOrDefaultAsync(b => b.GuestPhone == phone && b.BookingCode == bookingCode);

    if (booking == null)
    {
        return NotFound("Бронювання не знайдено. Перевірте номер телефону та код.");
    }

    return Ok(booking);
}
[HttpPost("cancel")]
public async Task<ActionResult> CancelBooking(string bookingCode)
{
    var booking = await _context.Bookings.FirstOrDefaultAsync(b => b.BookingCode == bookingCode);
    if (booking == null) return NotFound("Бронювання не знайдено");

    booking.Status = "Cancelled";
    await _context.SaveChangesAsync();

    return Ok(new { message = $"Бронювання {bookingCode} успішно скасовано." });
}

// 4. Подивитися ВСІ бронювання (Тільки Адмін)
[Authorize(Roles = "Admin")]
[HttpGet("all")]
public async Task<ActionResult<List<Booking>>> GetAllBookings()
{
```

```
            var bookings = await _context.Bookings
                .Include(b => b.Room)
                .OrderByDescending(b => b.CreatedAt)
                .ToListAsync();

            return Ok(bookings);
        }

        // 5. Зміна статусу бронювання (ВИПРАВЛЕНО: без FromBody)
        [Authorize(Roles = "Admin")]
        [HttpPut("{id:int}/status")]
        public async Task<IActionResult> UpdateBookingStatus(int id, string newStatus)
        {
            var booking = await _context.Bookings.FindAsync(id);
            if (booking == null) return NotFound("Бронювання не знайдено");

            booking.Status = newStatus;
            await _context.SaveChangesAsync();

            return Ok(booking);
        }
    }
}
```

RoomController.cs

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using HotelBookingAPI.Data;
using HotelBookingAPI.Entities;
using Microsoft.AspNetCore.Authorization;

namespace HotelBookingAPI.Controllers
{
    // Це адреса, за якою сайт буде звертатися: https://localhost:xxxx/api/room
    [Route("api/[controller]")]
    [ApiController]
    public class RoomController : ControllerBase
    {
        private readonly DataContext _context;

        // Тут ми "беремо кухню" (базу даних) в роботу
        public RoomController(DataContext context)
        {
            _context = context;
        }
        // Цей метод віддає список всіх номерів
        [HttpGet]
        public async Task<ActionResult<List<Room>>> GetAllRooms()
```

```
{
    // Йдемо в базу -> беремо таблицю Rooms -> підтягуємо Картинки і Зручності -> перетворюємо в список
    var rooms = await _context.Rooms
        .Include(r => r.Images)     // Важливо! Не забудь завантажити фото
        .Include(r => r.Amenities)  // Важливо! Не забудь завантажити зручності
        .ToListAsync();

    return Ok(rooms); // Повертаємо статус 200 OK і список
}
// GET: api/room/{id}
// Цей метод віддає інформацію про ОДИН конкретний номер (для сторінки оформлення)
[HttpGet("{id:int}")]
public async Task<ActionResult<Room>> GetRoomById(int id)
{
    var room = await _context.Rooms
        .Include(r => r.Images)
        .Include(r => r.Amenities)
        .FirstOrDefaultAsync(r => r.Id == id);

    if (room == null)
        return NotFound("Кімната не знайдена");

    return Ok(room);
}

// 2. POST: api/room
// Цей метод додає новий номер
[Authorize(Roles = "Admin")]
[HttpPost]
public async Task<ActionResult<Room>> CreateRoom(Room room)
{
    // Додаємо в чергу
    _context.Rooms.Add(room);

    // Зберігаємо зміни фізично в базу
    await _context.SaveChangesAsync();

    return Ok(room); // Повертаємо доданий номер
}
// 3. POST: api/room/{id}/media
// Окремий метод, щоб додати фото або відео до існуючої кімнати
[Authorize(Roles = "Admin")]
[HttpPost("{id:int}/media")]
public async Task<ActionResult<RoomImage>> AddRoomMedia(int id, [FromBody] RoomImage media)
{
    // 1. Перевіряємо, чи існує така кімната
    var room = await _context.Rooms.FindAsync(id);
```

```
    if (room == null)
        return NotFound("Кімната не знайдена");

    // 2. Валідація для РЕАЛЬНОГО проекту (щоб адмін не написав дурниць)
    if (media.Type != "image" && media.Type != "video")
    {
        return BadRequest("Тип медіа має бути тільки 'image' або 'video'");
    }

    // 3. Прив'язуємо медіа до кімнати
    media.RoomId = id;

    _context.RoomImages.Add(media);
    await _context.SaveChangesAsync();

    return Ok(media);
}

// 4. GET: api/room/search
[HttpGet("search")]
public async Task<ActionResult<List<Room>>> SearchRooms(
    DateTime checkIn,
    DateTime checkOut,
    int adults = 1,
    int children = 0)
{
    // 1. Валідація дат
    if (checkIn.Date < DateTime.Today || checkOut.Date <= checkIn.Date)
    {
        return BadRequest("Некоректні дати бронювання.");
    }

    int totalPeople = adults + children;

    var availableRooms = await _context.Rooms
        .Include(r => r.Images)
        .Include(r => r.Amenities)
        // 2. Відсіюємо неактивні та ті, що не підходять по місткості
        .Where(r => r.IsActive && (r.BaseCapacity + 2) >= totalPeople)

        // 3. Відсіюємо зайняті іншими клієнтами (існуючі броні)
        .Where(r => !r.Bookings.Any(b =>
            b.Status != "Cancelled" &&
            (checkIn < b.CheckOutDate && checkOut > b.CheckInDate)
        ))

        // 4. НОВЕ: Відсіюємо закриті адміністратором дати
```

```
        // Якщо в діапазоні дат є хоча б один день, де IsAvailable == false, номер відкидається
        .Where(r => !r.Availabilities.Any(a =>
            a.Date >= checkIn &&
            a.Date < checkOut &&
            a.IsAvailable == false
        ))
        .ToListAsync();

    return Ok(availableRooms);
}

// 5. POST: api/room/{roomId}/amenity/{amenityId}
// Метод, щоб додати зручність до кімнати
[Authorize(Roles = "Admin")]
[HttpPost("{roomId:int}/amenity/{amenityId:int}")]
public async Task<ActionResult> AddAmenityToRoom(int roomId, int amenityId)
{
    // 1. Шукаємо кімнату (обов'язково вантажимо існуючі зручності через Include!)
    var room = await _context.Rooms
        .Include(r => r.Amenities)
        .FirstOrDefaultAsync(r => r.Id == roomId);

    if (room == null)
        return NotFound("Кімната не знайдена");

    // 2. Шукаємо зручність
    var amenity = await _context.Amenities.FindAsync(amenityId);
    if (amenity == null)
        return NotFound("Зручність не знайдена");

    // 3. Перевірка: чи не додали ми цю зручність раніше?
    if (room.Amenities.Any(a => a.Id == amenityId))
    {
        return BadRequest("Ця зручність вже є в цій кімнаті");
    }

    // 4. Додаємо зв'язок
    room.Amenities.Add(amenity);

    // 5. Зберігаємо
    await _context.SaveChangesAsync();

    return Ok(new { message = $"Зручність '{amenity.Name}' успішно додано до кімнати '{room.Name}'" });
}

// 6. PUT: api/room/{id}
// Метод для повного оновлення текстових і числових характеристик номера
```

```
[Authorize(Roles = "Admin")]
[HttpPut("{id:int}")]
public async Task<IActionResult> UpdateRoom(int id, [FromBody] Room updatedRoom)
{
    // Знаходимо існуючий номер у базі
    var room = await _context.Rooms.FindAsync(id);

    if (room == null)
        return NotFound("Кімната не знайдена");

    // Оновлюємо базові поля
    room.Name = updatedRoom.Name;
    room.Description = updatedRoom.Description;
    room.IsActive = updatedRoom.IsActive;
    room.BasePrice = updatedRoom.BasePrice;
    room.ExtraPersonPrice = updatedRoom.ExtraPersonPrice;
    room.MinBookingDays = updatedRoom.MinBookingDays;
    room.BaseCapacity = updatedRoom.BaseCapacity;
    room.MaxCapacity = updatedRoom.MaxCapacity;
    room.Area = updatedRoom.Area;
    room.RoomsCount = updatedRoom.RoomsCount;

    // Зберігаємо зміни
    await _context.SaveChangesAsync();

    return Ok(room);
}

// 7. DELETE: api/room/{roomId}/amenity/{amenityId}
// Метод для видалення зручності з конкретного номера
[Authorize(Roles = "Admin")]
[HttpDelete("{roomId:int}/amenity/{amenityId:int}")]
public async Task<ActionResult> RemoveAmenityFromRoom(int roomId, int amenityId)
{
    // Завантажуємо кімнату разом із її поточними зручностями
    var room = await _context.Rooms
        .Include(r => r.Amenities)
        .FirstOrDefaultAsync(r => r.Id == roomId);

    if (room == null)
        return NotFound("Кімната не знайдена");

    // Шукаємо зручність всередині списку цієї кімнати
    var amenity = room.Amenities.FirstOrDefault(a => a.Id == amenityId);
    if (amenity == null)
        return NotFound("Ця зручність не прив'язана до даної кімнати");
```

```
    // Видаляємо зв'язок із колекції
    room.Amenities.Remove(amenity);

    await _context.SaveChangesAsync();

    return Ok(new { message = $"Зручність успішно видалено з номера '{room.Name}'" });
}

// 8. DELETE: api/room/{id}
// Метод для видалення номера
[Authorize(Roles = "Admin")]
[HttpDelete("{id:int}")]
public async Task<IActionResult> DeleteRoom(int id)
{
    // Шукаємо кімнату разом із її фотографіями та зручностями
    var room = await _context.Rooms
        .Include(r => r.Amenities)
        .Include(r => r.Images)
        .FirstOrDefaultAsync(r => r.Id == id);

    if (room == null) return NotFound("Кімната не знайдена");

    // 1. Відв'язуємо всі зручності (щоб не було помилки бази)
    room.Amenities.Clear();

    // 2. Видаляємо всі фотографії, пов'язані з цим номером
    if (room.Images.Any())
    {
        _context.RoomImages.RemoveRange(room.Images);
    }

    // 3. Тепер безпечно видаляємо сам номер
    _context.Rooms.Remove(room);
    await _context.SaveChangesAsync();

    return Ok(new { message = "Номер успішно видалено" });
}

// 9. PUT: api/room/{id}/sync-images
// Метод для оновлення списку фотографій
[Authorize(Roles = "Admin")]
[HttpPut("{id:int}/sync-images")]
public async Task<IActionResult> SyncRoomImages(int id, [FromBody] List<string> imageUrls)
{
    var room = await _context.Rooms
        .Include(r => r.Images)
        .FirstOrDefaultAsync(r => r.Id == id);
```

```
        if (room == null) return NotFound("Кімната не знайдена");

        // 1. Видаляємо всі існуючі фотографії цього номера (очищаємо сміття)
        _context.RoomImages.RemoveRange(room.Images);

        // 2. Додаємо нові посилання, якщо вони не пусті
        foreach (var url in imageUrls)
        {
            if (!string.IsNullOrWhiteSpace(url))
            {
                room.Images.Add(new RoomImage
                {
                    Url = url,
                    Type = "image",
                    IsMain = true,
                    RoomId = id
                });
            }
        }

        await _context.SaveChangesAsync();
        return Ok(new { message = "Фотографії успішно оновлено" });
    }
  }
}
```

ДОДАТОК Б

Лістинг коду клієнтської частини

App.jsx

```
import { BrowserRouter as Router, Routes, Route, Link, Outlet } from 'react-router-dom';
import { useState } from 'react';

// Імпорти сторінок
import Home from "./pages/Home/Home";
import Booking from "./pages/Booking/Booking";
import Checkout from "./pages/Checkout/Checkout";
import Login from './pages/Login/Login';
import AdminDashboard from './pages/AdminDashboard/AdminDashboard';
import CheckBooking from './pages/CheckBooking/CheckBooking';
import './App.css';
import ScrollToTop from './components/ScrollToTop/ScrollToTop';

// Імпортуємо наш логотип для клієнтської шапки
import logo from './assets/logo.png';

// ============================================================
// 1. МАКЕТ ДЛЯ КЛІЄНТІВ (з шапкою та підвалом)
// ============================================================
function ClientLayout({ isMenuOpen, setIsMenuOpen, closeMenu }) {
 return (
   <div className="hotel-app">
     {/* --- ШАПКА --- */}
     <header className="header">

       {/* ЛОГОТИП */}

         <img src={logo} alt="Гостинний двір Буковець" className="public-navbar-logo" />
       </Link>

       <div className={`hamburger ${isMenuOpen ? 'active' : ''}`} onClick={() => setIsMenuOpen(!isMenuOpen)}>
         <span className="bar"></span>
         <span className="bar"></span>
         <span className="bar"></span>
       </div>

       <nav className={`nav-menu ${isMenuOpen ? 'active' : ''}`}>
         <Link to="/" onClick={closeMenu}>Головна</Link>
         <Link to="/booking" onClick={closeMenu}>Номери</Link>
         <Link to="/my-booking" onClick={closeMenu}>Моє бронювання</Link>
         <a href="#contacts" onClick={closeMenu}>Контакти</a>

         {/* Кнопка для персоналу */}
```

```
          <Link to="/login" className="nav-staff-link" onClick={closeMenu}>
            <span style={{ fontSize: '12px' }}>🔒</span> Персонал
          </Link>
        </nav>
      </header>

      {/* --- ДИНАМІЧНИЙ КОНТЕНТ --- */}
      <main>
        <Outlet />
      </main>

      {/* --- ПІДВАЛ --- */}
      <footer id="contacts" className="footer">
        <div className="footer-content">
          <h3>Контакти</h3>
          <p>📍 с. Буковець, Карпати</p>
          <p>📞 +380 (XX) XXX-XX-XX</p>
          <p>✉️ info@zatyshny-dvir.com</p>
        </div>
      </footer>
    </div>
  );
}

// ==========================================================
// 2. ГОЛОВНИЙ КОМПОНЕНТ APP
// ==========================================================
function App() {
  const [isMenuOpen, setIsMenuOpen] = useState(false);
  const closeMenu = () => setIsMenuOpen(false);

  return (
    <Router>
      <Routes>

        {/* === ЗОНА КЛІЄНТА === */}
        <Route element={<ClientLayout isMenuOpen={isMenuOpen} setIsMenuOpen={setIsMenuOpen} closeMenu={closeMenu}
/>}>
          <Route path="/" element={<Home />} />
          <Route path="/booking" element={<Booking />} />
          <Route path="/checkout" element={<Checkout />} />
          <Route path="/my-booking" element={<CheckBooking />} />

        {/* === ЗОНА АДМІНІСТРАТОРА === */}
        <Route path="/login" element={<Login />} />
        <Route path="/admin-dashboard" element={<AdminDashboard />} />
```

```
      </Routes>
      <ScrollToTop />

    </Router>
  );
}

export default App;
```

Home.jsx

```
import React, { useState, useEffect } from 'react';
import { useNavigate, Link } from 'react-router-dom';
import { format } from 'date-fns';
import { FiWifi, FiDroplet, FiMapPin, FiWind, FiCheckCircle, FiX } from 'react-icons/fi';

import BookingDatePicker from '../../components/BookingDatePicker/BookingDatePicker';
import GuestSelector from '../../components/GuestSelector/GuestSelector';
import './Home.css';

function Home() {
  const navigate = useNavigate();

  const [dateRange, setDateRange] = useState([null, null]);
  const [startDate, endDate] = dateRange;



  const [guestConfig, setGuestConfig] = useState({ adults: 2, children: 0, rooms: 1 });
  const [isMobile, setIsMobile] = useState(window.innerWidth <= 768);
  const [featuredRooms, setFeaturedRooms] = useState([]);
  const [isLoadingRooms, setIsLoadingRooms] = useState(true);
  const [reviews, setReviews] = useState([]);
  const [isReviewModalOpen, setIsReviewModalOpen] = useState(false);
  const [newReview, setNewReview] = useState({ authorName: '', comment: '', rating: 5 });
  const [isSubmittingReview, setIsSubmittingReview] = useState(false);

  useEffect(() => {
    const handleResize = () => setIsMobile(window.innerWidth <= 768);
    window.addEventListener('resize', handleResize);

    fetchFeaturedRooms();
    fetchReviews(); // Завантажуємо відгуки

    return () => window.removeEventListener('resize', handleResize);
  }, []);

  const fetchFeaturedRooms = async () => {
    try {
      const response = await fetch('http://localhost:5000/api/Room');
```

```
    if (response.ok) {
      const data = await response.json();
      const topRooms = data.filter(r => r.isActive).slice(0, 3);
      setFeaturedRooms(topRooms);
    }
  } catch (error) {
    console.error("Помилка завантаження номерів:", error);

    setIsLoadingRooms(false);
  }
};
const fetchReviews = async () => {
  try {
    const response = await fetch('http://localhost:5000/api/Review');
    if (response.ok) {
      const data = await response.json();
      setReviews(data);
    }
  } catch (error) {
    console.error("Помилка завантаження відгуків:", error);
  }
};

const handleSearch = (e) => {
  e.preventDefault();
  if (!startDate || !endDate) {
    alert("Будь ласка, оберіть повний період проживання (заїзд та виїзд).");
    return;
  }
  const startStr = format(startDate, 'yyyy-MM-dd');
  const endStr = format(endDate, 'yyyy-MM-dd');
  navigate(`/booking?start=${startStr}&end=${endStr}&adults=${guestConfig.adults}&children=${guestConfig.children}`);
};

const submitReview = async (e) => {
  e.preventDefault();
  setIsSubmittingReview(true);
  try {
    const response = await fetch('http://localhost:5000/api/Review', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(newReview)
    });

    if (response.ok) {
      alert("Дякуємо! Ваш відгук успішно відправлено на модерацію.");
      setIsReviewModalOpen(false);
```

```
        setNewReview({ authorName: '', comment: '', rating: 5 }); // Очищаємо форму
      } else {
        alert("Сталася помилка при відправці.");
      }
    } catch (error) {
      alert("Немає зв'язку з сервером.");
    } finally {
      setIsSubmittingReview(false);
    }
  };

  return (
    <>
      <section className="hero">
        <div className="hero-content">
          <h1>Відпочинок, де зупиняється час</h1>
          <p>Ваш затишок у серці Карпат</p>
        </div>
      </section>

      <div className="search-widget-wrapper">
        <form className="search-widget" onSubmit={handleSearch}>
          <div className="widget-group date-widget-group" style={{ flex: 3 }}>
            <BookingDatePicker dateRange={dateRange} setDateRange={setDateRange} isMobile={isMobile} />
          </div>
          <div className="widget-group guests-widget-group" style={{ flex: 2 }}>
            <GuestSelector guestConfig={guestConfig} setGuestConfig={setGuestConfig} />
          </div>
          <button type="submit" className="btn-primary search-btn">Знайти номер</button>
        </form>
      </div>

      <section className="categories-bento-section">
        <h2 className="bento-title">Наші варіанти проживання</h2>
        <p className="bento-subtitle">Оберіть ідеальний простір для вашого відпочинку</p>
        <div className="bento-grid">
          <div className="bento-item bento-large" onClick={() => navigate('/booking')}>
            <img src="https://images.unsplash.com/photo-1510798831971-661eb04b3739?q=80&w=800&auto=format&fit=crop" alt="Котедж" className="bento-img" />
            <div className="bento-overlay"><h3>Дерев'яний Котедж</h3><p>Для великої компанії • до 8 гостей</p></div>
          </div>
          <div className="bento-item bento-medium" onClick={() => navigate('/booking')}>
            <img src="https://images.unsplash.com/photo-1522708323590-d24dbb6b0267?q=80&w=800&auto=format&fit=crop" alt="Нові 4-місні апартаменти" className="bento-img" />
            <div className="bento-overlay"><h3>Нові 4-місні апартаменти</h3><p>Ідеально для сім'ї • Власна кухня</p></div>
          </div>
          <div className="bento-item bento-small" onClick={() => navigate('/booking')}>
```

```
            <img src="https://images.unsplash.com/photo-1560448204-e02f11c3d0e2?q=80&w=800&auto=format&fit=crop" alt="2-місні номери" className="bento-img" />
            <div className="bento-overlay"><h3>Затишні 2-місні номери</h3><p>Для романтичного вікенду</p></div>
          </div>
        </div>
      </section>

      <section className="about" style={{ padding: '60px 20px', textAlign: 'center', maxWidth: '800px', margin: '0 auto' }}>
        <h2 style={{ color: '#1b4332', fontSize: '2.5rem', marginBottom: '20px' }}>Гармонія з природою</h2>
        <p style={{ fontSize: '1.2rem', color: '#555', lineHeight: '1.6' }}>Ми створили простір, де немає місця міському шуму. Тільки ви, чисте гірське повітря та абсолютний комфорт. "Гостинний двір Буковець" — це місце, куди хочеться повертатися за спокоєм, цілющим релаксом та справжньою карпатською гостинністю.</p>
      </section>

      <section className="chany-section" style={{ padding: '60px 20px', backgroundColor: '#f8f9fa' }}>
        <div style={{ maxWidth: '1100px', margin: '0 auto', display: 'flex', flexWrap: 'wrap', alignItems: 'center', gap: '50px' }}>
          <div style={{ flex: '1 1 400px' }}>
            <h2 style={{ color: '#1b4332', fontSize: '2.4rem', margin: '0 0 20px 0' }}>Магія гарячих чанів</h2>
            <p style={{ fontSize: '1.15rem', color: '#555', lineHeight: '1.7', marginBottom: '20px' }}>Уявіть: прохолодний гірський вечір, мерехтіння зірок над головою, а ви занурюєтесь у гарячу джерельну воду, що парує ароматами місцевих цілющих трав та хвої.</p>
            <p style={{ fontSize: '1.05rem', color: '#666', lineHeight: '1.6', marginBottom: '30px' }}>Наші чани на відкритому вогні — це давня карпатська традиція. Вони знімають втому, перезавантажують нервову систему та дарують відчуття абсолютного спокою. Ідеальне завершення активного дня в горах.</p>
            <div style={{ display: 'flex', gap: '15px', flexWrap: 'wrap' }}>
              <span style={{ backgroundColor: '#fff', border: '1px solid #cce3f0', color: '#1b4332', padding: '8px 16px', borderRadius: '20px', fontWeight: 'bold', fontSize: '0.9rem' }}>🌿 Трав'яні збори</span>
              <span style={{ backgroundColor: '#fff', border: '1px solid #cce3f0', color: '#1b4332', padding: '8px 16px', borderRadius: '20px', fontWeight: 'bold', fontSize: '0.9rem' }}>🔥 Нагрів на дровах</span>
              <span style={{ backgroundColor: '#fff', border: '1px solid #cce3f0', color: '#1b4332', padding: '8px 16px', borderRadius: '20px', fontWeight: 'bold', fontSize: '0.9rem' }}>✨ Просто неба</span>
            </div>
          </div>
          <div style={{ flex: '1 1 400px', position: 'relative' }}>
            <div style={{ borderRadius: '16px', overflow: 'hidden', boxShadow: '0 15px 35px rgba(0,0,0,0.15)', transform: 'rotate(2deg)', transition: 'transform 0.4s' }} onMouseOver={(e) => e.currentTarget.style.transform = 'rotate(0deg)'} onMouseOut={(e) => e.currentTarget.style.transform = 'rotate(2deg)'}>
              <img src="https://images.unsplash.com/photo-1544367567-0f2fcb009e0b?q=80&w=800&auto=format&fit=crop" alt="Карпатський чан" style={{ width: '100%', height: '400px', objectFit: 'cover', display: 'block' }} />
            </div>
          </div>
        </div>
      </section>

      <section className="amenities-overview" style={{ backgroundColor: '#fff', padding: '80px 20px' }}>
        <div style={{ maxWidth: '1200px', margin: '0 auto' }}>
          <h2 style={{ textAlign: 'center', color: '#1b4332', marginBottom: '50px' }}>Наші переваги</h2>
```

```
        <div style={{ display: 'grid', gridTemplateColumns: 'repeat(auto-fit, minmax(220px, 1fr))', gap: '40px', textAlign: 'center' }}>
          <div style={{ padding: '10px' }}><div style={{ fontSize: '3rem', color: '#ffc107', marginBottom: '15px' }}><FiWind /></div><h3 style={{ marginBottom: '10px' }}>Чисте повітря</h3><p style={{ color: '#666' }}>Екологічно чиста зона Карпат, ідеально для відновлення сил.</p></div>
          <div style={{ padding: '10px' }}><div style={{ fontSize: '3rem', color: '#ffc107', marginBottom: '15px' }}><FiWifi /></div><h3 style={{ marginBottom: '10px' }}>Швидкий Wi-Fi</h3><p style={{ color: '#666' }}>Залишайтеся на зв'язку або працюйте віддалено з видом на гори.</p></div>
          <div style={{ padding: '10px' }}><div style={{ fontSize: '3rem', color: '#ffc107', marginBottom: '15px' }}><FiDroplet /></div><h3 style={{ marginBottom: '10px' }}>Оздоровчі чани</h3><p style={{ color: '#666' }}>Розслаблення у гарячій воді з травами просто неба.</p></div>
          <div style={{ padding: '10px' }}><div style={{ fontSize: '3rem', color: '#ffc107', marginBottom: '15px' }}><FiMapPin /></div><h3 style={{ marginBottom: '10px' }}>Зручна локація</h3><p style={{ color: '#666' }}>Легкий доїзд та близькість до основних туристичних маршрутів.</p></div>
        </div>
      </div>
    </section>

    <section className="featured-rooms-section" style={{ padding: '80px 20px', backgroundColor: '#f8f9fa' }}>
      <div style={{ maxWidth: '1200px', margin: '0 auto' }}>
        <div style={{ display: 'flex', justifyContent: 'space-between', alignItems: 'flex-end', marginBottom: '40px', flexWrap: 'wrap', gap: '20px' }}>
          <div><h2 style={{ color: '#1b4332', fontSize: '2.2rem', margin: '0 0 10px 0' }}>Популярні номери</h2><p style={{ color: '#666', margin: 0, fontSize: '1.1rem' }}>Оберіть простір, який підійде саме вам</p></div>
          <Link to="/booking" style={{ color: '#1b4332', fontWeight: 'bold', textDecoration: 'none', borderBottom: '2px solid #ffc107' }}>Дивитися всі →</Link>
        </div>
        {isLoadingRooms ? <div style={{ textAlign: 'center', padding: '40px' }}>Завантаження номерів...</div> : (
          <div style={{ display: 'grid', gridTemplateColumns: 'repeat(auto-fit, minmax(300px, 1fr))', gap: '30px' }}>
            {featuredRooms.map(room => (
              <div key={room.id} style={{ borderRadius: '12px', overflow: 'hidden', boxShadow: '0 4px 15px rgba(0,0,0,0.08)', backgroundColor: '#fff', transition: 'transform 0.3s' }} className="room-card-hover">
                <div style={{ height: '220px', backgroundColor: '#e0e0e0', position: 'relative' }}>
                  {room.images && room.images.length > 0 ? <img src={room.images[0].url} alt={room.name} style={{ width: '100%', height: '100%', objectFit: 'cover' }} /> : <div style={{ display: 'flex', alignItems: 'center', justifyContent: 'center', height: '100%', color: '#999' }}>Фото готується</div>}
                  <div style={{ position: 'absolute', top: '15px', right: '15px', backgroundColor: '#ffc107', color: '#1b4332', padding: '6px 16px', borderRadius: '20px', fontWeight: 'bold', fontSize: '0.9rem' }}>Від {room.basePrice} ₴</div>
                </div>
                <div style={{ padding: '25px' }}>
                  <h3 style={{ margin: '0 0 10px 0', color: '#2c3e50', fontSize: '1.4rem' }}>{room.name}</h3>
                  <p style={{ color: '#666', fontSize: '0.95rem', marginBottom: '20px', display: '-webkit-box', WebkitLineClamp: '2', WebkitBoxOrient: 'vertical', overflow: 'hidden', minHeight: '40px' }}>{room.description || "Затишний номер."}</p>
                  <div style={{ display: 'flex', gap: '20px', color: '#555', fontSize: '0.9rem', marginBottom: '25px' }}>
                    <span style={{ display: 'flex', alignItems: 'center', gap: '6px' }}><FiCheckCircle color="#1b4332" /> До {room.maxCapacity} гостей</span><span style={{ display: 'flex', alignItems: 'center', gap: '6px' }}><FiCheckCircle color="#1b4332" /> {room.area} м²</span>
                  </div>
```

```
              <button onClick={() => navigate('/booking')} style={{ width: '100%', padding: '14px', backgroundColor: '#f8f9fa', border: '1px solid #1b4332', color: '#1b4332', borderRadius: '8px', fontWeight: 'bold', cursor: 'pointer', transition: 'all 0.2s', fontSize: '1rem' }}>Перевірити дати</button>
            </div>
          </div>
        ))}
      </div>
    )}
  </div>
</section>
<section className="reviews-section" style={{ padding: '80px 20px', backgroundColor: '#fff' }}>
  <div style={{ maxWidth: '1200px', margin: '0 auto' }}>
    <h2 style={{ textAlign: 'center', color: '#1b4332', fontSize: '2.4rem', marginBottom: '10px' }}>Що кажуть наші гості</h2>
    <p style={{ textAlign: 'center', color: '#666', fontSize: '1.1rem', marginBottom: '40px' }}>Справжні емоції тих, хто вже обрав відпочинок у нас</p>

    {reviews.length === 0 ? (
      <div style={{ textAlign: 'center', color: '#888', fontStyle: 'italic', marginBottom: '30px' }}>Поки що немає відгуків. Станьте першим!</div>
    ) : (
      <div style={{ display: 'grid', gridTemplateColumns: 'repeat(auto-fit, minmax(300px, 1fr))', gap: '30px', marginBottom: '40px' }}>
        {reviews.map(review => (
          <div key={review.id} style={{ backgroundColor: '#f8f9fa', padding: '35px 25px', borderRadius: '16px', boxShadow: '0 4px 15px rgba(0,0,0,0.03)', position: 'relative' }}>
            <div style={{ display: 'flex', gap: '5px', color: '#ffc107', marginBottom: '20px', fontSize: '1.3rem' }}>
              {'★'.repeat(review.rating)}{'☆'.repeat(5 - review.rating)}
            </div>
            <p style={{ color: '#555', fontSize: '1.05rem', lineHeight: '1.6', fontStyle: 'italic', marginBottom: '25px', minHeight: '80px' }}>"{review.comment}"</p>
            <div style={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center', borderTop: '1px solid #eaeaea', paddingTop: '20px' }}>
              <strong style={{ color: '#1b4332', fontSize: '1.1rem' }}>{review.authorName}</strong>
              <span style={{ color: '#999', fontSize: '0.85rem' }}>{new Date(review.createdAt).toLocaleDateString('uk-UA')}</span>
            </div>
            <div style={{ position: 'absolute', top: '20px', right: '20px', fontSize: '4rem', color: '#e0e0e0', opacity: '0.5', lineHeight: '1', fontFamily: 'serif' }}>"</div>
          </div>
        ))}
      </div>
    )}

    {/* Кнопка "Залишити відгук" */}
    <div style={{ textAlign: 'center' }}>
      <button
        onClick={() => setIsReviewModalOpen(true)}
```

```
            style={{ backgroundColor: '#fff', color: '#1b4332', border: '2px solid #1b4332', padding: '12px 30px', fontSize: '1.1rem', fontWeight: 'bold', borderRadius: '8px', cursor: 'pointer', transition: '0.2s' }}
            onMouseOver={(e) => { e.target.style.backgroundColor = '#1b4332'; e.target.style.color = '#fff'; }}
            onMouseOut={(e) => { e.target.style.backgroundColor = '#fff'; e.target.style.color = '#1b4332'; }}
          >
            Залишити свій відгук
          </button>
        </div>
      </div>
    </section>

    {/* Модальне вікно для створення відгуку */}
    {isReviewModalOpen && (
     <div style={{ position: 'fixed', top: 0, left: 0, right: 0, bottom: 0, backgroundColor: 'rgba(0,0,0,0.7)', display: 'flex', justifyContent: 'center', alignItems: 'center', zIndex: 1000, padding: '20px' }}>
       <div style={{ backgroundColor: '#fff', borderRadius: '16px', padding: '30px', maxWidth: '500px', width: '100%', position: 'relative' }}>
        <button onClick={() => setIsReviewModalOpen(false)} style={{ position: 'absolute', top: '15px', right: '15px', background: 'none', border: 'none', fontSize: '1.5rem', cursor: 'pointer', color: '#666' }}><FiX /></button>
        <h2 style={{ color: '#1b4332', marginBottom: '20px' }}>Поділіться враженнями</h2>
        <form onSubmit={submitReview}>
         <div style={{ marginBottom: '15px' }}>
          <label style={{ display: 'block', marginBottom: '5px', fontWeight: 'bold' }}>Ваше ім'я</label>
            <input required type="text" value={newReview.authorName} onChange={(e) => setNewReview({...newReview, authorName: e.target.value})} style={{ width: '100%', padding: '10px', borderRadius: '6px', border: '1px solid #ccc' }} placeholder="Напр. Олена та Максим" />
         </div>
         <div style={{ marginBottom: '15px' }}>
          <label style={{ display: 'block', marginBottom: '5px', fontWeight: 'bold' }}>Оцінка</label>
           <select value={newReview.rating} onChange={(e) => setNewReview({...newReview, rating: parseInt(e.target.value)})} style={{ width: '100%', padding: '10px', borderRadius: '6px', border: '1px solid #ccc' }}>
            <option value="5">⭐⭐⭐⭐⭐ Відмінно</option>
            <option value="4">⭐⭐⭐⭐ Добре</option>
            <option value="3">⭐⭐⭐ Нормально</option>
            <option value="2">⭐⭐ Погано</option>
            <option value="1">⭐ Жахливо</option>
           </select>
         </div>
         <div style={{ marginBottom: '20px' }}>
          <label style={{ display: 'block', marginBottom: '5px', fontWeight: 'bold' }}>Текст відгуку</label>
            <textarea required value={newReview.comment} onChange={(e) => setNewReview({...newReview, comment: e.target.value})} rows="4" style={{ width: '100%', padding: '10px', borderRadius: '6px', border: '1px solid #ccc' }} placeholder="Що вам найбільше сподобалося?"></textarea>
         </div>
        <button type="submit" disabled={isSubmittingReview} style={{ width: '100%', padding: '12px', backgroundColor: '#1b4332', color: '#fff', border: 'none', borderRadius: '8px', fontSize: '1.1rem', fontWeight: 'bold', cursor: 'pointer' }}>
          {isSubmittingReview ? 'Відправка...' : 'Відправити відгук'}
```

```
            </button>
          </form>
        </div>
      </div>
    )}

    {/* 8. БЛОК: КЕРУВАННЯ БРОНЮВАННЯМ */}
    <section style={{ backgroundColor: '#1b4332', padding: '70px 20px', color: 'white', textAlign: 'center' }}>
      <div style={{ maxWidth: '600px', margin: '0 auto' }}>
        <h2 style={{ color: '#ffc107', marginBottom: '15px', fontSize: '2.2rem' }}>Вже маєте бронювання?</h2>
        <p style={{ marginBottom: '35px', fontSize: '1.15rem', lineHeight: '1.6', opacity: '0.9' }}>Ви можете легко перевірити деталі свого відпочинку, переглянути суму до сплати або скасувати бронювання онлайн.</p>
        <Link to="/my-booking" style={{ display: 'inline-block', padding: '16px 35px', backgroundColor: '#ffc107', color: '#1b4332', textDecoration: 'none', fontWeight: 'bold', borderRadius: '8px', fontSize: '1.1rem' }}>Перевірити статус</Link>
      </div>
    </section>
  </>
 );
}

export default Home;
```

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: “Розробка веб-орієнтованої інформаційної системи бронювання номерів для готельного комплексу”

Обсяг пояснювальної записки: 98

Дата закінчення роботи 10 червня 2024р.

Підпис_____