Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Старченко Максима Павловича*
(ПІБ)

академічної групи *121-22ск-1*
(шифр)

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

освітньої програми *Інженерія програмного забезпечення*
(назва освітньої програми)

на тему: *Розробка RESTful вебдодатку для управління готельним комплексом та бронювання номерів*

Керівники	Прізвище, ініціали	Оцінка за шкалою		Підпис
		рейтинговою	інституційною	
кваліфікаційної роботи	*доц. Зіборов І.К.*			
розділів:				
спеціальний	*доц. Зіборов І.К.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Мартиненко А.А.*			

Дніпро
2025

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » _____ 2025 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *121-22ск-1* *Старченко М.П.*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка RESTful вебдодатку для управління готельним комплексом та бронювання номерів*

затверджена наказом ректора НТУ «ДП» від *05.05.2025* № *336-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***16.05.2025 р.***
Економічний	***Провести розрахунок трудомісткості розробки комплексних рішень, витрат на створення комплексних рішень й тривалості їх розробки***	***30.05.2025 р.***

Завдання видав _____ *Зіборов І.К.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Старченко М.П.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *20.01.2025 р.*

Термін подання кваліфікаційної роботи до ЕК: *13.06.2025 р.*

РЕФЕРАТ

Пояснювальна записка: 79 с., 36 рис., 3 дод., 24 джерел.

Об’єкт розробки: вебдодаток для управління готельним комплексом та бронювання номерів.

Мета кваліфікаційної роботи: розробка RESTful вебдодатку для управління готельним комплексом та бронювання номерів.

У вступі обґрунтовано необхідність впровадження автоматизованих веб-систем для управління готельним бізнесом, проаналізовано актуальність використання RESTful веб-додатків для оптимізації процесів бронювання номерів, обліку клієнтів та підвищення якості сервісу. Висвітлено переваги централізованого зберігання даних, інтеграції із зовнішніми сервісами та забезпечення інформаційної безпеки.

У першому розділі проаналізовано сучасний стан і проблематику цифровізації готельного бізнесу, проведено огляд існуючих систем управління готелями, їх можливостей, переваг та обмежень для малих і середніх закладів. Визначено основні завдання розробки та галузь застосування майбутнього програмного продукту.

У другому розділі описано проектування та розробку програмного продукту: розглянуто структуру, архітектуру, використані технології та мови програмування. Детально висвітлено реалізацію ключового функціоналу — управління клієнтами, номерами, бронюваннями, а також організацію взаємодії користувача з системою через зручний веб-інтерфейс. Окрему увагу приділено питанням безпеки, автентифікації та авторизації.

У третьому, економічному, розділі розраховано трудомісткість розробки комплексних рішень, витрати на створення та тривалість розробки проекту.

Практична значення полягає у створенні прототипу системи, яка може бути впроваджена в роботу середніх і малих готелів, а також слугувати основою для подальшого розширення функціональності.

Актуальність створення веб-системи для управління готелем визначається зростаючою потребою в автоматизації бізнес-процесів, підвищенні якості обслуговування клієнтів та забезпеченні конкурентоспроможності закладу на ринку послуг.

Список ключових слів: ГОТЕЛЬ, ВЕБДОДАТОК, JAVA, SPRING BOOT, THYMELEAF, POSTGRESQL, АВТЕНТИФІКАЦІЯ, БРОНЮВАННЯ, REST API, АВТОРИЗАЦІЯ.

ABSTRACT

Explanatory note: 79 p., 36 figs., 3 app., 24 sources.

Object of development: web application for hotel complex management and room booking.

Purpose of qualification work: development of a RESTful web application for hotel complex management and room booking.

The introduction substantiates the need to implement automated web systems for hotel business management, analyzes the relevance of using RESTful web applications to optimize room booking processes, customer accounting and improve service quality. Highlights the advantages of centralized data storage, integration with external services and ensuring information security.

The first section analyzes the current state and issues of digitalization of the hotel business, reviews existing hotel management systems, their capabilities, advantages and limitations for small and medium-sized establishments. The main development tasks and the field of application of the future software product are determined.

The second section describes the design and development of the software product: the structure, architecture, technologies and programming languages are considered. The implementation of the key functionality is covered in detail - managing customers, rooms, reservations, as well as organizing user interaction with the system through a convenient web interface. Special attention is paid to security, authentication and authorization issues.

The third, economic, section calculates the complexity of developing complex solutions, the costs of creating and the duration of project development.

The practical significance lies in creating a prototype of the system that can be implemented in the work of medium and small hotels, as well as serve as the basis for further expansion of functionality.

The relevance of creating a web system for hotel management is determined by the growing need for automating business processes, improving the quality of customer service and ensuring the competitiveness of the institution in the services market.

Keywords: HOTEL, WEB APPLICATION, JAVA, SPRING BOOT, THYMELEAF, POSTGRESQL, AUTHENTICATION, BOOKING, REST API, AUTHORIZATION.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface (інтерфейс прикладного програмування)

CRUD – Create, Read, Update, Delete (операції створення, читання, оновлення, видалення даних)

IDE – Integrated Development Environment (інтегроване середовище розробки)

JPA – Java Persistence API (інтерфейс для роботи з базами даних у Java, використовується в Spring Data JPA)

JWT – JSON Web Token (відкритий стандарт для безпечної передачі інформації у вигляді JSON-об’єкта)

MVC – Model-View-Controller (архітектурний шаблон «модель-представлення-контролер»)

RBAC – Role-Based Access Control (розмежування прав доступу на основі ролей)

REST – Representational State Transfer (архітектурний стиль взаємодії компонентів у мережі)

SQL – Structured Query Language (структурована мова запитів до бази даних)

ВСТУП

Розвиток сучасних інформаційних технологій сприяє підвищенню ефективності управлінських процесів у різних галузях економіки. Готельний бізнес, як одна з найконкурентніших сфер послуг, вимагає впровадження автоматизованих систем для оптимізації бронювання номерів, обліку клієнтів та ресурсів, а також підвищення якості обслуговування. RESTful вебдодатки, що ґрунтуються на архітектурному підході REST, дозволяють забезпечити надійний, масштабований і зручний у розгортанні сервіс для управління готелем.

Мета дипломного проєкту – розробити RESTful вебдодаток для управління готельним комплексом та бронювання номерів, який забезпечить:

– централізоване зберігання інформації про номери, бронювання і клієнтів;

– зручний інтерфейс для адміністраторів та клієнтів;

– інтеграцію з платіжними та зовнішніми сервісами;

– гарантії інформаційної безпеки та захист персональних даних.

Об’єкт дослідження: процеси управління готельним бізнесом і бронюванням номерів.

Предмет дослідження: методи та технології розробки RESTful вебдодатків для автоматизації готельного бізнесу.

Практична цінність роботи полягає у створенні прототипу системи, яка може бути впроваджена в роботу середніх і малих готелів, а також слугувати основою для подальшого розширення функціональності.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Загальні відомості з предметної галузі

У сучасних умовах готельний бізнес переживає інтенсивну цифровізацію, однак досить багато малих та середніх закладів досі використовують застарілі або фрагментовані програмні рішення для управління бронюванням і обліком клієнтів. Провідні ринкові продукти (наприклад, Opera PMS, Cloudbeds, Hotelogix) забезпечують широкий функціонал, але мають високу вартість впровадження, складні налаштування та обмежену гнучкість під специфічні бізнес-процеси клієнта.

Незважаючи на численні наукові та галузеві дослідження в сфері автоматизованих систем менеджменту готельного бізнесу, існують такі прогалини:

– технічні протиріччя між необхідністю високої масштабованості та легкістю розгортання для невеликих закладів;

– недостатній рівень підтримки мультимовності та мультивалютності у відкритих чи доступних за ціною рішеннях;

– обмежені можливості інтеграції з сучасними хмарними платіжними та crm-системами без суттєвих доопрацювань;

– відсутність у деяких системах механізмів докладного аудиту та гнучкого розмежування прав доступу, що ускладнює відповідність стандартам інформаційної безпеки.

Таким чином, незадоволені вимоги до програмних виробів у цій галузі включають:

– доступна вартість впровадження при збереженні основного функціоналу;

– можливість швидкого кастомізованого налаштування під конкретні процеси готелю;

– вбудована підтримка сучасних стандартів захисту даних та аудиту;

– проста інтеграція через RESTful API з зовнішніми сервісами.

Галузь застосування розроблюваної системи – управління готелями, хостелами та апартаментами, де основним об'єктом є бронювання тимчасового проживання клієнтів. Програмний виріб буде використовуватися як адміністратором закладу для ведення обліку номерного фонду, так і клієнтами для самостійного оформлення бронювання через веб-інтерфейс або мобільний додаток.

1.2. Призначення розробки та галузь застосування

Розроблюваний RESTful веб-додаток призначений для автоматизації управління готельним комплексом та процесу бронювання номерів. У термінології проєкту є декілька ключових понять.

RESTful API – архітектурний стиль веб-сервісів [1], заснований на концепції представлення ресурсів через уніфіковані інтерфейси. Кожен ендпоінт відповідає за конкретний ресурс, а операції (CRUD [2]) реалізуються за допомогою HTTP-методів: GET для отримання даних, POST для створення, PUT/PATCH для оновлення та DELETE для видалення.

JSON (JavaScript Object Notation) – легковаговий текстовий формат обміну даними, зручний для передачі структурованої інформації між клієнтом та сервером. Оскільки JSON читається людиною та легко парситься більшістю мов програмування, він став стандартом для RESTful сервісів [3].

JWT (JSON Web Token) – стандарт відкритого формату для безпечної передачі інформації між сторонами як JSON-об’єкт. Використовується для аутентифікації та авторизації користувачів у додатку: після успішного входу сервер генерує токен, який клієнт додає в заголовок кожного запиту як Bearer Token [4].

Адміністратор – роль користувача з розширеними правами в системі. Адміністратор може створювати, редагувати та видаляти ресурси (номери,

бронювання, акаунти клієнтів), отримувати детальні звіти про завантаженість та доходи, а також керувати налаштуваннями системи.

Клієнт – кінцевий користувач, який взаємодіє з веб-інтерфейсом або мобільним додатком для пошуку вільних номерів, здійснення бронювання та оплати проживання. Клієнт має обмежені права: переглядати доступні номери, створювати свої бронювання, переглядати історію замовлень та керувати особистим профілем.

Необхідність розробки пов’язана з такими факторами:

– відсутність у багатьох малих та середніх готелів гнучкого та доступного за вартістю програмного рішення;

– зростаючий попит клієнтів на можливість швидкого онлайн-бронювання та оплату через веб-інтерфейс;

– потреба у централізації даних про клієнтів, бронювання та платежі для поліпшення аналітики та звітності;

Галузі застосування системи:

– незалежні міські та курортні готелі, які потребують доступного рішення для управління бронюваннями;

– мережі хостелів та апартаментів, де необхідна централізація даних між різними локаціями;

– bed&breakfast (bnb) заклади та вілли, що прагнуть автоматизувати процес реєстрації гостей;

– туристичні агенції та онлайн-платформи, які можуть інтегруватися через rest api для бронювання готелів клієнтів.

1.3. Підстави для розробки

Відповідно до освітньої програми, згідно навчального плану та графіків навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу.

Тема роботи узгоджується з керівником проекту, випускаючою кафедрою, та затверджується з наказом ректора.

Таким чином, підставами для розробки (виконання кваліфікаційної роботи) є:

– освітня програма спеціальності 121 «Інженерія програмного забезпечення»;

– навчальний план та графік навчального процесу;

– наказ ректора Національного технічного університету «Дніпровська політехніка» №336-с від 05.05.2025;

– завдання на кваліфікаційну роботу на тему «Розробка RESTful веб-додаток для управління готельним комплексом та бронювання номерів».

1.4. Постановка завдання

Метою розробки системи є створення інструменту, що забезпечить ефективне та економічно доцільне управління процесами бронювання та обліку клієнтів у готельному комплексі. Досягнення мети передбачає реалізацію наступних завдань:

– забезпечення централізованого зберігання та обробки інформації про номерний фонд, бронювання, клієнтів та платежі;

– скорочення часу обробки нескладних операцій (створення бронювання, реєстрація гостей);

– підвищення точності аналітики завантаженості номерів та доходів готелю;

– мінімізацію витрат на впровадження та обслуговування пз за рахунок використання відкритих технологій.

Вихідні дані системи використовуються для оперативного управління графіком заїздів/виїздів, формування фінансових звітів, аналітики заповнюваності та планування маркетингових кампаній.

Вимоги до організації збору та контролю вхідної інформації:

– дані про бронювання вводяться через захищений веб-інтерфейс або rest арі із валідацією полів (дати, контактні дані);

– забезпечити контроль коректності даних шляхом перевірки наявності конфліктів бронювань для одного номера;

– можливість коригування даних адміністратором з логуванням змін (аудит).

Якщо при спробі створення бронювання виявлено конфлікт дат з існуючим бронюванням – операція відхиляється з кодом помилки 409.

Якщо термін дії JWT-токена минув – доступ до ресурсів обмежується до повторної аутентифікації.

Розподіл функцій між персоналом і технічними засобами:

– адміністратор, створення та коригування бронювань, управління акаунтами, генерація звітів;

– система, валідація даних, обробка запитів, зберігання та резервне копіювання інформації.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги функціональних характеристик

Функціональні характеристики розроблюваного додатку:

- реєстрація та аутентифікація користувачів (адміністратор, клієнт);
- створення, змінення та скасування бронювання;
- перегляд статусу номерів у реальному часі;
- генерація звітів по завантаженості та доходах;
- підтримка мультивалютності при оплаті.

1.5.2. Вимоги до інформаційної безпеки

Система повинна забезпечувати безперервну та відмовостійку роботу, використовуючи реплікацію бази даних та балансування навантаження для підтримки стабільного функціонування у разі збою окремих компонентів. Вхідна

інформація проходить суворий контроль: усі запити валідуються відповідно до визначеної JSON Schema, з перевіркою форматів даних (дата, валюта, контактна інформація) та підтвердженням прав доступу. Перш ніж надсилати клієнтові відповідь, система фільтрує конфіденційні поля, такі як паролі та внутрішні токени, і у випадку помилок повертає уніфіковану структуру повідомлення без витоку деталей реалізації.

Час відновлення мінімальної функціональності для прийому бронювань після критичної відмови не повинен перевищувати 15 хвилин (RTO). Для захисту від несанкціонованого доступу всі запити передаються через HTTPS/TLS, а аутентифікація та авторизація здійснюються з використанням JWT та ролей RBAC [5].

Захист системи від несанкціонованого копіювання та розповсюдження програмного забезпечення досягається обфускацією коду серверних компонентів, ліцензійною перевіркою та захистом ключів шифрування за допомогою менеджера секретів.

Цілісність даних гарантується впровадженням контрольних сум для критичних таблиць та механізмами контролю версій записів з аудитом змін. Щоденне автоматичне резервне копіювання (повні та інкрементні копії) забезпечує збереження даних протягом не менше ніж 30 днів з тестуванням відновлення щомісяця.

1.5.3. Вимоги до складу та параметрів технічних засобів

Мінімальні вимоги:

- серверна інфраструктура, мінімум 2 CPU, 4 ГБ ОЗП;
- дисковий простір, не менше 50 гб;
- операційна система, Linux (Ubuntu 20.04 LTS);
- для фронтенду, сучасний браузер із підтримкою ES6.

1.5.4. Вимоги до інформаційної та програмної сумісності

Вимоги до інформаційної та програмної сутності:

- підтримка RESTful стандартів (JSON, HTTP verbs);
- сумісність із SQL-сервером (PostgreSQL);
- використання OpenAPI (Swagger) для документування api;
- можливість масштабування через Docker/Kubernetes.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1 Функціональне призначення програми

Розроблена інформаційна система — це RESTful вебдодаток, створений для автоматизації основних процесів управління готельним комплексом. Вона дозволяє ефективно організувати роботу адміністрації, забезпечити зручність обслуговування клієнтів та оптимізувати бронювання номерів.

Функціональне призначення системи полягає в реалізації наступних можливостей:

– ведення структурованої бази даних клієнтів та готельних номерів;

– здійснення онлайн-бронювання номерів;

– управління персоналом готелю з розмежуванням доступу до функціоналу відповідно до ролей;

– збереження історії бронювань для подальшого аналізу й обслуговування постійних клієнтів;

Експлуатаційне призначення вебдодатку полягає в покращенні організації роботи готельного комплексу з точки зору кінцевого користувача та адміністрації. Впровадження інформаційної системи забезпечує автоматизацію основних бізнес-процесів, таких як бронювання номерів, облік клієнтів і керування персоналом, що дозволяє зменшити потребу в ручній праці та знизити ризик помилок.

Крім того, система сприяє підвищенню оперативності прийняття рішень, забезпечує доступ до актуальної інформації в реальному часі та створює передумови для впровадження нових сервісів. У результаті готельний комплекс підвищує якість обслуговування, оптимізує витрати та отримує можливість ефективно реагувати на зміну ринкових умов.

2.2 Опис застосованих математичних методів

У процесі створення вебдодатку для управління готельним комплексом не використовувались складні математичні методи, пов’язані з чисельним моделюванням, статистичним аналізом або оптимізацією. Проте в межах реалізації функціональних можливостей системи застосовано базові елементи прикладної математики, які забезпечують коректну роботу логіки та алгоритмів обробки даних.

Зокрема, при реалізації механізмів пошуку вільних номерів, фільтрації та сортування результатів за різними критеріями (тип, ціна, кількість місць тощо) використовуються логічні операції, що базуються на принципах математичної логіки.

Також при побудові запитів до бази даних враховується структура реляційних залежностей між сутностями, що відповідає формалізованим уявленням про множини та відношення між ними.

Окремі елементи алгоритмів пошуку та сортування застосовуються для формування списків бронювань, впорядкування номерів за ціною, зручністю або датою. Ці алгоритми є частиною загальних принципів побудови інформаційних систем і використовуються для підвищення продуктивності й зручності інтерфейсу.

Таким чином, хоча система не містить складних математичних обчислень, вона базується на фундаментальних математичних поняттях, що забезпечують її надійну й ефективну роботу.

2.3 Опис використаної архітектури та шаблонів проектування

У процесі розробки вебдодатку для управління готельним комплексом було застосовано архітектуру типу «клієнт-сервер», яка дозволяє розділити логіку роботи між серверною частиною та інтерфейсом користувача. Такий

підхід спрощує як підтримку системи, так і можливість її масштабування у майбутньому [7].

Серверна частина реалізована за допомогою Spring Boot [8] – сучасного фреймворку на базі мови програмування Java, який надає інструменти для швидкого створення RESTful вебсервісів. У реалізації додатку використано шаблон проєктування MVC (Model-View-Controller), що забезпечує чітке розділення обов'язків між компонентами системи: моделі відповідають за структуру та збереження даних, контролери обробляють запити та визначають логіку маршрутизації, а сервісний рівень виконує обробку даних і забезпечує бізнес-логіку.

Клієнтська частина вебдодатку створена у вигляді простого інтерфейсу, який використовує базові вебтехнології (HTML, CSS, JavaScript). Фронтенд взаємодіє з сервером через HTTP-запити до REST API, отримуючи дані у форматі JSON. Основна увага приділяється зручності користувача, доступності базової функціональності та коректній візуалізації інформації.

Щодо організації процесу розробки, то було обрано інкрементну модель життєвого циклу програмного забезпечення. Це дозволило реалізовувати проєкт поступово, зосереджуючись на ключових модулях і паралельно тестуючи їх роботу. Такий підхід сприяв підвищенню якості програмного продукту та забезпечив гнучкість при внесенні змін на будь-якому етапі.

2.4 Опис використаних технологій та мов програмування

У процесі розробки вебдодатку для управління готельним комплексом було застосовано сучасні перевірені технології, які забезпечують стабільність, масштабованість, безпеку та ефективність усіх складових системи. Кожна технологія була обрана на основі аналізу її можливостей, з урахуванням завдань, що стояли перед системою.

2.4.1 Java

Java – це об’єктно-орієнтована, статично типізована, компільована мова програмування загального призначення, яка була розроблена компанією Sun Microsystems у 1995 році (зараз підтримується Oracle). Основною ідеєю Java є принцип «Write Once, Run Anywhere» – програма, написана на Java, може виконуватись на будь-якій платформі, що підтримує JVM (Java Virtual Machine), без необхідності змін у коді [9].

Серед основних переваг Java можна відзначити високу надійність та стабільність, сувору типізацію, що дозволяє виявляти помилки ще на етапі компіляції, а також багатий набір бібліотек і фреймворків, які значно пришвидшують розробку. Мова добре підходить як для розробки великих корпоративних додатків, так і для побудови масштабованих мікросервісних архітектур. Java має розвинену багатопоточність, підтримку роботи з мережею, базами даних та розподіленими системами.

Особливістю Java є підтримка автоматичного збирання сміття (garbage collection), що звільняє розробника від необхідності вручну керувати пам’яттю, як у C або C++. Це значно знижує ймовірність помилок, пов’язаних із витоками пам’яті або подвійним звільненням ресурсів. Окрім того, Java має розвинену систему пакетів, які дозволяють структуровано організовувати великі проєкти, а також систему модулів (починаючи з Java 9), що покращує інкапсуляцію.

Java має багату екосистему. У її арсеналі – такі потужні фреймворки як Spring, Hibernate, Jakarta EE (колишній Java EE), а також сучасні інструменти для тестування, безперервної інтеграції, профілювання й моніторингу. Для побудови RESTful API, як у цьому проєкті, найчастіше використовується Spring Boot – інструмент, що дозволяє швидко створити продуктивний вебдодаток з мінімальними налаштуваннями.

Розробка у середовищі Java зазвичай ведеться з використанням таких IDE, як IntelliJ IDEA або Eclipse, які мають потужну підтримку аналізу коду, автодоповнення та налагодження.

Серед недоліків Java варто згадати порівняно високі вимоги до ресурсів у виконанні додатків, особливо у порівнянні з легшими мовами (наприклад, Go чи Node.js). Програми на Java мають більший розмір JAR-файлів та вимагають завантаження JVM, що може впливати на швидкість старту. Ще однією складністю є досить високий поріг входу у повноцінну екосистему Java, особливо для початківців. Робота з фреймворками Spring, Hibernate вимагає знання концепцій інверсії управління (IoC), впровадження залежностей (DI), анотаційної конфігурації, які не завжди інтуїтивно зрозумілі.

Java як мова орієнтована на об’єктно-орієнтоване програмування, хоча починаючи з Java 8 у ній з’явилися елементи функціонального стилю — лямбда-вирази, стріми, Optional. Стандартні структури типів добре підтримують розширення через інтерфейси, абстрактні класи та композицію, хоча немає алгебраїчних типів або pattern matching у тому вигляді, як у сучасних мовах (окрім часткової підтримки в Java 21).

У цьому проєкті Java була обрана як основна мова програмування серверної частини через її стабільність, широку підтримку, багатий набір фреймворків і можливість легко масштабувати рішення в майбутньому. Наявність попереднього досвіду роботи з Java та Spring Boot, а також зручні засоби інтеграції з базами даних, безпекою та шаблонізаторами, дозволили суттєво скоротити час на розробку і забезпечити високу якість результату.

2.4.2 Spring Boot

Spring Boot – це модуль фреймворку Spring, призначений для спрощення створення, налаштування та розгортання Java-додатків. Вперше представлений у 2014 році, Spring Boot став стандартом у світі розробки RESTful-сервісів та мікросервісів завдяки принципу «convention over configuration» – більшості необхідних конфігурацій уже задано за замовчуванням, що дозволяє розробнику зосередитись на бізнес-логіці.

Spring Boot використовує анотаційний підхід, що дозволяє суттєво зменшити обсяг шаблонного коду та спростити структуру застосунку. Ініціалізація додатка займає мінімум часу: вже з перших хвилин створення проєкту доступна структура з інтегрованим вебсервером (Tomcat, Jetty або Undertow), підтримкою REST-контролерів, системою логування, безпекою та взаємодією з базами даних. Це дозволяє створити робочий вебдодаток за лічені хвилини.

Серед переваг Spring Boot – глибока інтеграція з іншими проєктами Spring (Spring MVC, Spring Data, Spring Security), а також активна підтримка з боку спільноти та наявність великої кількості стартових модулів (starters), які автоматично підключають необхідні залежності. Spring Boot добре підходить для побудови як монолітних додатків, так і мікросервісної архітектури, завдяки підтримці конфігурацій на основі YAML або properties, гнучкості системи залежностей та сумісності з контейнеризацією (наприклад, Docker).

Недоліками Spring Boot можна вважати ускладнення при спробі глибокого кастомного налаштування або відходу від типових сценаріїв використання. У таких випадках необхідно добре орієнтуватись у внутрішній структурі Spring Framework – механізмах автоматичної конфігурації, життєвому циклі бінів, ієрархії контекстів тощо. Крім того, чимало магії «за лаштунками» ускладнює налагодження, особливо для початківців. Значний обсяг залежностей також може призвести до збільшення розміру JAR-файлу та складності при конфлікті версій бібліотек.

У цьому проєкті Spring Boot було обрано завдяки своїй здатності швидко створити готовий до запуску вебдодаток з мінімальними налаштуваннями, гнучкій структурі, сумісності з реляційними базами даних, інтеграції з Spring Security та підтримці шаблонізаторів типу Thymeleaf. Наявність вбудованого сервера значно спростила тестування й розгортання, а можливість швидкої інтеграції з іншими модулями Spring забезпечила масштабованість і розширюваність системи у майбутньому.

2.4.3 Spring Security

Spring Security – це потужний і гнучкий фреймворк для забезпечення безпеки Java-додатків, який є частиною екосистеми Spring [10]. Його основне призначення – контроль доступу, захист вебзастосунків від загроз і реалізація повного циклу аутентифікації та авторизації. Spring Security підтримує як прості механізми входу за логіном і паролем, так і складні сценарії – включно з OAuth2, JWT, LDAP, SAML та інтеграцією з іншими системами автентифікації.

Однією з головних переваг фреймворку є високий рівень гнучкості, що дозволяє адаптувати механізми безпеки під специфічні вимоги конкретного застосунку. Spring Security забезпечує деталізовану конфігурацію доступу до ресурсів, реалізацію власних фільтрів безпеки, механізмів сесій, ролей і прав, а також захист від типових вебзагроз, таких як CSRF (міжсайтові підробки запитів), XSS (міжсайтове виконання скриптів) і Session Fixation. Крім того, бібліотека має вбудовану підтримку шифрування паролів, обробки форм реєстрації, логіну та логауту.

Принципова архітектура Spring Security базується на ланцюжку фільтрів, які перехоплюють усі запити до серверу і, залежно від налаштувань, або допускають їх до виконання, або блокують з відповідним повідомленням. Також є можливість підключати власні провайдери аутентифікації, які дозволяють виконувати перевірку облікових даних за будь-якою логікою (наприклад, через зовнішні сервіси, базу даних або API).

Проте гнучкість і потужність Spring Security водночас є його слабким місцем. Для налаштування навіть базової конфігурації аутентифікації розробнику необхідно ознайомитися з великою кількістю понять: фільтрами безпеки, ланцюжком обробки запитів, менеджерами аутентифікації, обробниками подій та політиками авторизації.

У межах даного вебдодатку для готельного комплексу Spring Security було обрано як ключовий інструмент для розмежування прав доступу між звичайними користувачами та адміністраторами. Звичайні користувачі мають змогу

реєструватися, переглядати доступні номери та оформлювати бронювання, тоді як адміністратор отримує розширений інтерфейс керування, з можливістю редагування списку номерів, перегляду клієнтів та підтвердження бронювань.

2.4.4 Thymeleaf

Thymeleaf – це сучасний серверний шаблонізатор, розроблений для створення динамічних HTML-сторінок у середовищі Java, зокрема в поєднанні з фреймворком Spring Boot. Його основне завдання – забезпечити ефективну генерацію HTML-коду на основі даних, отриманих із серверної частини програми. Завдяки глибокій інтеграції з Spring MVC, Thymeleaf є зручним інструментом для побудови повноцінних вебінтерфейсів у Java-застосунках, де сервер формує HTML-сторінки і передає їх на клієнт [11].

Ключовою особливістю Thymeleaf є підтримка так званого «натурального шаблонізування». Це означає, що HTML-шаблони, які створюються розробником, залишаються валідними та доступними для перегляду у звичайному браузері навіть без попередньої обробки сервером. Такий підхід спрощує візуальну розробку інтерфейсу, дозволяє дизайнерам та розробникам ефективно співпрацювати, а також значно пришвидшує процес верстки.

Серед переваг Thymeleaf можна також виділити високу читабельність шаблонів, логічну структуру коду, відсутність необхідності у використанні додаткових мов (як, наприклад, у JSP), а також активну підтримку з боку спільноти Spring. Завдяки тому, що шаблони Thymeleaf є звичайними HTML-документами, вони легко піддаються редагуванню в будь-якому середовищі розробки або редакторі коду.

Однак незважаючи на зручність і простоту, Thymeleaf має певні обмеження. Він не призначений для створення високодинамічних інтерфейсів, які потребують реактивної взаємодії без перезавантаження сторінки, як це реалізується у клієнтських фреймворках типу React, Vue чи Angular.

2.4.5 PostgreSQL

Для зберігання даних вебдодатку використовується реляційна система управління базами даних PostgreSQL [12]. Вибір цієї СУБД обумовлений її високою надійністю, підтримкою складних запитів та дотриманням стандарту SQL, що дозволяє ефективно працювати зі структурованими даними і зв’язками між сутностями – користувачами, бронюваннями, номерами тощо.

Реляційна модель забезпечує цілісність даних завдяки нормалізації таблиць і використанню зовнішніх ключів. Це гарантує відсутність дублювань та узгодженість інформації в системі, що особливо важливо для коректної роботи функцій бронювання та управління користувачами. Крім того, підтримка транзакцій забезпечує надійність операцій, дозволяючи виконувати складні зміни в базі цілісно або відкотити їх у разі помилки.

Серед недоліків реляційного підходу варто відзначити необхідність попереднього планування структури бази та складність масштабування у горизонтальному напрямку. Проте для завдань управління готельним комплексом, де важлива точність і зв’язність даних, PostgreSQL є оптимальним вибором, що забезпечує стабільність і надійність роботи системи.

2.4.6 Додаткові інструменти розробки (Maven, Git)

Для управління залежностями та збірки проєкту у процесі розробки використовувався Maven – потужний інструмент, який дозволяє автоматизувати процес завантаження потрібних бібліотек, компіляції, тестування та створення кінцевого артефакту [13]. Завдяки Maven забезпечується уніфікований і передбачуваний процес збірки, що значно спрощує підтримку проєкту та інтеграцію нових компонентів. Однак Maven має власний синтаксис і структуру конфігураційних файлів, які вимагають певного часу для вивчення, особливо новачками.

Для контролю версій і організації командної роботи використовується система Git [14]. Вона дозволяє відслідковувати всі зміни в коді, працювати з різними гілками проєкту, об’єднувати та керувати кодом кількох розробників одночасно. Git є стандартом у сучасній розробці, що забезпечує гнучкість і безпеку у веденні проєкту.

2.5 Опис структури програми та алгоритмів її функціонування

Вебдодаток для управління готельним комплексом та бронювання номерів розроблено з використанням сучасних принципів архітектури корпоративних додатків, що забезпечує гнучкість, масштабованість і простоту супроводу. Система побудована з використанням фреймворку Spring Boot, що дозволяє організувати чітку багаторівневу структуру застосунку та ефективно розділити відповідальність між складовими.

Архітектура системи реалізує класичну трирівневу модель (three-tier architecture), що розділяє застосунок на рівень представлення, рівень бізнес-логіки та рівень доступу до даних. Такий підхід забезпечує слабкий зв’язок між компонентами, спрощує розширення функціоналу та підвищує безпеку.

На рівні представлення реалізовано веб-інтерфейс за допомогою технології Thymeleaf, що забезпечує інтерактивну взаємодію користувача із системою. Кожна веб-сторінка містить окремий шаблон, який отримує всі необхідні дані через контролери. Це дозволяє гнучко формувати інтерфейс залежно від ролі користувача (адміністратор чи клієнт) та забезпечує відокремлення логіки від представлення.

Рівень бізнес-логіки втілено у вигляді окремих сервісних класів (BookingService, CustomerService, RoomService, CustomUserDetailsService), які інкапсулюють основні функціональні процеси системи. Всі зміни стану (наприклад, створення бронювання, редагування даних клієнта, оновлення статусу номера) виконуються через ці сервіси, що гарантує цілісність транзакцій та дотримання бізнес-правил.

Рівень доступу до даних є абстрагованим від основної логіки за рахунок використання Spring Data JPA [15]. Репозиторії для основних сутностей (UserRepository, RoomRepository, BookingRepository, CustomerRepository) розширюють JpaRepository, що дозволяє реалізовувати CRUD-операції без написання SQL-коду і спрощує розробку.

Система реалізує низку алгоритмів, що забезпечують коректну роботу основних бізнес-процесів готелю.

Реєстрація користувача передбачає перевірку введеного пароля: якщо пароль співпадає зі спеціальним адміністраторським кодом, користувачу призначається роль ADMIN, інакше — роль USER. Перед збереженням у базі даних пароль хешується за допомогою BCrypt, що виключає можливість компрометації паролів у разі витоку даних. Після успішної реєстрації користувач автоматично перенаправляється на сторінку входу.

Авторизація та аутентифікація користувачів здійснюється засобами Spring Security. Доступ до різних частин системи обмежується відповідно до ролі користувача. Наприклад, адміністратор має повні права керування всіма клієнтами, номерами та бронюваннями, тоді як звичайний користувач може переглядати лише свої бронювання і здійснювати нові замовлення. Усі сесії захищені, після виходу з системи відбувається інвалідація сесії та видалення cookie.

Бронювання номера – ключовий алгоритм, який включає перевірку наявності вільних кімнат на вибрані дати, створення нового клієнта (за потреби), формування об’єкта Booking, асоціацію його з відповідними сутностями Room і Customer, а також виставлення статусу бронювання. При створенні бронювання адміністратор або клієнт обирає необхідні параметри: клієнта, номер, дати заїзду та виїзду, а також статус замовлення. Система автоматично визначає доступність кімнат через відповідний сервіс, що унеможливлює дублювання бронювань на одні й ті самі дати.

Керування клієнтами та номерами реалізовано через інтерфейс адміністратора. Можливе додавання, редагування чи видалення записів. При

створенні або редагуванні кімнати адміністратор вказує номер, тип, вартість за ніч та доступність для бронювання. Усі операції виконуються транзакційно, що гарантує відсутність «битих» записів у базі даних.

Обробка помилок у системі реалізована через окремий контролер CustomErrorController. Він перехоплює всі нештатні ситуації (наприклад, помилки 403, 404, 500) і відображає користувачу дружні повідомлення з описом проблеми та можливими шляхами вирішення. Це підвищує юзабіліті та зменшує навантаження на службу підтримки.

Основою для зберігання інформації в системі є реляційна база даних PostgreSQL. Структура бази даних оптимізована для швидкого пошуку й обробки даних, а також гарантує цілісність зв'язків між сутностями.

Таблиця users містить інформацію про користувачів: унікальний email, хешований пароль та роль (user або admin).

Таблиця customers призначена для зберігання даних клієнтів: імені, електронної пошти та номера телефону.

Таблиця rooms відображає кімнати готелю із зазначенням номера, типу (наприклад, одномісна, двомісна, люкс), ціни за ніч та статусу доступності.

Таблиця bookings містить інформацію про бронювання: посилання на клієнта та кімнату, дати заїзду/виїзду, а також статус бронювання (наприклад, confirmed, pending, cancelled).

Зв'язки між таблицями реалізовані за допомогою зовнішніх ключів: кожне бронювання містить посилання на конкретного клієнта і на конкретний номер. Це дозволяє легко отримувати всі бронювання певного клієнта чи всі бронювання певної кімнати, а також забезпечує цілісність даних.

Використання JPA-реалізацій дозволяє працювати з об'єктною моделлю напряму, не турбуючись про написання SQL-запитів, що значно спрощує розробку та супровід системи.

Для забезпечення можливості взаємодії з іншими програмними продуктами, Hotel Management API містить повноцінний REST API. Через цей інтерфейс сторонні системи можуть здійснювати операції з клієнтами, номерами

та бронюваннями – переглядати списки, отримувати детальну інформацію, створювати нові записи чи редагувати існуючі. Це відкриває перспективи для інтеграції із зовнішніми порталами, мобільними застосунками чи іншими корпоративними рішеннями.

Всі реалізовані алгоритми оптимізовані для забезпечення високої продуктивності, захисту персональних даних та уникнення дублювання записів. Використання транзакційності, хешування паролів, суворої валідації введених даних та розмежування прав доступу відповідає сучасним вимогам до інформаційної безпеки.

Реалізована логіка дозволяє гнучко керувати ролями користувачів, розподіляти бізнес-операції між сервісами та контролерами, а також гарантує простоту супроводу та тестування. Усі компоненти системи чітко взаємодіють за допомогою інтерфейсів, що дозволяє швидко змінювати або розширювати функціонал без ризику порушення роботи всієї програми.

2.6 Обґрунтування та організація вхідних та вихідних даних програми

Вхідні дані надходять у вигляді інформації, яку вводить користувач через веб-інтерфейс (форми реєстрації, авторизації, бронювання номерів, додавання клієнтів і кімнат). Попередня підготовка вхідних даних забезпечується валідацією на стороні клієнта (HTML5) та серверною валідацією (за допомогою анотацій javax.validation у моделях). Це дозволяє уникнути введення некоректних чи неповних даних і гарантувати цілісність інформації при подальшому збереженні у базі даних. Основні вхідні поля включають текстові (ПІБ, email, статус), числові (ціна), дати (дата заїзду/виїзду), а також булеві (доступність номера).

Вихідні дані організовано у вигляді таблиць, списків або детальних записів, що відображаються у веб-інтерфейсі для різних категорій користувачів. Для адміністратора це списки клієнтів, бронювань, номерів з можливістю

фільтрації та пошуку. Для клієнта – власні бронювання, поточний статус замовлення, підтвердження або відмова у наданні послуг.

Передача вхідних та вихідних даних між клієнтською частиною та сервером відбувається у форматі JSON (для REST API) або у вигляді параметрів HTTP-запитів (для класичних форм). Всі дані кодуються у стандарті UTF-8, що забезпечує підтримку української та інших мов. Формат даних суворо регламентований моделями сутностей (User, Customer, Room, Booking).

2.7 Опис розробленого програмного продукту

2.7.1 Використані технічні засоби

Серверна частина системи розгорталася на ноутбуці з такими характеристиками:

- процесор, i5-12500H;
- графічний адаптер, NVIDIA GeForce RTX 3060;
- оперативна пам’ять, 16 ГБ DDR4;
- монітор з роздільною здатністю 1920x1080 пікселів;
- клавіатура та миша.

Для клієнтської роботи з веб-інтерфейсом достатньо стандартного персонального комп’ютера або ноутбука з сучасним браузером (Google Chrome, Mozilla Firefox, Microsoft Edge чи Safari), підключенням до локальної мережі або Інтернету. Необхідність у спеціалізованих периферійних пристроях відсутня.

2.7.2 Використані програмні засоби

Під час розробки було використано такі програмні засоби:

- операційна система, Windows 10/11;
- мова програмування, Java 17;

– система управління базами даних, PostgreSQL;
– збірка проєкту, Maven;
– інтегроване середовище розробки (IDE), IntelliJ IDEA;
– фреймворк серверної частини, Spring Boot;
– автентифікація та авторизація, Spring Security;
– для роботи з БД, Spring Data JPA;
– шаблонізатор, Thymeleaf;
– бібліотека для генерації допоміжного коду (анотації), Lombok [16];
– бібліотека для роботи з JWT-токенами, JJWT;
– засоби для валідації даних, jakarta.validation;
– веб-браузер, Google Chrome.

2.7.3 Виклик та завантаження програми

Виклик серверної частини системи здійснюється через запуск виконуваного JAR-файлу, зібраного за допомогою Maven. Для цього необхідно переконатися, що на комп'ютері встановлено Java 17 або вище, а також наявна база даних PostgreSQL. JAR-файл розташовується у робочій директорії користувача.

Запуск програми виконується командою в терміналі:

java -jar hotel-management-api-1.1.jar

Після запуску програма автоматично ініціалізує сервер, підключається до бази даних (налаштування у файлі application.properties) та розпочинає обробку запитів на стандартному порту 8080.

Клієнтська частина не потребує встановлення: доступ до інтерфейсу здійснюється через сучасний браузер (Google Chrome, Mozilla Firefox або інші) за адресою, наприклад, http://localhost:8080/home, або через IP-адресу сервера у локальній мережі.

Для коректної роботи програми необхідно:

– встановлена Java 17+;

– працююча служба PostgreSQL, створена база даних з відповідними параметрами (див. application.properties);

– не менше 2 ГБ вільної оперативної пам’яті;

– близько 200 МБ місця на диску для програми та бази даних.

Обсяг JAR-файлу складає приблизно 50 МБ. Після запуску програма використовує близько 150-250 МБ оперативної пам’яті залежно від кількості одночасних користувачів.

2.7.4 Опис інтерфейсу користувача

При першому вході на вебсайт готельної системи користувач бачить мінімалістичну головну сторінку з привітанням і стислим поясненням можливостей сервісу (рис.2.1).

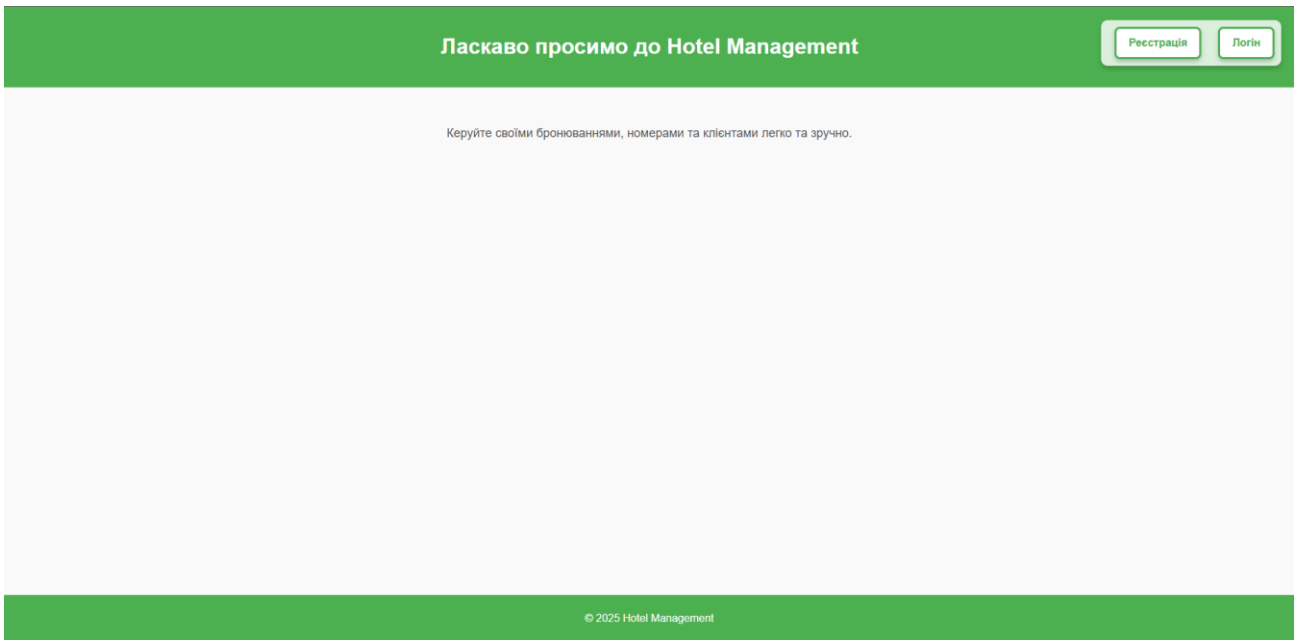

Рис. 2.1. Головна сторінка

У верхньому правому куті сторінки розташовані кнопки для логіну та реєстрації (рис 2.2).

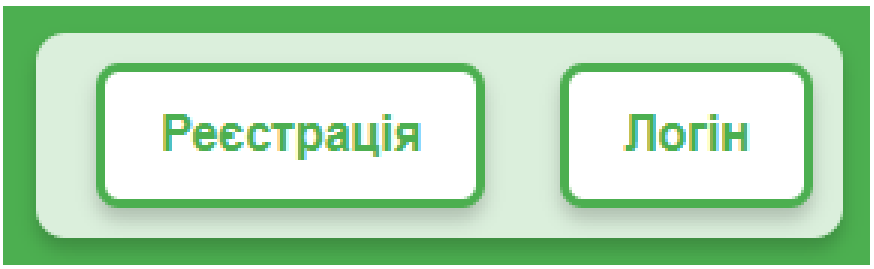

Рис. 2.2. Кнопки переходу на сторінки рєстрації та логіну

Без проходження авторизації клікабельні картки з переходом на функціонал недоступні, а якщо спробувати скористатися ними напряму (наприклад, перейти за посиланням на список бронювань), система перенаправить на сторінку входу (рис 2.3).

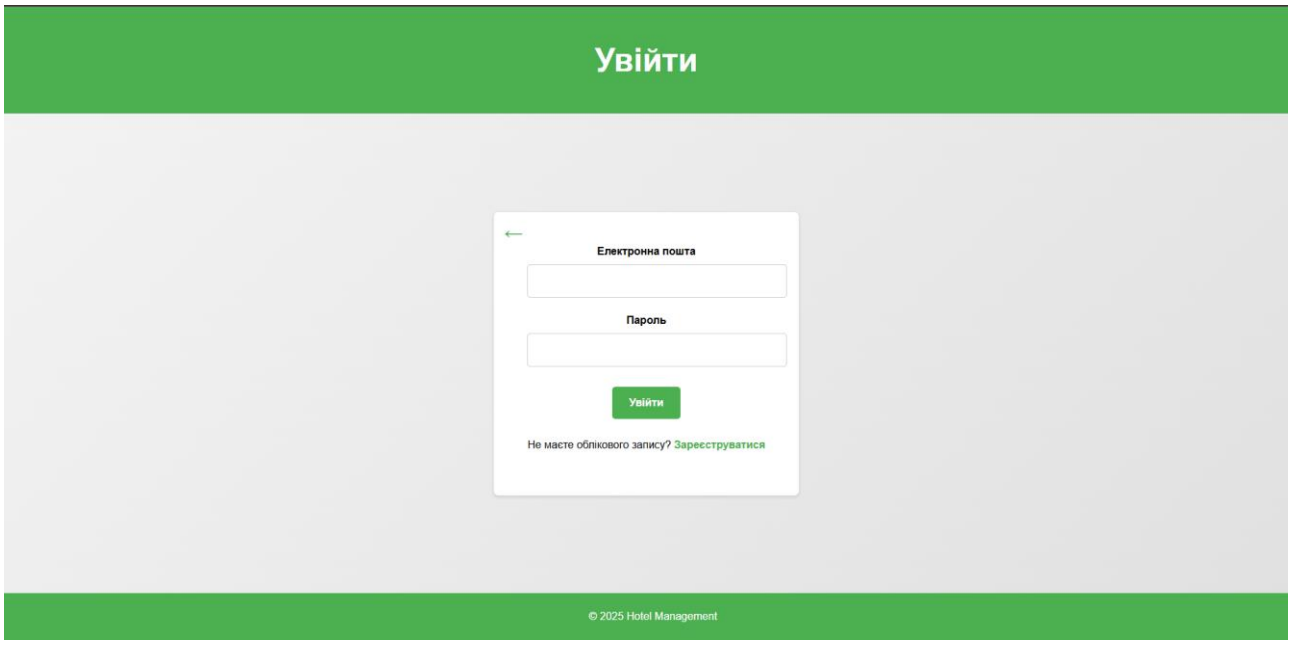

Рис. 2.3. Сторінка для входу

Після натискання на кнопку «Реєстрація» відкривається проста форма для введення електронної пошти та пароля (рис. 2.4).

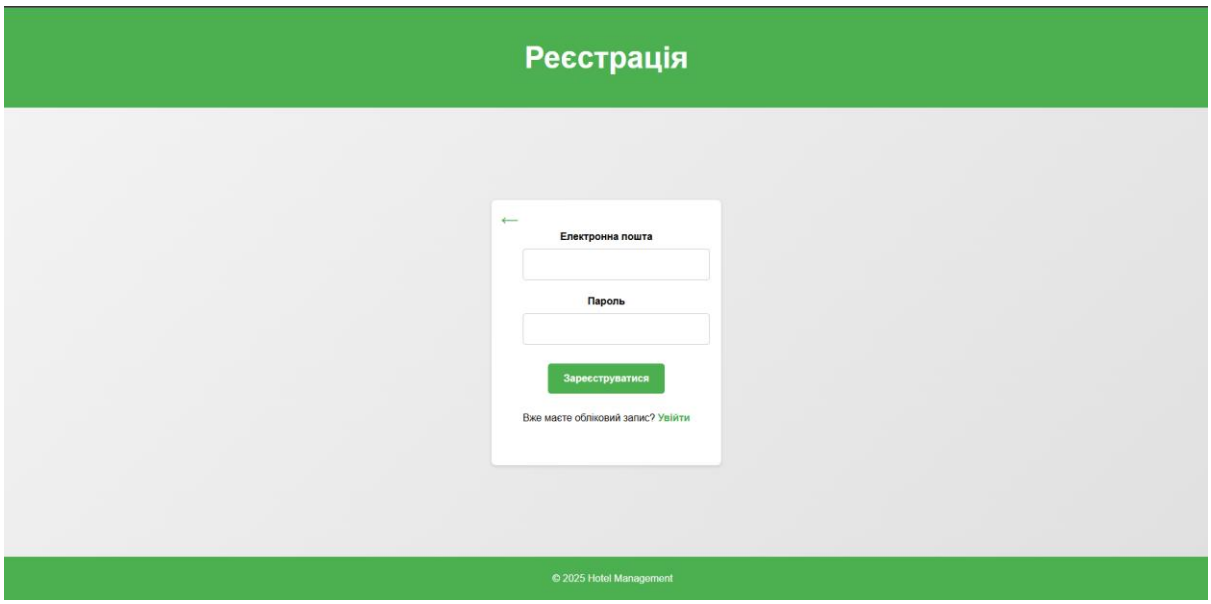

Рис. 2.4. Сторінка для реєстрації

Вхід відбувається через таку ж просту форму, після чого користувач повертається на головну сторінку, але тепер інтерфейс змінюється згідно з його роллю. Якщо користувач вводить спеціальний адміністративний пароль, то його обліковий запис отримує роль адміністратора, інакше – звичайного користувача (рис 2.5-2.8).

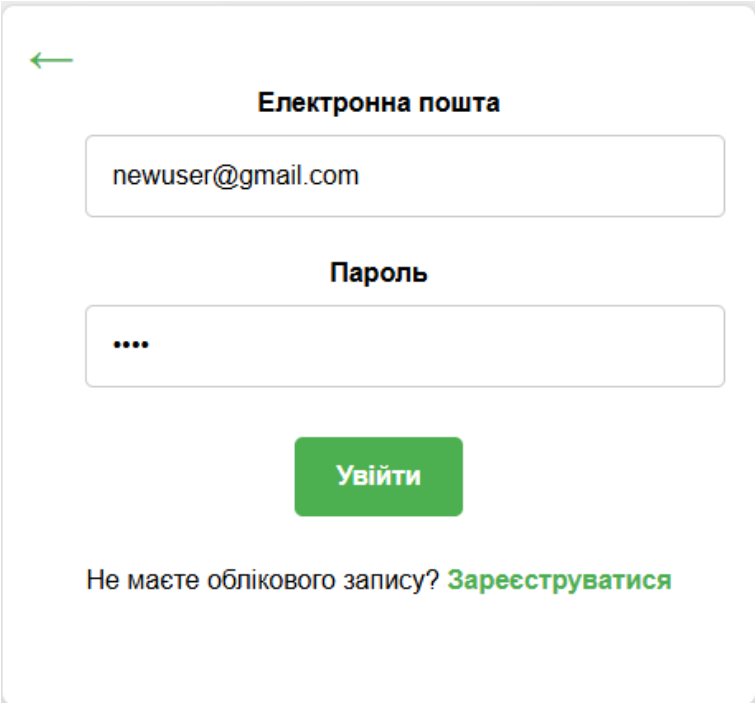

Рис. 2.5. Вхід під правами user

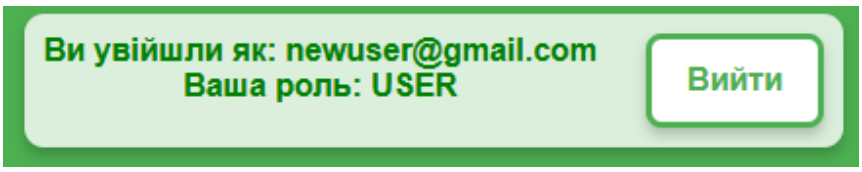

Рис. 2.6. Повідомлення для користувача з правами user

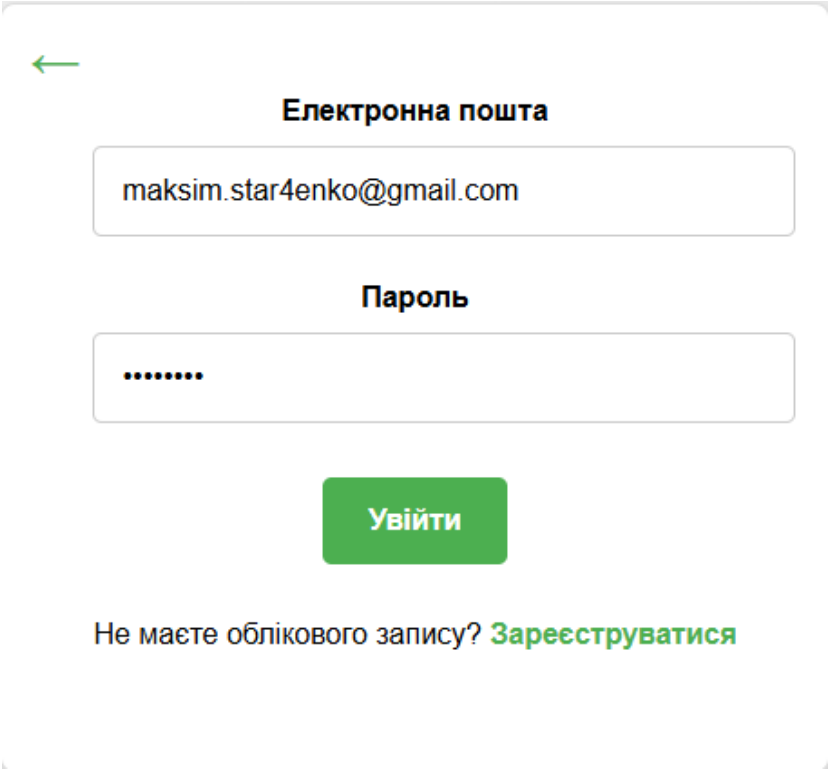

Рис. 2.7. Вхід під правами admin

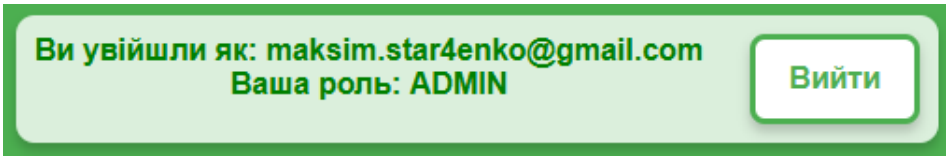

Рис. 2.8. Повідомлення для користувача з правами admin

Для адміністратора на головній сторінці стають доступними три основні розділи: керування клієнтами, номерами та бронюваннями (рис. 2.9).

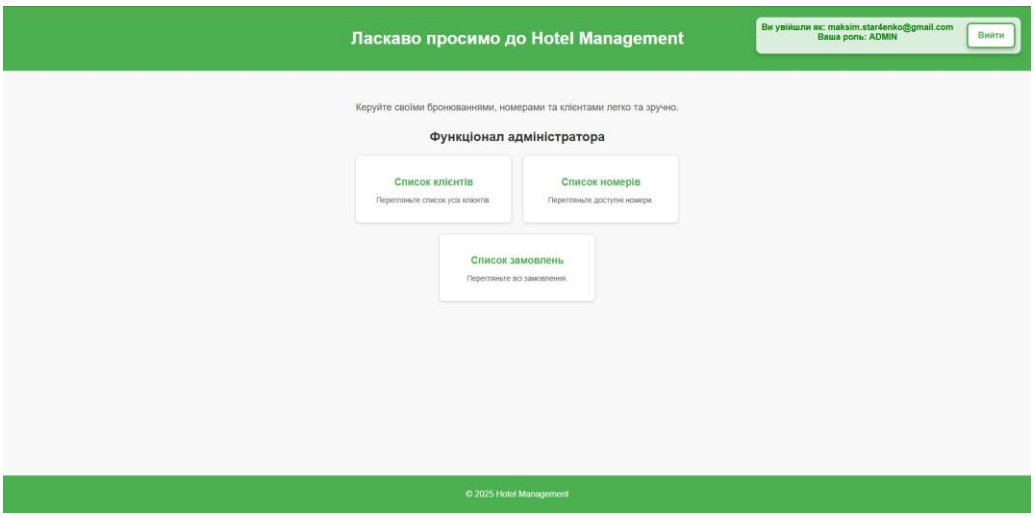

Рис. 2.9. Головна сторінка для адміністратора

Кожний розділ відкривається у вигляді таблиці з відповідною інформацією та кнопками для додавання, редагування чи видалення записів.

Наприклад, розділ «Клієнти» дозволяє переглядати список усіх гостей, їхні контакти, додавати нових, змінювати або видаляти дані (рис 2.10-2.15).

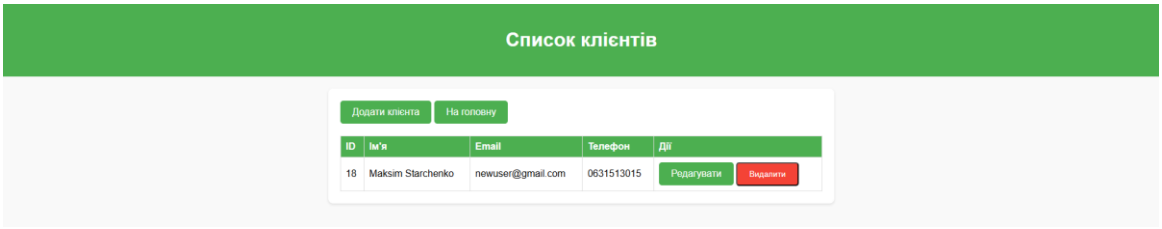

Рис. 2.10. Сторінка перегляду даних про клієнтів

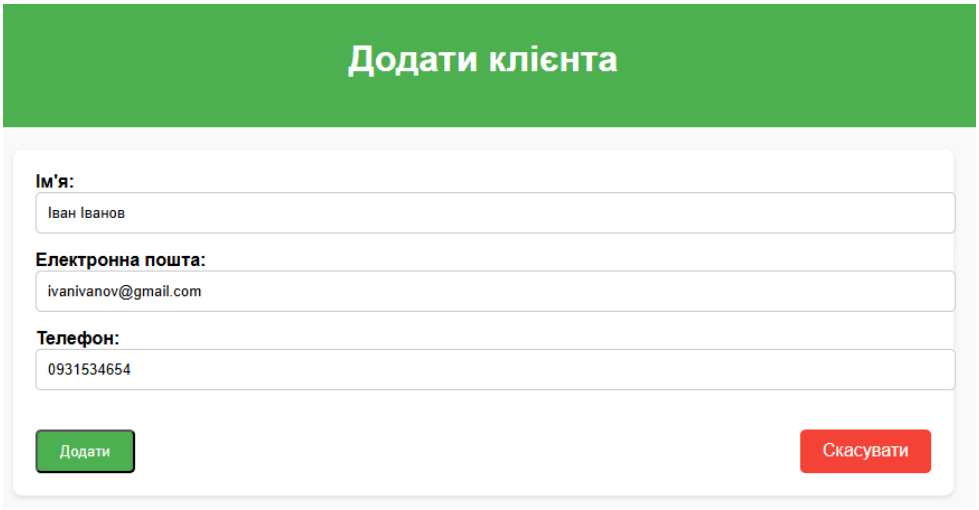

Рис. 2.11. Спроба додати дані про клієнта

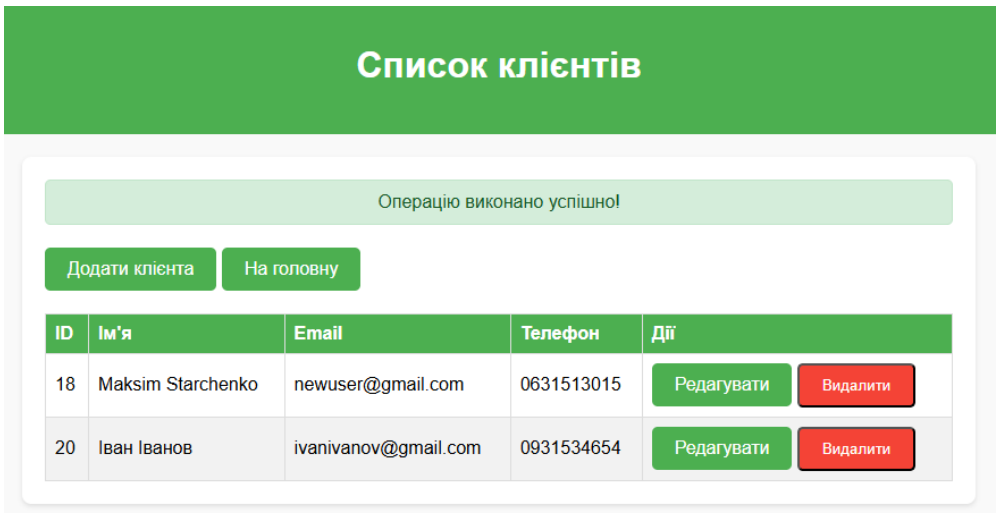

Рис. 2.12. Успішне додавання даних про клієнта

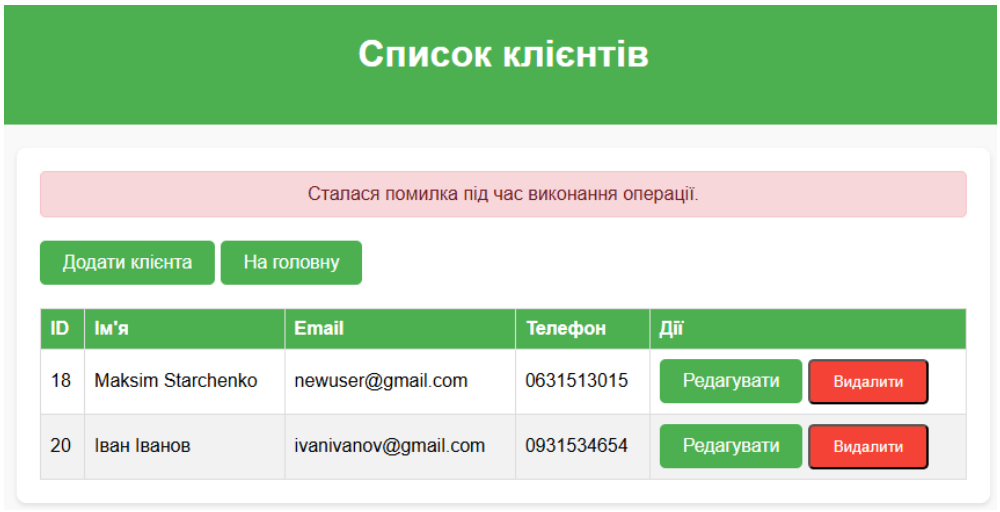

Рис. 2.13. Спроба видалення клієнта, який є активним

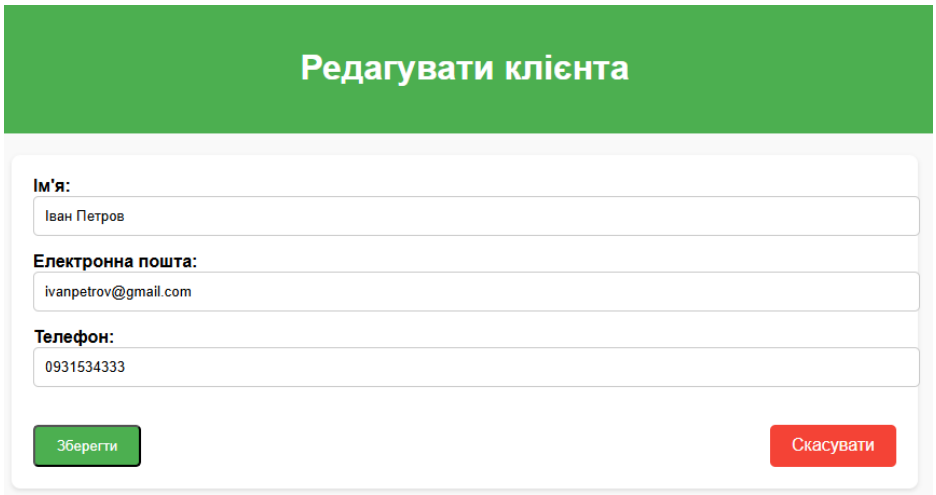

Рис. 2.14. Спроба редагування клієнта

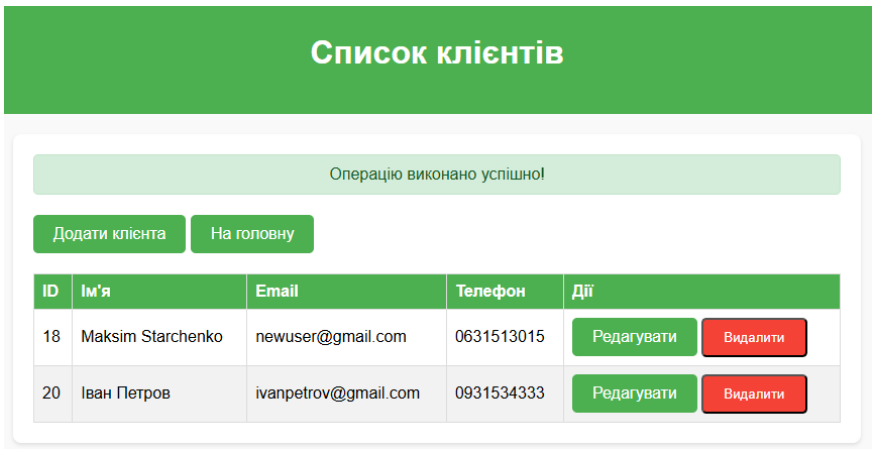

Рис. 2.15. Успішне редагування даних про клієнта

Аналогічно працюють сторінки номерів і бронювань – адміністратор може бачити всі замовлення, додавати нові бронювання, редагувати їх або переглядати деталі кожного бронювання на окремій сторінці (рис. 2.16-2.28).

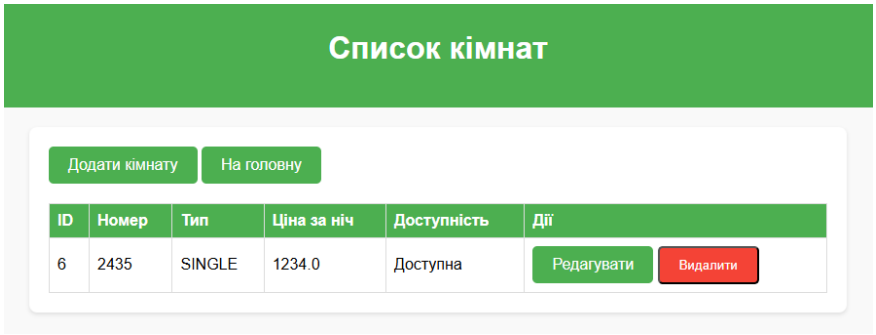

Рис. 2.16. Сторінка перегляду номерів

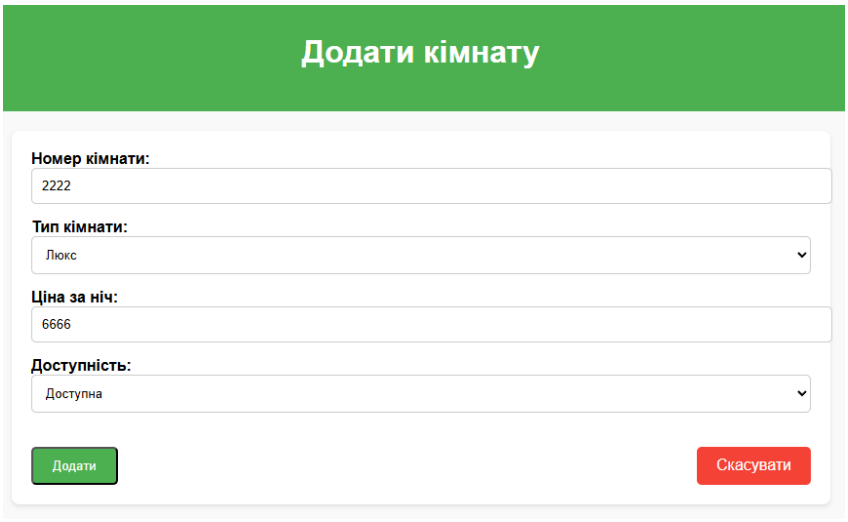

Рис. 2.17. Спроба додати номер

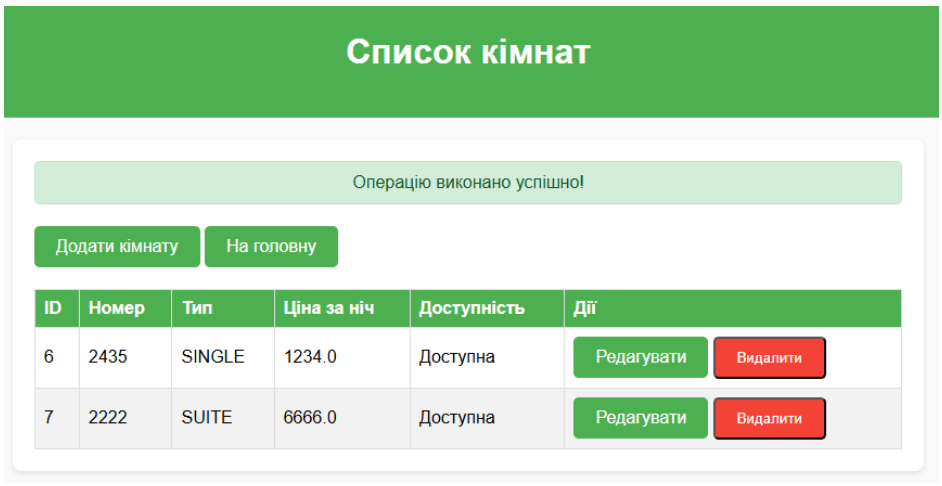

Рис. 2.18. Успішне додавання номеру

Рис. 2.19. Спроба видалення номеру, який є активним

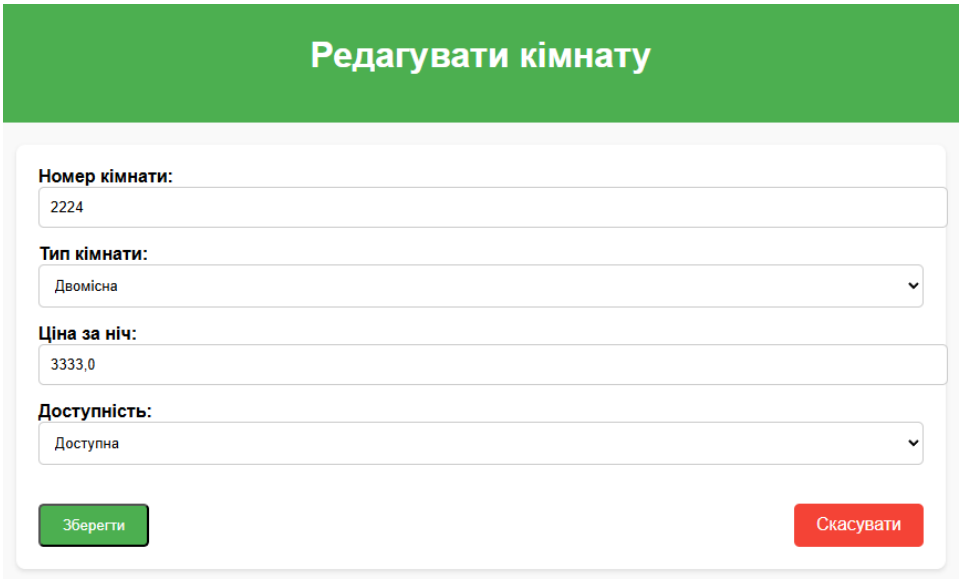

Рис. 2.20. Спроба редагування номеру

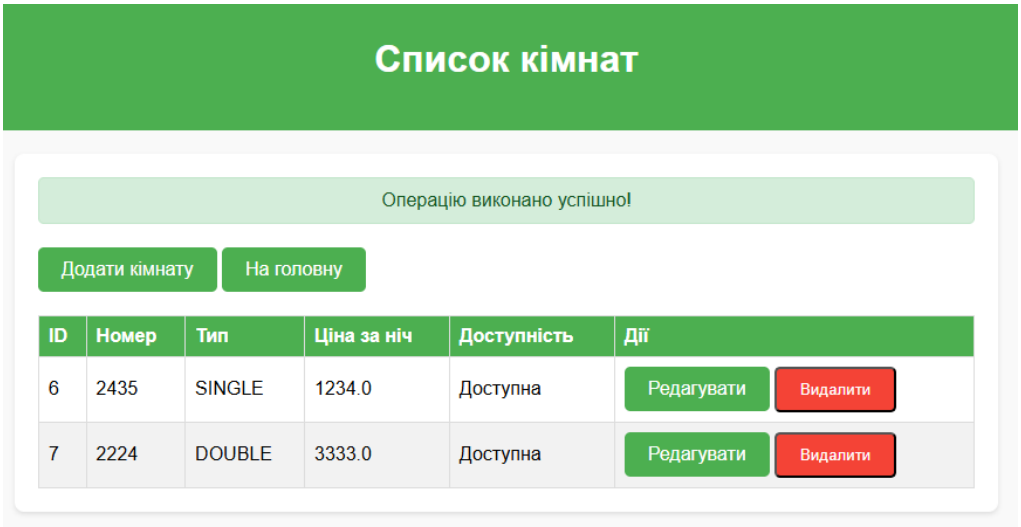

Рис. 2.21. Успішне редагування номеру

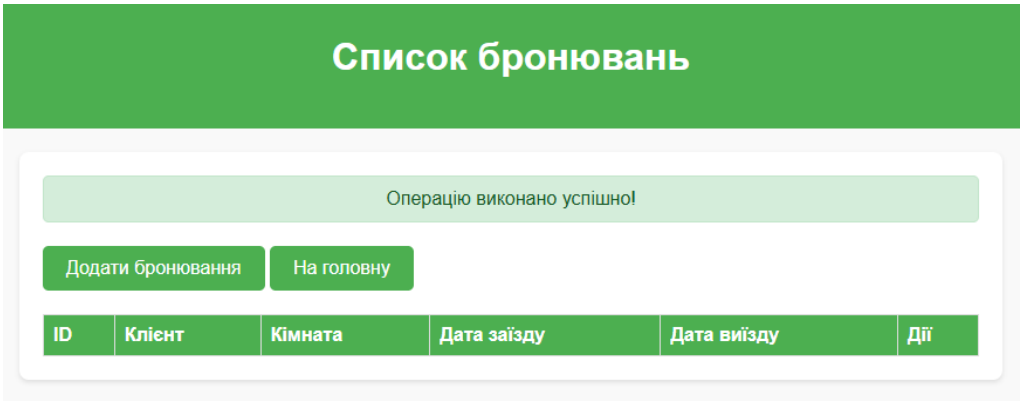

Рис. 2.22. Перегляд даних про бронювання

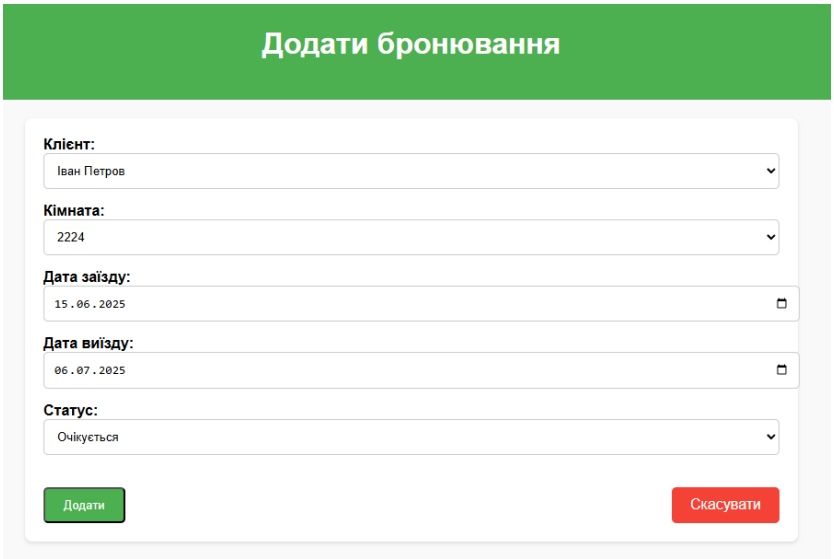

Рис. 2.23. Спроба додати бронювання

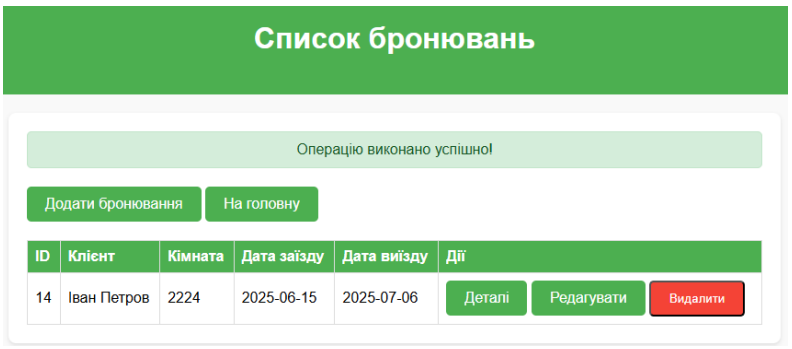

Рис. 2.24. Успішне додавання даних про бронювання

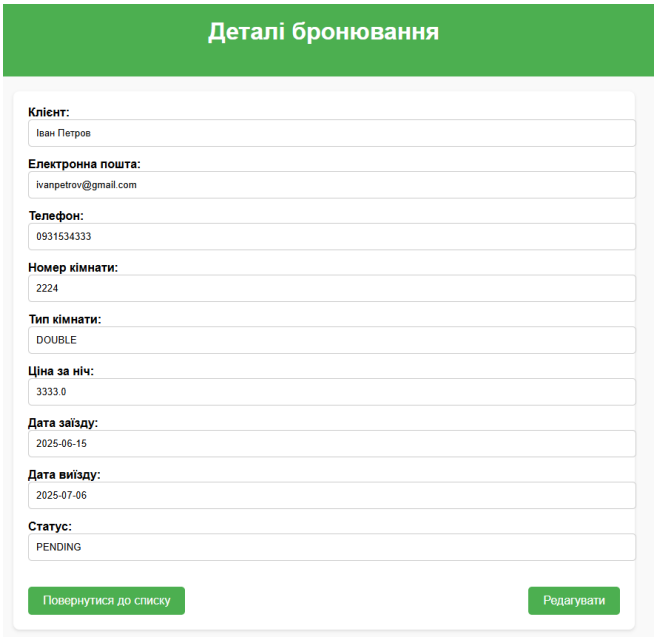

Рис. 2.25. Перегляд деталей бронювання

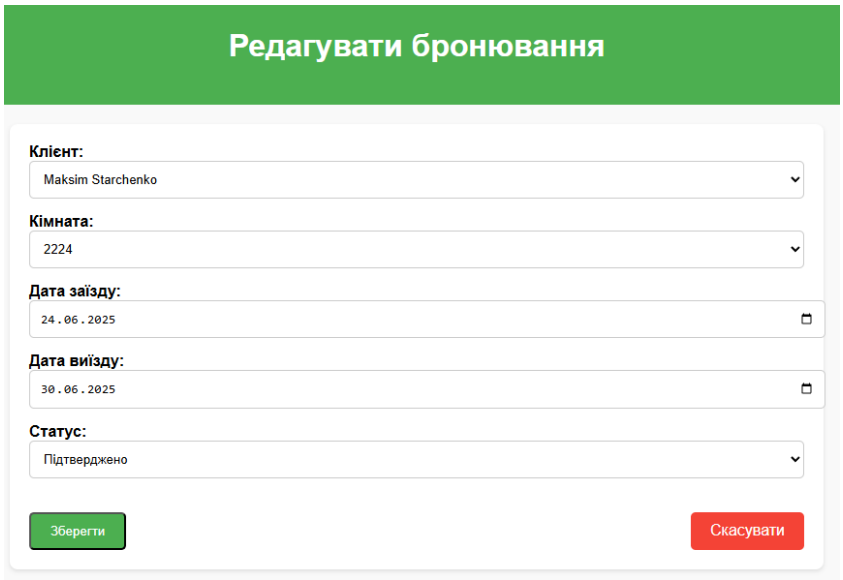

Рис. 2.26. Спроба редагування бронювання

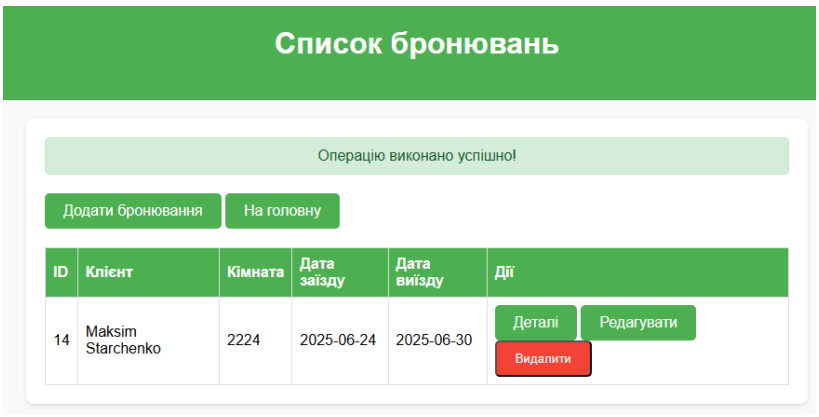

Рис. 2.27. Успішне редагування даних про бронювання

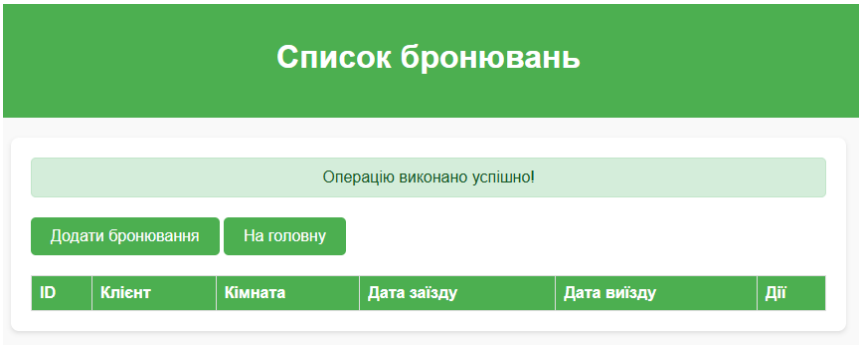

Рис. 2.28. Успішне видалення бронювання

Для звичайного користувача (клієнта) інтерфейс інший. На головній сторінці стають доступними лише дві основні опції: зробити бронювання та переглянути власні замовлення (рис. 2.29).

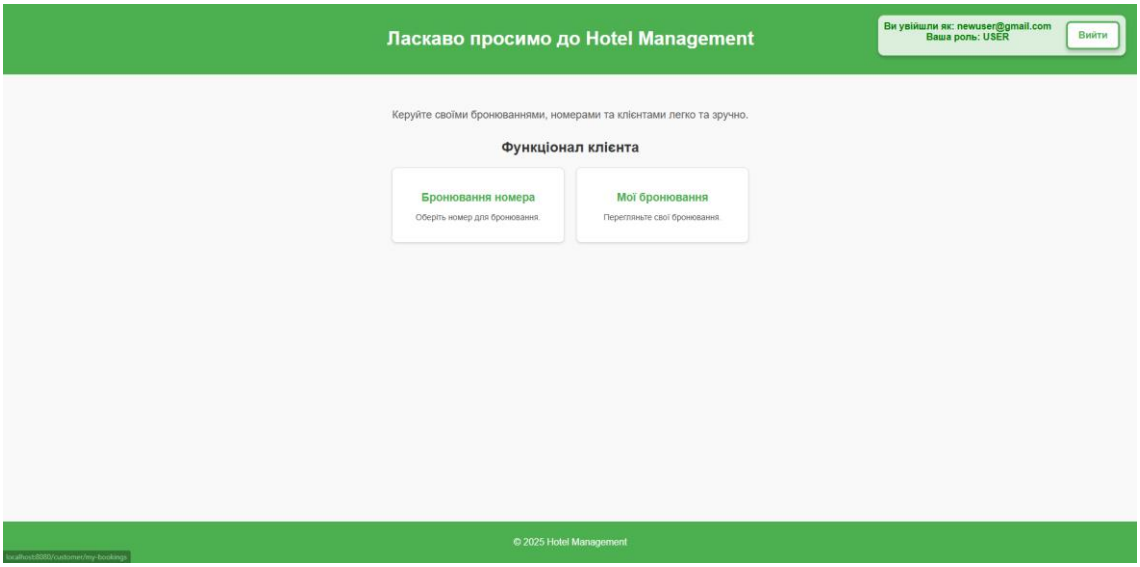

Рис. 2.29. Головна сторінка для користувача (клієнта)

Форма бронювання містить поля для вибору кімнати, введення контактної інформації, а також дати заїзду та виїзду (рис. 2.30).

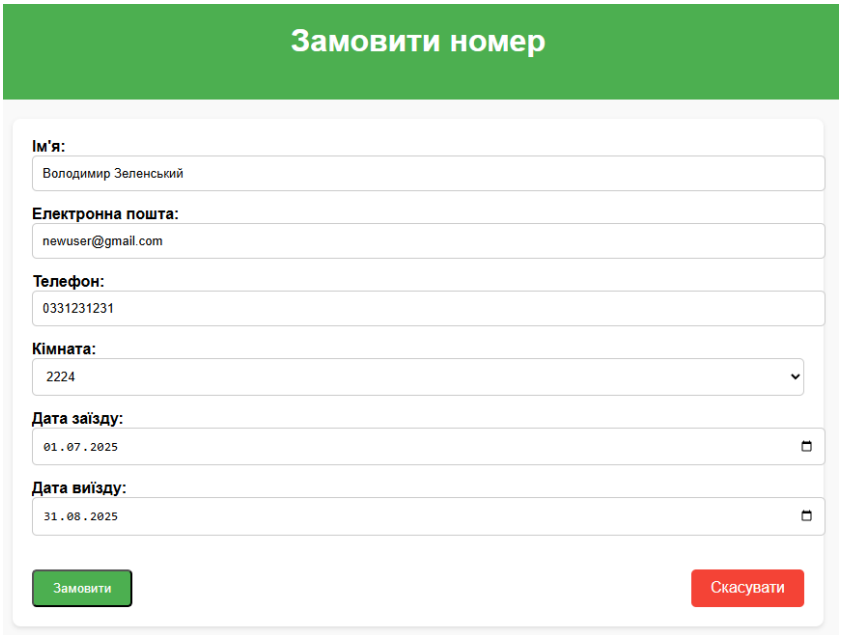

Рис. 2.30. Бронювання номера зі сторони клієнта

Після створення бронювання користувач автоматично переходить до списку своїх бронювань, де може переглядати всі власні замовлення та деталі кожного з них (рис. 2.31-2.32).

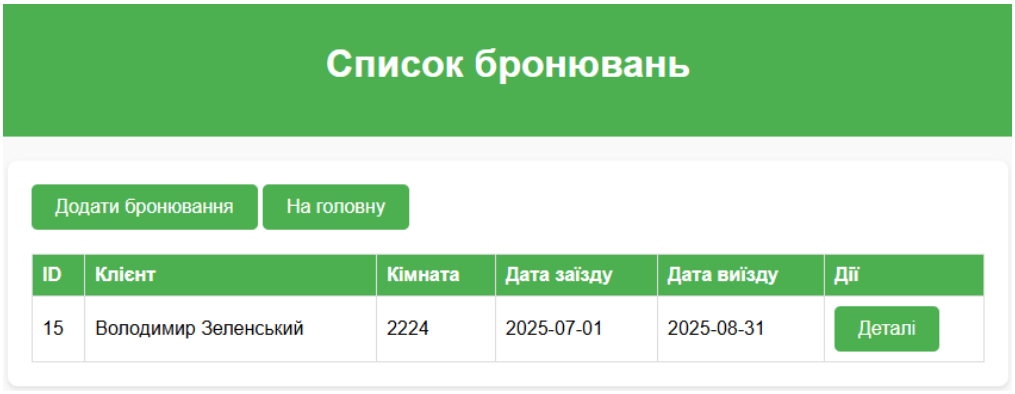

Рис. 2.31. Список бронювань для користувача (клієнта)

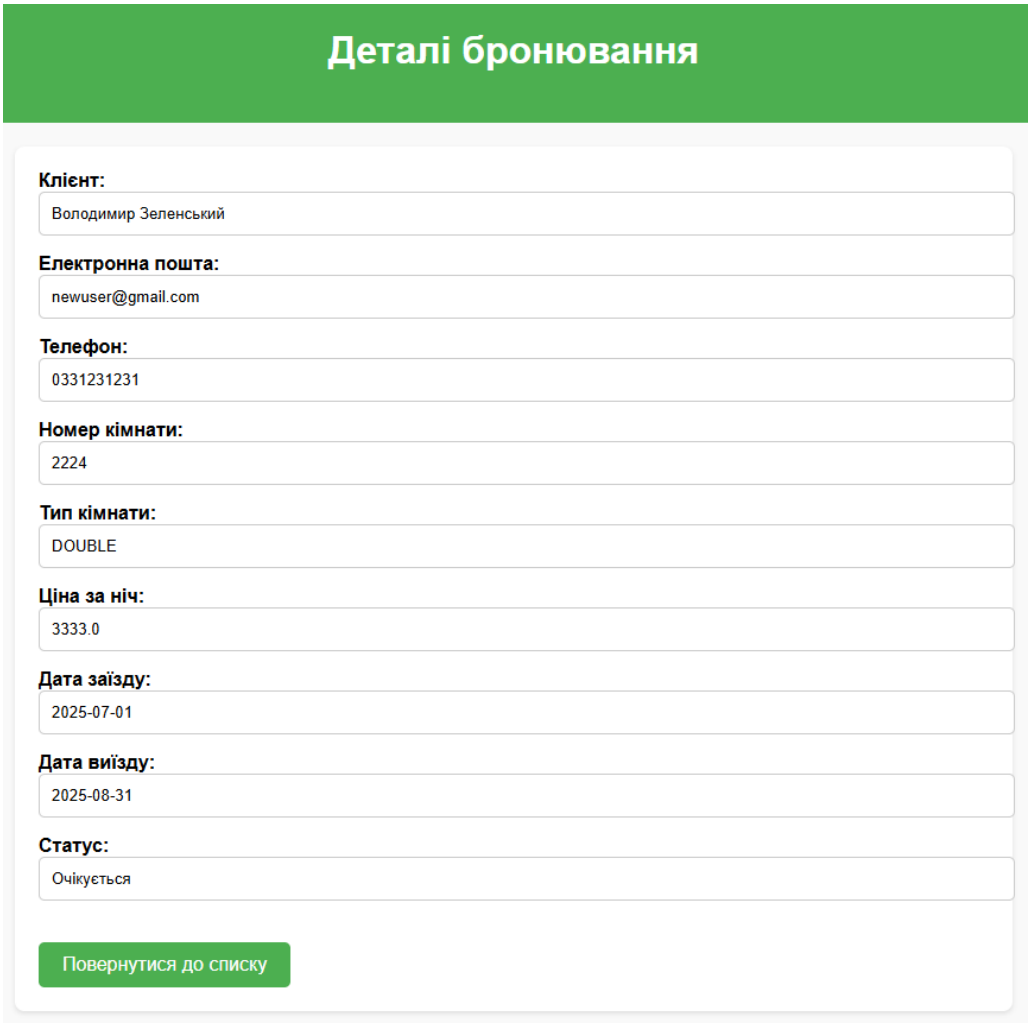

Рис. 2.31. Деталі бронювання користувача (клієнта)

Усі дії супроводжуються повідомленнями: у разі успіху з’являється підтвердження, у разі помилки – відповідне попередження (рис. 2.32-2.33).

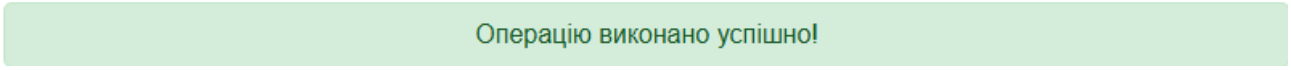

Рис. 2.32. Повідомлення про успішність операції

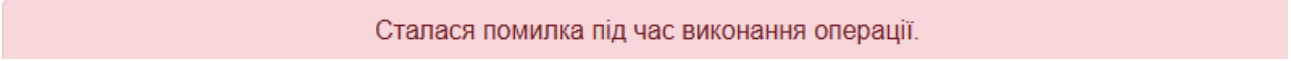

Рис. 2.33. Повідомлення про те, що сталась помилка

Якщо користувач намагається отримати доступ до сторінки, яка не відповідає його ролі, система показує повідомлення про помилку (рис 2.34).

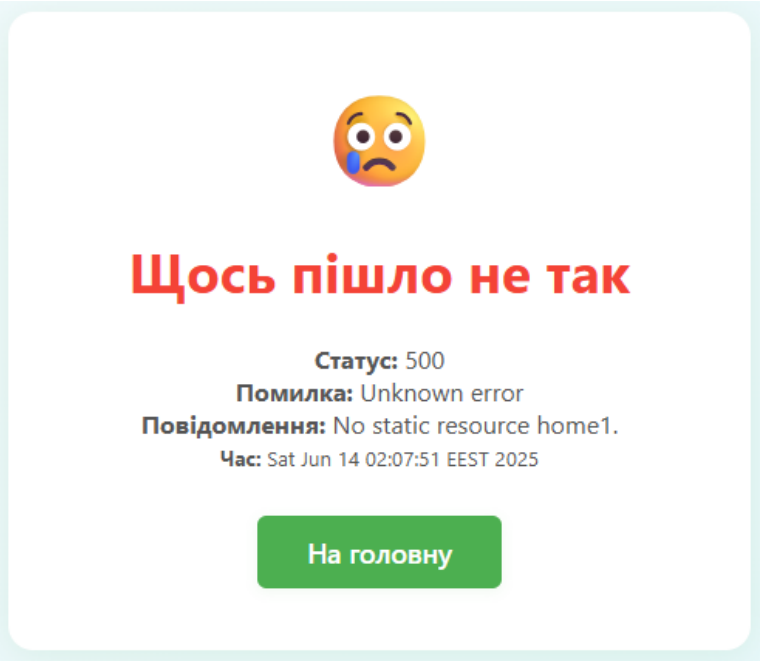

Рис. 2.34. Повідомлення про те, що сталась помилка

Весь інтерфейс оформлено у сучасному стилі, кнопки виділяються кольором і мають підказки. Усі сторінки містять навігаційні елементи для повернення на головну сторінку або до попереднього розділу (рис. 2.35-2.36).

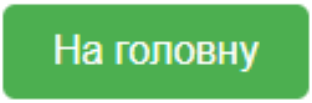

Рис. 2.35. Кнопка повернення на головну сторінку

Рис. 2.36. Кнопка у вигляді стрілки для повернення на попередню сторінку

Таким чином, робота з системою виглядає максимально просто: користувач завжди бачить тільки ті можливості, які доступні його ролі, а весь діалог із системою відбувається через прості та зрозумілі форми та таблиці.

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Вихідні дані:

1. q – передбачуване число операторів – 1453;

2. C – коефіцієнт складності програми – 1,33;

3. p – коефіцієнт корекції програми в ході її розробки – 0,06;

4. B – коефіцієнт збільшення витрат праці в наслідок недостатнього опису – 1,29;

5. k – коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 0,8;

6. $С_{пр}$ – середня зарплатня – 236,19 грн/год. Відповідно до даних DOU, медіанна місячна зарплатня Junior Java розробника складає приблизно 1000$.

7. B_k – число виконавців – 1;

8. $F_р$ – місячний фонд робочого часу (при 40 годинному робочому тижні $F_р$ = 176 годин);

9. $С_{мч}$ – вартість машинно-години – 2,54 грн/год. Комп'ютер на якому проводилася розробка споживає до 200 Вт електроенергії на годину. Вартість електроенергії за кВт складає 4,32 грн [17]. Вартість Інтернету за місяць складає 295 грн [18].

Нормування праці при створенні програмного забезпечення суттєво ускладнюється в силу творчого характеру роботи програміста. Тому оцінка трудомісткості розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_о + t_и + t_а + t_п + t_{отл} + t_д, \quad (3.1)$$

де $t_о$ – витрати праці на підготовку й опис поставленої задачі (приймається 50 людино-годин);

$t_и$ – витрати праці на дослідження алгоритму рішення задачі, людино-годин;

$t_а$ – витрати праці на розробку блок-схеми алгоритму, людино-годин;

$t_п$ – витрати праці на програмування по готовій блок-схемі, людино-годин;

$t_{отл}$ – витрати праці на налагодження програми на ЕОМ, людино-годин;

$t_д$ – витрати праці на підготовку документації, людино-годин.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм) визначається за формулою:

$$Q = q \cdot C \cdot (1 + p),$$

де q – передбачуване число операторів;

C – коефіцієнт складності програми;

p – коефіцієнт корекції програми в ході її розробки.

Виконаємо розрахунок відповідно до формули:

$$Q = 1453 \cdot 1{,}33 \cdot (1 + 0{,}06) = 2049{,}5$$

Витрати праці на дослідження алгоритму рішення задачі визначаються за формулою:

$$t_и = \frac{Q \cdot B}{(75..85) \cdot k},$$

де Q – умовне число операторів програми;

B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

k – коефіцієнт кваліфікації програміста.

Виконаємо розрахунок відповідно до формули:

$$t_{\text{и}} = \frac{2049{,}5 \cdot 1{,}29}{84 \cdot 0{,}8} = 39{,}34 \text{ людино} - \text{годин}.$$

Витрати праці на розробку блок-схеми алгоритму визначаються за формулою:

$$t_{\text{а}} = \frac{Q}{(20..25) \cdot k},$$

де Q – умовне число операторів програми;

k – коефіцієнт кваліфікації програміста.

Виконаємо розрахунок відповідно до формули:

$$t_{\text{а}} = \frac{2049{,}5}{24 \cdot 0{,}8} = 106{,}77 \text{ людино} - \text{годин}.$$

Витрати праці на програмування по готовій блок-схемі визначаються за формулою:

$$t_{\text{п}} = \frac{Q}{(20..25) \cdot k},$$

де Q – умовне число операторів програми;

k – коефіцієнт кваліфікації програміста.

Виконаємо розрахунок відповідно до формули:

$$t_{\text{п}} = \frac{2049{,}5}{23 \cdot 0{,}8} = 111{,}39 \text{ людино} - \text{годин}.$$

Витрати праці на налагодження програми на ЕОМ визначаються за формулою:

$$t_{\text{отл}} = \frac{Q}{(4..5) \cdot k},$$

де Q – умовне число операторів програми;

k – коефіцієнт кваліфікації програміста.

Виконаємо розрахунок відповідно до формули:

$$t_{\text{отл}} = \frac{2049{,}5}{4 \cdot 0{,}8} = 640{,}47 \text{ людино} - \text{годин}.$$

Витрати праці на підготовку документації визначаються за формулою:

$$t_{\text{д}} = t_{\text{др}} + t_{\text{до}}, \tag{3.2}$$

де $t_{\text{др}}$ – трудомісткість підготовки матеріалів і рукопису, людино-годин;

$t_{\text{до}}$ – трудомісткість редагування, печатки й оформлення документації, людино-годин.

Трудомісткість підготовки матеріалів і рукопису визначають за формулою:

$$t_{\text{др}} = \frac{Q}{(15..20) \cdot k},$$

де Q – умовне число операторів програми;

k – коефіцієнт кваліфікації програміста.

Виконаємо розрахунок відповідно до формули:

$$t_{\text{др}} = \frac{2049,5}{18 \cdot 0,8} = 142,32 \text{ людино} - \text{годин}.$$

Трудомісткість редагування, печатки й оформлення документації визначаються за формулою:

$$t_{\text{до}} = 0,75 \cdot t_{\text{др}},$$

де $t_{\text{др}}$ – трудомісткість підготовки матеріалів і рукопису.

Виконаємо розрахунок відповідно до формули:

$$t_{\text{до}} = 0,75 \cdot 142,32 = 106,74 \text{ людино} - \text{годин}.$$

Відповідно до формули (3.2), розрахуємо витрати праці на підготовку документації:

$$t_{\text{д}} = 142,32 + 106,74 = 249,06 \text{ людино} - \text{годин}.$$

Повертаючись до формули (3.1), отримаємо повну оцінку трудомісткості розробки програмного забезпечення:

$$t = 50 + 39,34 + 106,77 + 111,39 + 640,47 + 249,06$$
$$= 1197,03 \text{ людино} - \text{годин}.$$

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ розраховуються за формулою:

$$К_{по} = З_{зп} + З_{мч}, \quad (3.3)$$

де $З_{зп}$ – витрати на заробітну платню розробника програми, грн;

$З_{мч}$ – витрати машинного часу, необхідного для налагодження програми на ЕОМ, грн.

Заробітна плата виконавців визначається за формулою:

$$З_{зп} = t \cdot С_{пр},$$

де t – загальна трудомісткістю;

$С_{пр}$ – середня годинна заробітна плата програміста.

Виконаємо розрахунок відповідно до формули:

$$З_{зп} = 1197{,}03 \cdot 236{,}19 = 282739{,}49 \text{ грн.}$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{мч} = t_{отл} \cdot С_{мч},$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ;

$С_{мч}$ – вартість машино-години ЕОМ.

Виконаємо розрахунок відповідно до формули:

$$З_{мч} = 640{,}47 \cdot 2{,}54 = 1626{,}59 \text{ грн.}$$

Звідси, за формулою (3.3), витрати на створення програмного продукту:

$$К_{по} = 282739{,}49 + 1626{,}59 = 284366{,}08 \text{ грн.}$$

Очікуваний період створення програмного застосунку розраховується за формулою:

$$T = \frac{t}{B_k \cdot F_p},$$

де B_k – число виконавців;

F_p – місячний фонд робочого часу.

Виконаємо розрахунок відповідно до формули:

$$T = \frac{1197{,}03}{1 \cdot 176} = 6{,}8 \text{ місяців}$$

Висновки: розробка проекту кваліфікаційної роботи буде займати 1197,03 людино-годин. Очікувана тривалість розробки – 6,8 місяців при робочому місяці, що складає 176 годин. Витрати на створення програмного продукту складатимуть 284366,08 грн.

ВИСНОВКИ

Відповідно до завдання та мети кваліфікаційної роботи було реалізовано вебдодаток для управління готельним комплексом та бронювання номерів.

У ході виконання кваліфікаційної роботи було розроблено багаторівневу архітектуру на основі Spring Boot з використанням мов програмування Java та HTML (Thymeleaf). Впроваджено автентифікацію, авторизацію для різних ролей користувачів (адміністратор, клієнт), кастомні сторінки обробки помилок, а також функціонал керування клієнтами, номерами готелю та бронюваннями. Було підтверджено зручність використання фреймворку Spring Boot для створення сучасних веб-застосунків, а також швидкість та надійність інтеграції з СУБД PostgreSQL.

Практичним значенням розробленого застосунку є спрощення обліку клієнтів і бронювань для малих готелів, а також можливість розширення та адаптації під різні бізнес-потреби. Завдяки використанню відкритих технологій додаток легко розгортається, масштабується та інтегрується з іншими корпоративними рішеннями.

Відповідно до розрахунків проведених у економічному розділі кваліфікаційної роботи, розробка проекту буде займати 1197,03 людино-годин з очікуваним періодом розробки у 6,8 місяців та вартості у 284366,08 грн.

Додаток має ряд недоліків та покращень над якими планується робота у майбутньому:

– впровадження JS фреймворків для підвищення зручності користування;

– інтеграція з зовнішніми сервісами;

– автоматизація тестування;

– можливість експорту у різні формати даних;

– реалізація двофакторної автентифікації (2FA).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. RESTful API – як спосіб спілкування компонент веб-додатків. URL: https://foxminded.ua/shcho-take-rest-api/. Дата звернення 12.06.2025

2. CRUD – перевірка бази даних. Що таке CRUD? URL: https://training.qatestlab.com/blog/technical-articles/crud-database-validation/. Дата звернення 12.06.2025

3. JSON. URL: https://qalight.ua/baza-znaniy/shho-take-json/. Дата звернення 12.06.2025

4. JWT – Wikipedia. URL: https://en.wikipedia.org/wiki/JSON_Web_Token. Дата звернення 12.06.2025

5. RBAC – Контроль доступу на основі ролей. URL: https://uk.wikipedia.org/wiki/Керування_доступом_на_основі_ролей. Дата звернення 12.06.2025

6. MVC. URL: https://www.geeksforgeeks.org/software-engineering/mvc-framework-introduction/. Дата звернення 12.06.2025

7. Client–server model – Wikipedia. URL: https://en.wikipedia.org/wiki/Client%E2%80%93server_model. Дата звернення 12.06.2025

8. Spring Boot. URL: https://docs.spring.io/spring-boot/index.html. Дата звернення 12.06.2025

9. Java. URL: https://docs.oracle.com/en/database/oracle/oracle-database/19/jjdev/Java-overview.html#GUID-061CB7CD-144F-4B3C-9409-748B94C25A09. Дата звернення 12.06.2025

10. Spring Security. URL: https://spring.io/projects/spring-security, https://docs.spring.io/spring-security/reference/. Дата звернення 12.06.2025

11. Thymeleaf. URL: https://www.thymeleaf.org/documentation.html. Дата звернення 12.06.2025

12. PostgreSQL. URL: https://www.postgresql.org/docs/. Дата звернення 12.06.2025

13. Maven. URL: https://maven.apache.org/guides/. Дата звернення 12.06.2025

14. Git. URL: https://git-scm.com/doc. Дата звернення 12.06.2025

15. Spring Data JPA. URL: https://docs.spring.io/spring-data/jpa/reference/index.html. Дата звернення 12.06.2025

16. Lombok. URL: https://www.it-notes.wiki/java/library-lombok/. Дата звернення 12.06.2025

17. Тарифи. URL: https://yasno.com.ua/b2c-tariffs. Дата звернення 12.06.2025

18. Інтернет Фрегат. URL: https://fregat.com/rates-and-promotions/price/. Дата звернення 12.06.2025

19. Deinum M., Cosmina I. Pro Spring MVC with WebFlux: Web Development in Spring Framework 5 and Spring Boot 2. Springer, 2021. DOI: 10.1007/978-1-4842-5666-4

20. Cosmina I., Harrop R., Schaefer C., Ho C. Spring MVC. В кн.: Pro Spring 6. Springer, 2023. DOI: 10.1007/978-1-4842-8640-1_142

21. Downey T. Spring MVC. В кн.: Guide to Web Development with Java. Springer, 2021. DOI: 10.1007/978-3-030-62274-9_54

22. Bloch J. Effective Java (3rd Edition). Addison-Wesley, 2018. URL: https://kea.nu/files/textbooks/new/Effective%20Java%20%282017%2C%20Addison-Wesley%29.pdf

23. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley, 2003. URL: https://fabiofumarola.github.io/nosql/readingMaterial/Evans03.pdf

24. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994. URL: https://www.javier8a.com/itc/bd1/articulo.pdf6

ДОДАТОК А

КОД ПРОГРАМИ

User.java
```
package com.starchenko.hotelmanagementapi.model;
import jakarta.persistence.*;
import lombok.Data;
@Entity
@Data
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(unique = true, nullable = false)
    private String email;
    private String password;
    private String role;
}
```

Room.java
```
package com.starchenko.hotelmanagementapi.model;
import jakarta.persistence.*;
import lombok.Data;
@Entity
@Data
public class Room {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String number;
    private String type;
    @Column(name = "price_per_night")
    private Double pricePerNight;
    @Column(name = "is_available")
    private Boolean isAvailable;
}
```

Customer.java
```
package com.starchenko.hotelmanagementapi.model;
import jakarta.persistence.*;
import lombok.*;
@Entity
@Data
public class Customer {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String email;
    @Column(name = "phone_number")
    private String phoneNumber;
}
```

Booking.java
```
package com.starchenko.hotelmanagementapi.model;
```

```
import jakarta.persistence.*;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotNull;
import lombok.Data;
import java.time.LocalDate;
@Entity
@Data
public class Booking {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @ManyToOne
    @JoinColumn(name = "room_id")
    private Room room;
    @ManyToOne
    @JoinColumn(name = "customer_id")
    private Customer customer;
    @Column(name = "check_in_date")
    private LocalDate checkInDate;
    @Column(name = "check_out_date")
    private LocalDate checkOutDate;
    private String status;
}
```

UserRepository.java

```
package com.starchenko.hotelmanagementapi.repository;
import com.starchenko.hotelmanagementapi.model.User;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.Optional;
public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByEmail(String email);
}
```

RoomRepository.java

```
package com.starchenko.hotelmanagementapi.repository;
import com.starchenko.hotelmanagementapi.model.Room;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.List;
public interface RoomRepository extends JpaRepository<Room, Long> {
    List<Room> findByIsAvailableTrue();
}
```

CustomerRepository.java

```
package com.starchenko.hotelmanagementapi.repository;
import com.starchenko.hotelmanagementapi.model.Customer;
import org.springframework.data.jpa.repository.JpaRepository;
public interface CustomerRepository extends JpaRepository<Customer, Long> {
    public Customer findByEmail(String email);
}
```

BookingRepository.java

```
package com.starchenko.hotelmanagementapi.repository;
import com.starchenko.hotelmanagementapi.model.Booking;
import com.starchenko.hotelmanagementapi.model.Customer;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.List;
public interface BookingRepository extends JpaRepository<Booking, Long> {
    List<Booking> findAllByCustomer(Customer customer);
    List<Booking> findAllByCustomerEmail(String email);
}
```

RoomService.java

```
package com.starchenko.hotelmanagementapi.service;
import com.starchenko.hotelmanagementapi.model.Room;
import com.starchenko.hotelmanagementapi.repository.RoomRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;
@Service
public class RoomService {
    @Autowired
    private RoomRepository roomRepository;
    public List<Room> getAllRooms() {
        return roomRepository.findAll();
    }
    public List<Room> getAvailableRooms() {
        return roomRepository.findByIsAvailableTrue();
    }
    public Room createRoom(Room room) {
        return roomRepository.save(room);
    }
    public Room getRoomById(Long id) {
        return roomRepository.findById(id).orElseThrow(() -> new RuntimeException("Room
not found"));
    }
    public Room updateRoom(Long id, Room room) {
        if (!roomRepository.existsById(id)) {
            throw new RuntimeException("Customer not found");
        }
        room.setId(id);
        return roomRepository.save(room);
    }
    public void deleteRoom(Long id) {
        if (!roomRepository.existsById(id)) {
            throw new RuntimeException("Customer not found");
        }
        roomRepository.deleteById(id);
    }
}
```

CustomerService.java

```
package com.starchenko.hotelmanagementapi.service;
import com.starchenko.hotelmanagementapi.model.Customer;
import com.starchenko.hotelmanagementapi.repository.CustomerRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;

@Service
public class CustomerService {
    @Autowired
    private CustomerRepository customerRepository;
    public List<Customer> getAllCustomers() {
        return customerRepository.findAll();
    }
    public Customer getCustomerById(Long id) {
        return customerRepository.findById(id).orElseThrow(() -> new
RuntimeException("Customer not found"));
    }
    public Customer getCustomerByEmail(String email) {
        return customerRepository.findByEmail(email);
    }
    public Customer createCustomer(Customer customer) {
        return customerRepository.save(customer);
```

```
    }
    public Customer updateCustomer(Long id, Customer customer) {
        if (!customerRepository.existsById(id)) {
            throw new RuntimeException("Customer not found");
        }
        customer.setId(id);
        return customerRepository.save(customer);
    }
    public void deleteCustomer(Long id) {
        if (!customerRepository.existsById(id)) {
            throw new RuntimeException("Customer not found");
        }
        customerRepository.deleteById(id);
    }
}
```

BookingService.java

```
package com.starchenko.hotelmanagementapi.service;
import com.starchenko.hotelmanagementapi.model.Booking;
import com.starchenko.hotelmanagementapi.model.Customer;
import com.starchenko.hotelmanagementapi.repository.BookingRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;
@Service
public class BookingService {
    @Autowired
    private BookingRepository bookingRepository;
    public List<Booking> getAllBookings() {
        return bookingRepository.findAll();
    }
    public Booking getBookingById(Long id) {
        return bookingRepository.findById(id).orElseThrow(() -> new
RuntimeException("Booking not found"));
    }
    public List<Booking> getBookingsByCustomer(Customer customer) {
        return bookingRepository.findAllByCustomer(customer);
    }
    public List<Booking> getBookingsByCustomerEmail(String email) {
        return bookingRepository.findAllByCustomerEmail(email);
    }
    public Booking createBooking(Booking booking) {
        return bookingRepository.save(booking);
    }
    public Booking updateBooking(Long id, Booking booking) {
        if (!bookingRepository.existsById(id)) {
            throw new RuntimeException("Booking not found");
        }
        booking.setId(id);
        return bookingRepository.save(booking);
    }
    public void deleteBooking(Long id) {
        if (!bookingRepository.existsById(id)) {
            throw new RuntimeException("Booking not found");
        }
        bookingRepository.deleteById(id);
    }
}
```

CustomUserDetailsService.java

```
package com.starchenko.hotelmanagementapi.service;
import com.starchenko.hotelmanagementapi.model.User;
```

```
import com.starchenko.hotelmanagementapi.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
@Service
public class CustomUserDetailsService implements UserDetailsService {
    @Autowired
    private UserRepository userRepository;
    @Override
    public UserDetails loadUserByUsername(String email) throws
UsernameNotFoundException {
        User user = userRepository.findByEmail(email)
                .orElseThrow(() -> new UsernameNotFoundException("User not found: " +
email));
        return org.springframework.security.core.userdetails.User.builder()
                .username(user.getEmail())
                .password(user.getPassword())
                .roles(user.getRole())
                .build();
    }
}
```

CustomErrorController.java

```
package com.starchenko.hotelmanagementapi.controller;
import org.springframework.boot.web.error.ErrorAttributeOptions;
import org.springframework.boot.web.servlet.error.ErrorAttributes;
import org.springframework.boot.web.servlet.error.ErrorController;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.context.request.WebRequest;
import java.util.Map;

@Controller
public class CustomErrorController implements ErrorController {
    private final ErrorAttributes errorAttributes;
    public CustomErrorController(ErrorAttributes errorAttributes) {
        this.errorAttributes = errorAttributes;
    }
    @GetMapping("/error")
    public String handleError(WebRequest webRequest, Model model) {
        Map<String, Object> errorDetails =
errorAttributes.getErrorAttributes(webRequest,
ErrorAttributeOptions.of(ErrorAttributeOptions.Include.MESSAGE));
        int status = (int) errorDetails.get("status");
        model.addAttribute("status", status);
        model.addAttribute("error", errorDetails.get("error"));
        model.addAttribute("message", errorDetails.get("message"));
        model.addAttribute("timestamp", errorDetails.get("timestamp"));
        return "error";
    }
}
```

WebController.java

```
package com.starchenko.hotelmanagementapi.controller;
import com.starchenko.hotelmanagementapi.model.Booking;
import com.starchenko.hotelmanagementapi.model.Customer;
import com.starchenko.hotelmanagementapi.model.Room;
import com.starchenko.hotelmanagementapi.model.User;
import com.starchenko.hotelmanagementapi.repository.UserRepository;
```

```
import com.starchenko.hotelmanagementapi.service.BookingService;
import com.starchenko.hotelmanagementapi.service.CustomerService;
import com.starchenko.hotelmanagementapi.service.RoomService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
import java.time.LocalDate;
import java.util.List;
@Controller
public class WebController {
    @Autowired
    private BookingService bookingService;
    @Autowired
    private CustomerService customerService;
    @Autowired
    private RoomService roomService;
    @Autowired
    private UserRepository userRepository;
    @Autowired
    private PasswordEncoder passwordEncoder;
    private static final String ADMIN_PASSWORD = "admin123";
    @GetMapping("/home")
    public String homePage() {
        return "home";
    }
    @GetMapping("/register")
    public String showRegistrationForm(Model model) {
        model.addAttribute("user", new User());
        return "register";
    }
    @PostMapping("/register")
    public String registerUser(@ModelAttribute User user) {
        if (ADMIN_PASSWORD.equals(user.getPassword())) {
            user.setRole("ADMIN");
        } else {
            user.setRole("USER");
        }
        user.setPassword(passwordEncoder.encode(user.getPassword()));
        userRepository.save(user);
        return "redirect:/login";
    }
    @GetMapping("/login")
    public String loginPage(Model model) {
        model.addAttribute("user", new User());
        return "login";
    }
    @PostMapping("/login")
    public String processLogin() {
        return "redirect:/home";
    }
    @GetMapping("/admin/customers")
    public String viewCustomers(Model model) {
        model.addAttribute("customers", customerService.getAllCustomers());
        return "admin/customers/customers";
    }
    @GetMapping("/admin/customers/add")
    public String addCustomerPage(Model model) {
```

```
    model.addAttribute("customer", new Customer());
    return "admin/customers/add-customer";
}
@PostMapping("/admin/customers/add")
public String addCustomer(@ModelAttribute Customer customer) {
    try {
        customerService.createCustomer(customer);
        return "redirect:/admin/customers?success";
    } catch (Exception e) {
        return "redirect:/admin/customers?error";
    }
}
@GetMapping("/admin/customers/edit/{id}")
public String editCustomerPage(@PathVariable Long id, Model model) {
    model.addAttribute("customer", customerService.getCustomerById(id));
    return "admin/customers/edit-customer";
}
@PostMapping("/admin/customers/edit/{id}")
public String editCustomer(@PathVariable Long id,
                           @ModelAttribute Customer customer) {
    try {
        customerService.updateCustomer(id, customer);
        return "redirect:/admin/customers?success";
    } catch (Exception e) {
        return "redirect:/admin/customers?error";
    }
}
@PostMapping("/admin/customers/delete/{id}")
public String deleteCustomer(@PathVariable Long id) {
    try {
        customerService.deleteCustomer(id);
        return "redirect:/admin/customers?success";
    } catch (Exception e) {
        return "redirect:/admin/customers?error";
    }
}
@GetMapping("/admin/bookings")
public String viewBookings(Model model) {
    model.addAttribute("bookings", bookingService.getAllBookings());
    return "admin/bookings/bookings";
}
@GetMapping("/admin/bookings/details/{id}")
public String viewBookingDetails(@PathVariable Long id, Model model) {
    Booking booking = bookingService.getBookingById(id);
    model.addAttribute("booking", booking);
    return "admin/bookings/booking-details";
}
@GetMapping("/admin/bookings/add")
public String addBookingPage(Model model) {
    model.addAttribute("booking", new Booking());
    model.addAttribute("customers", customerService.getAllCustomers());
    model.addAttribute("rooms", roomService.getAllRooms());
    return "admin/bookings/add-booking";
}
@PostMapping("/admin/bookings/add")
public String addBooking(@ModelAttribute Booking booking,
                         @RequestParam Long customerId,
                         @RequestParam Long roomId) {
    try {
        Customer customer = customerService.getCustomerById(customerId);
        Room room = roomService.getRoomById(roomId);
        booking.setCustomer(customer);
```

```
            booking.setRoom(room);
            bookingService.createBooking(booking);
            return "redirect:/admin/bookings?success";
        } catch (Exception e) {
            return "redirect:/admin/bookings?error";
        }
    }
    @GetMapping("/admin/bookings/edit/{id}")
    public String editBookingPage(@PathVariable Long id, Model model) {
        model.addAttribute("booking", bookingService.getBookingById(id));
        model.addAttribute("customers", customerService.getAllCustomers());
        model.addAttribute("rooms", roomService.getAllRooms());
        return "admin/bookings/edit-booking";
    }
    @PostMapping("/admin/bookings/edit/{id}")
    public String editBooking(@PathVariable Long id,
                              @RequestParam Long customerId,
                              @RequestParam Long roomId,
                              @ModelAttribute Booking booking) {
        try {
            Customer customer = customerService.getCustomerById(customerId);
            Room room = roomService.getRoomById(roomId);
            booking.setCustomer(customer);
            booking.setRoom(room);
            bookingService.updateBooking(id, booking);
            return "redirect:/admin/bookings?success";
        } catch (Exception e) {
            return "redirect:/admin/bookings?error";
        }
    }
    @PostMapping("/admin/bookings/delete/{id}")
    public String deleteBooking(@PathVariable Long id) {
        try {
            bookingService.deleteBooking(id);
            return "redirect:/admin/bookings?success";
        } catch (Exception e) {
            return "redirect:/admin/bookings?error";
        }
    }
    @GetMapping("/admin/rooms")
    public String viewRooms(Model model) {
        model.addAttribute("rooms", roomService.getAllRooms());
        return "admin/rooms/rooms";
    }
    @GetMapping("/admin/rooms/add")
    public String addRoomPage(Model model) {
        model.addAttribute("room", new Room());
        return "admin/rooms/add-room";
    }
    @PostMapping("/admin/rooms/add")
    public String addRoom(@ModelAttribute Room room) {
        try {
            roomService.createRoom(room);
            return "redirect:/admin/rooms?success";
        } catch (Exception e) {
            return "redirect:/admin/rooms?error";
        }
    }
    @GetMapping("/admin/rooms/edit/{id}")
    public String editRoomPage(@PathVariable Long id, Model model) {
        model.addAttribute("room", roomService.getRoomById(id));
        return "admin/rooms/edit-room";
```

```
    }
    @PostMapping("/admin/rooms/edit/{id}")
    public String editRoom(@PathVariable Long id, @ModelAttribute Room room) {
        try {
            roomService.updateRoom(id, room);
            return "redirect:/admin/rooms?success";
        } catch (Exception e) {
            return "redirect:/admin/rooms?error";
        }
    }

    @PostMapping("/admin/rooms/delete/{id}")
    public String deleteRoom(@PathVariable Long id) {
        try {
            roomService.deleteRoom(id);
            return "redirect:/admin/rooms?success";
        } catch (Exception e) {
            return "redirect:/admin/rooms?error";
        }
    }
    @GetMapping("/customer/book-room")
    public String bookRoomPage(Model model) {
        model.addAttribute("rooms", roomService.getAvailableRooms());
        model.addAttribute("customer", new Customer());
        return "customer/book-room";
    }
    @PostMapping("/customer/book-room")
    public String processCustomerBooking(@ModelAttribute Customer customer,
                                         @RequestParam Long roomId,
                                         @RequestParam LocalDate checkInDate,
                                          @RequestParam LocalDate checkOutDate)
       {
        try {
            Room room = roomService.getRoomById(roomId);
            customer = customerService.createCustomer(customer);
            Booking booking = new Booking();
            booking.setCustomer(customer);
            booking.setRoom(room);
            booking.setCheckInDate(checkInDate);
            booking.setCheckOutDate(checkOutDate);
            booking.setStatus("Очікується");
            bookingService.createBooking(booking);
            return "redirect:/customer/my-bookings?success";
        } catch (Exception e) {
            return "redirect:/customer/my-bookings?error";
        }
    }
    @GetMapping("/customer/my-bookings")
    public String viewMyBookings(Model model) {
        Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
        String email = authentication.getName();
        List<Booking> bookings = bookingService.getBookingsByCustomerEmail(email);
        model.addAttribute("bookings", bookings);
        return "customer/my-bookings";
    }
    @GetMapping("/customer/my-bookings/details/{id}")
    public String viewCustomerBookingDetails(@PathVariable Long id, Model model) {
        Booking booking = bookingService.getBookingById(id);
        model.addAttribute("booking", booking);
        return "customer/booking-details";
    }
```

```
}
```

SecurityConfig.java

```
package com.starchenko.hotelmanagementapi.config;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.config.annotation.authentication.configuration.Authenticat
ionConfiguration;
import
org.springframework.security.config.annotation.method.configuration.EnableMethodSecurit
y;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
@Configuration
@EnableWebSecurity
@EnableMethodSecurity
public class SecurityConfig {
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception
{
        http
                .authorizeHttpRequests(authz -> authz
                        .requestMatchers("/", "/home", "/register", "/login",
"/css/**", "/js/**", "/error").permitAll()
                        .requestMatchers("/customer/**").hasRole("USER")
                        .requestMatchers("/admin/**").hasRole("ADMIN")
                        .anyRequest().authenticated()
                )
                .formLogin(form -> form
                        .loginPage("/login")
                        .defaultSuccessUrl("/home", true)
                        .usernameParameter("email")
                        .passwordParameter("password")
                        .permitAll()
                )
                .logout(logout -> logout
                        .logoutSuccessUrl("/")
                        .invalidateHttpSession(true)
                        .deleteCookies("JSESSIONID")
                        .permitAll()
                );
        return http.build();
    }
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
    @Bean
    public AuthenticationManager authenticationManager(AuthenticationConfiguration
authenticationConfiguration) throws Exception {
        return authenticationConfiguration.getAuthenticationManager();
    }
}
```

ThymeleafConfig.java

```
package com.starchenko.hotelmanagementapi.config;
```

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.thymeleaf.extras.springsecurity6.dialect.SpringSecurityDialect;
@Configuration
public class ThymeleafConfig {
    @Bean
    public SpringSecurityDialect springSecurityDialect() {
        return new SpringSecurityDialect();
    }
}
```

/templates/home.html

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/extras/spring-security">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Головна сторінка</title>
    <link rel="stylesheet" th:href="@{/css/home.css}">
</head>
<body>
<header>
    <h1>Ласкаво просимо до Hotel Management</h1>
    <div class="header-buttons">
        <span sec:authorize="isAuthenticated()" class="green-text">
            Ви увійшли як: <span sec:authentication="principal.username"></span><br>
            Ваша роль:
            <span
th:text="${#strings.replace(#strings.replace(#authentication.authorities,'[',''),']',''
).replace('ROLE_','')}"></span>
        </span>
        <form th:action="@{/logout}" method="post" sec:authorize="isAuthenticated()">
            <button type="submit" class="button">Вийти</button>
        </form>
        <a th:href="@{/register}" class="button"
sec:authorize="!isAuthenticated()">Реєстрація</a>
        <a th:href="@{/login}" class="button"
sec:authorize="!isAuthenticated()">Логін</a>
    </div>
</header>
<main>
    <section class="welcome">
        <p>Керуйте своїми бронюваннями, номерами та клієнтами легко та зручно.</p>
    </section>
    <section class="cards" sec:authorize="hasRole('ADMIN')">
        <h2>Функціонал адміністратора</h2>
        <div class="card-container">
            <a th:href="@{/admin/customers}" class="card">
                <h3>Список клієнтів</h3>
                <p>Перегляньте список усіх клієнтів.</p>
            </a>
            <a th:href="@{/admin/rooms}" class="card">
                <h3>Список номерів</h3>
                <p>Перегляньте доступні номери.</p>
            </a>
            <a th:href="@{/admin/bookings}" class="card">
                <h3>Список замовлень</h3>
                <p>Перегляньте всі замовлення.</p>
            </a>
        </div>
```

```
    </section>
    <section class="cards" sec:authorize="hasRole('USER')">
        <h2>Функціонал клієнта</h2>
        <div class="card-container">
            <a th:href="@{/customer/book-room}" class="card">
                <h3>Бронювання номера</h3>
                <p>Оберіть номер для бронювання.</p>
            </a>
            <a th:href="@{/customer/my-bookings}" class="card">
                <h3>Мої бронювання</h3>
                <p>Перегляньте свої бронювання.</p>
            </a>
        </div>
    </section>
</main>
<footer>
    <p>&copy; 2025 Hotel Management</p>
</footer>
</body>
</html>
```

/templates/register.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Реєстрація</title>
    <link rel="stylesheet" th:href="@{/css/auth.css}">
</head>
<body>
<header>
    <h1>Реєстрація</h1>
</header>
<main>
    <section class="form-container">
        <a th:href="@{/home}" class="home-arrow" title="На головну">&#8592;</a>
        <form th:action="@{/register}" th:object="${user}" method="post">
            <div class="form-group">
                <label for="email">Електронна пошта</label>
                <input type="email" id="email" name="email" th:field="*{email}"
required>
            </div>
            <div class="form-group">
                <label for="password">Пароль</label>
                <input type="password" id="password" name="password"
th:field="*{password}" required>
            </div>
            <div class="form-actions">
                <button type="submit" class="button">Зареєструватися</button>
            </div>
        </form>
        <p class="form-footer">
            Вже маєте обліковий запис? <a th:href="@{/login}">Увійти</a>
        </p>
    </section>
</main>
<footer>
    <p>&copy; 2025 Hotel Management</p>
</footer>
</body>
</html>
```

/templates/login.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Увійти</title>
    <link rel="stylesheet" th:href="@{/css/auth.css}">
</head>
<body>
<header>
    <h1>Увійти</h1>
</header>
<main>
    <section class="form-container">
        <a th:href="@{/home}" class="home-arrow" title="На головну">&#8592;</a>
        <form th:action="@{/login}" method="post">
            <div class="form-group">
                <label for="email">Електронна пошта</label>
                <input type="email" id="email" name="email" required>
            </div>
            <div class="form-group">
                <label for="password">Пароль</label>
                <input type="password" id="password" name="password" required>
            </div>
            <div class="form-actions">
                <button type="submit" class="button">Увійти</button>
            </div>
        </form>
        <p class="form-footer">
            Не маєте облікового запису? <a th:href="@{/register}">Зареєструватися</a>
        </p>
    </section>
</main>
<footer>
    <p>&copy; 2025 Hotel Management</p>
</footer>
</body>
</html>
```

/templates/admin/bookings/bookings.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Список бронювань</title>
    <link rel="stylesheet" th:href="@{/css/style.css}">
</head>
<body>
<header>
    <h1>Список бронювань</h1>
</header>
<main>
    <div th:if="${param.success}" class="success-message">
        Операцію виконано успішно!
    </div>
    <div th:if="${param.error}" class="error-message">
        Сталася помилка під час виконання операції.
    </div>
    <div class="actions">
        <a th:href="@{/admin/bookings/add}" class="button">Додати бронювання</a>
        <a th:href="@{/home}" class="button">На головну</a>
```

```
    </div>
    <table>
        <thead>
        <tr>
            <th>ID</th>
            <th>Клієнт</th>
            <th>Кімната</th>
            <th>Дата заїзду</th>
            <th>Дата виїзду</th>
            <th>Дії</th>
        </tr>
        </thead>
        <tbody>
        <tr th:each="booking : ${bookings}">
            <td th:text="${booking.id}"></td>
            <td th:text="${booking.customer.name}"></td>
            <td th:text="${booking.room.number}"></td>
            <td th:text="${booking.checkInDate}"></td>
            <td th:text="${booking.checkOutDate}"></td>
            <td>
                <a th:href="@{/admin/bookings/details/{id}(id=${booking.id})}"
class="button">Деталі</a>
                <a th:href="@{/admin/bookings/edit/{id}(id=${booking.id})}"
class="button">Редагувати</a>
                <form th:action="@{/admin/bookings/delete/{id}(id=${booking.id})}"
method="post" style="display:inline;">
                    <button type="submit" class="button cancel-
button">Видалити</button>
                </form>
            </td>
        </tr>
        </tbody>
    </table>
</main>
</body>
</html>
```

/templates/admin/bookings/add-booking.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Додати бронювання</title>
    <link rel="stylesheet" th:href="@{/css/style.css}">
</head>
<body>
<header>
    <h1>Додати бронювання</h1>
</header>
<main>
    <form th:action="@{/admin/bookings/add}" method="post" class="form-container">
        <div class="form-group">
            <label for="customerId">Клієнт:</label>
            <select id="customerId" name="customerId" required>
                <option value="" disabled selected>Оберіть клієнта</option>
                <option th:each="customer : ${customers}" th:value="${customer.id}"
th:text="${customer.name}"></option>
            </select>
        </div>
        <div class="form-group">
            <label for="roomId">Кімната:</label>
            <select id="roomId" name="roomId" required>
                <option value="" disabled selected>Оберіть кімнату</option>
```

```
                <option th:each="room : ${rooms}" th:value="${room.id}"
th:text="${room.number}"></option>
            </select>
        </div>
        <div class="form-group">
            <label for="checkInDate">Дата заїзду:</label>
            <input type="date" id="checkInDate" name="checkInDate" required>
        </div>
        <div class="form-group">
            <label for="checkOutDate">Дата виїзду:</label>
            <input type="date" id="checkOutDate" name="checkOutDate" required>
        </div>
        <div class="form-group">
            <label for="status">Статус:</label>
            <select id="status" name="status" required>
                <option value="CONFIRMED">Підтверджено</option>
                <option value="PENDING">Очікується</option>
                <option value="CANCELLED">Скасовано</option>
            </select>
        </div>
        <div class="form-actions">
            <button type="submit" class="button">Додати</button>
            <a th:href="@{/admin/bookings}" class="button cancel-button">Скасувати</a>
        </div>
    </form>
</main>
</body>
</html>
```

/templates/admin/bookings/edit-booking.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Редагувати бронювання</title>
    <link rel="stylesheet" th:href="@{/css/style.css}">
</head>
<body>
<header>
    <h1>Редагувати бронювання</h1>
</header>
<main>
    <form th:action="@{/admin/bookings/edit/{id}(id=${booking.id})}" method="post"
class="form-container">
        <div class="form-group">
            <label for="customerId">Клієнт:</label>
            <select id="customerId" name="customerId" required>
                <option value="" disabled>Оберіть клієнта</option>
                <option th:each="customer : ${customers}" th:value="${customer.id}"
th:text="${customer.name}"
                        th:selected="${customer.id == booking.customer.id}"></option>
            </select>
        </div>
        <div class="form-group">
            <label for="roomId">Кімната:</label>
            <select id="roomId" name="roomId" required>
                <option value="" disabled>Оберіть кімнату</option>
                <option th:each="room : ${rooms}" th:value="${room.id}"
th:text="${room.number}"
                        th:selected="${room.id == booking.room.id}"></option>
            </select>
        </div>
        <div class="form-group">
```

```
            <label for="checkInDate">Дата заїзду:</label>
            <input type="date" id="checkInDate" name="checkInDate"
th:value="${booking.checkInDate}" required>
        </div>
        <div class="form-group">
            <label for="checkOutDate">Дата виїзду:</label>
            <input type="date" id="checkOutDate" name="checkOutDate"
th:value="${booking.checkOutDate}" required>
        </div>
        <div class="form-group">
            <label for="status">Статус:</label>
            <select id="status" name="status" required>
                <option value="CONFIRMED" th:selected="${booking.status ==
'CONFIRMED'}">Підтверджено</option>
                <option value="PENDING" th:selected="${booking.status ==
'PENDING'}">Очікується</option>
                <option value="CANCELLED" th:selected="${booking.status ==
'CANCELLED'}">Скасовано</option>
            </select>
        </div>
        <div class="form-actions">
            <button type="submit" class="button">Зберегти</button>
            <a th:href="@{/admin/bookings}" class="button cancel-button">Скасувати</a>
        </div>
    </form>
</main>
</body>
</html>
```

/templates/admin/bookings/booking-details.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Деталі бронювання</title>
    <link rel="stylesheet" th:href="@{/css/style.css}">
</head>
<body>
<header>
    <h1>Деталі бронювання</h1>
</header>
<main>
    <form class="form-container">
        <div class="form-group">
            <label>Клієнт:</label>
            <input type="text" th:value="${booking.customer.name}" readonly>
        </div>
        <div class="form-group">
            <label>Електронна пошта:</label>
            <input type="text" th:value="${booking.customer.email}" readonly>
        </div>
        <div class="form-group">
            <label>Телефон:</label>
            <input type="text" th:value="${booking.customer.phoneNumber}" readonly>
        </div>
        <div class="form-group">
            <label>Номер кімнати:</label>
            <input type="text" th:value="${booking.room.number}" readonly>
        </div>
        <div class="form-group">
            <label>Тип кімнати:</label>
```

```
            <input type="text" th:value="${booking.room.type}" readonly>
        </div>
        <div class="form-group">
            <label>Ціна за ніч:</label>
            <input type="text" th:value="${booking.room.pricePerNight}" readonly>
        </div>
        <div class="form-group">
            <label>Дата заїзду:</label>
            <input type="text" th:value="${booking.checkInDate}" readonly>
        </div>
        <div class="form-group">
            <label>Дата виїзду:</label>
            <input type="text" th:value="${booking.checkOutDate}" readonly>
        </div>
        <div class="form-group">
            <label>Статус:</label>
            <input type="text" th:value="${booking.status}" readonly>
        </div>
        <div class="form-actions">
            <a th:href="@{/admin/bookings}" class="button">Повернутися до списку</a>
            <a th:href="@{/admin/bookings/edit/{id}(id=${booking.id})}"
class="button">Редагувати</a>
        </div>
    </form>
</main>
</body>
</html>
```

/templates/customer/book-room.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Замовити номер</title>
    <link rel="stylesheet" th:href="@{/css/style.css}">
</head>
<body>
<header>
    <h1>Замовити номер</h1>
</header>
<main>
    <form th:action="@{/customer/book-room}" method="post" class="form-container">
        <div class="form-group">
            <label for="name">Ім'я:</label>
            <input type="text" id="name" name="name" required>
        </div>
        <div class="form-group">
            <label for="email">Електронна пошта:</label>
            <input type="email" id="email" name="email"
th:value="${#authentication.name}" readonly required>
        </div>
        <div class="form-group">
            <label for="phone">Телефон:</label>
            <input type="tel" id="phone" name="phoneNumber" required>
        </div>
        <div class="form-group">
            <label for="roomId">Кімната:</label>
            <select id="roomId" name="roomId" required>
                <option value="" disabled selected>Оберіть кімнату</option>
                <option th:each="room : ${rooms}" th:value="${room.id}"
th:text="${room.number}"></option>
            </select>
        </div>
```

```
        <div class="form-group">
            <label for="checkInDate">Дата заїзду:</label>
            <input type="date" id="checkInDate" name="checkInDate" required>
        </div>
        <div class="form-group">
            <label for="checkOutDate">Дата виїзду:</label>
            <input type="date" id="checkOutDate" name="checkOutDate" required>
        </div>
        <div class="form-actions">
            <button type="submit" class="button">Замовити</button>
            <a th:href="@{/home}" class="button cancel-button">Скасувати</a>
        </div>
    </form>
</main>
</body>
</html>
```

/static/css/home.css

```
html, body {
    height: 100%;
    margin: 0;
    display: flex;
    flex-direction: column;
}
main {
    flex: 1;
    margin: 20px auto;
    max-width: 900px;
    padding: 20px;
}
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    background-color: #f9f9f9;
}
header {
    background-color: #4CAF50;
    color: white;
    padding: 20px;
    text-align: center;
}
main {
    margin: 20px auto;
    max-width: 900px;
    padding: 20px;
}
footer {
    text-align: center;
    padding: 10px;
    background-color: #4CAF50;
    color: white;
    margin-top: auto;
}
/* Стили для секций */
.welcome {
    text-align: center;
    margin-bottom: 30px;
    font-size: 18px;
    color: #555;
}
.cards {
```

```
    margin-bottom: 30px;
}
.cards h2 {
    text-align: center;
    margin-bottom: 20px;
    color: #333;
}
.card-container {
    display: flex;
    gap: 20px;
    justify-content: center;
    flex-wrap: wrap;
}
/* Стили для карточек */
.card {
    background-color: white;
    border: 1px solid #ddd;
    border-radius: 8px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
    padding: 20px;
    text-align: center;
    text-decoration: none;
    color: black;
    width: 250px;
    transition: transform 0.3s, box-shadow 0.3s;
}
.card:hover {
    transform: translateY(-5px);
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
}
.card h3 {
    margin-bottom: 10px;
    color: #4CAF50;
}
.card p {
    font-size: 14px;
    color: #555;
}
.header-buttons .button {
    padding: 12px 18px;
    font-size: 16px;
    font-weight: bold;
    color: #4CAF50;
    background-color: white;
    border: 3px solid #4CAF50;
    border-radius: 8px;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.2);
    transition: transform 0.3s, box-shadow 0.3s, background-color 0.3s, color 0.3s;
    margin-left: 10px;
    text-decoration: none;
}
.header-buttons .button:hover {
    background-color: #4CAF50;
    color: white;
    transform: scale(1.1);
    box-shadow: 0 6px 8px rgba(0, 0, 0, 0.3);
}
.header-buttons {
    position: absolute;
    top: 20px;
    right: 20px;
    display: flex;
```

```
    gap: 15px;
    background-color: rgba(255, 255, 255, 0.8);
    padding: 10px;
    border-radius: 10px;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.2);
}
.green-text {
    color: green;
    font-weight: bold;
}
```

/static/css/auth.css

```
html, body {
    height: 100%;
    margin: 0;
    padding: 0;
    overflow-x: hidden;
    display: flex;
    flex-direction: column;
    background: linear-gradient(135deg, #f4f4f4, #e0e0e0);
    background-size: cover;
}
header {
    background-color: #4CAF50;
    color: white;
    padding: 20px;
    text-align: center;
    font-size: 24px;
    font-weight: bold;
}
main {
    flex: 1;
    display: flex;
    justify-content: center;
    align-items: center;
    padding: 20px;
}
footer {
    background-color: #4CAF50;
    color: white;
    text-align: center;
    padding: 10px;
    width: 100%;
}
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
}
.form-container {
    background-color: white;
    border: 1px solid #ddd;
    border-radius: 8px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
    padding: 50px;
    max-width: 600px;
    margin: 50px auto;
    text-align: center;
    position: relative;
}
.form-group {
    margin-bottom: 25px;
```

```
}
.form-group label {
    display: block;
    font-weight: bold;
    margin-bottom: 10px;
    font-size: 16px;
}
.form-group input, .form-group select {
    width: 100%;
    padding: 15px;
    border: 1px solid #ccc;
    border-radius: 5px;
    font-size: 16px;
}
.form-actions {
    display: flex;
    justify-content: center;
    align-items: center;
    gap: 20px;
    margin-top: 30px;
}
.button {
    padding: 15px 25px;
    font-size: 16px;
    font-weight: bold;
    color: white;
    background-color: #4CAF50;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    text-decoration: none;
    text-align: center;
    transition: background-color 0.3s ease;
}
.button:hover {
    background-color: #45A049;
}
.home-arrow {
    font-size: 28px;
    color: #4CAF50;
    text-decoration: none;
    cursor: pointer;
    transition: color 0.3s ease;
    position: absolute;
    top: 15px;
    left: 15px;
}
.home-arrow:hover {
    color: #45A049;
}
.form-footer {
    text-align: center;
    margin-top: 30px;
    font-size: 16px;
}
.form-footer a {
    color: #4CAF50;
    text-decoration: none;
    font-weight: bold;
}
.form-footer a:hover {
    text-decoration: underline;
```

```
}
```

/static/css/style.css

```
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    background-color: #f9f9f9;
}
header {
    background-color: #4CAF50;
    color: white;
    padding: 20px;
    text-align: center;
}
main {
    margin: 20px auto;
    max-width: 800px;
    background: white;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}
table {
    width: 100%;
    border-collapse: collapse;
    margin-top: 20px;
}
table th, table td {
    border: 1px solid #ddd;
    padding: 8px;
    text-align: left;
}

table th {
    background-color: #4CAF50;
    color: white;
}
table tr:nth-child(even) {
    background-color: #f2f2f2;
}
table tr:hover {
    background-color: #ddd;
}
.button {
    text-decoration: none;
    padding: 10px 20px;
    border-radius: 5px;
    color: white;
    background-color: #4CAF50;
    text-align: center;
    transition: background-color 0.3s;
    display: inline-block;
}
.button:hover {
    background-color: #45a049;
}
.button.cancel-button {
    background-color: #f44336;
}
.button.cancel-button:hover {
    background-color: #d32f2f;
```

```
}
.form-container {
    display: flex;
    flex-direction: column;
    gap: 15px;
}
.form-group label {
    font-weight: bold;
    margin-bottom: 5px;
}
.form-group input, .form-group select {
    padding: 10px;
    border: 1px solid #ccc;
    border-radius: 5px;
    width: 100%;
}
.form-actions {
    display: flex;
    justify-content: space-between;
    margin-top: 20px;
}
.details-container p {
    font-size: 16px;
    margin: 10px 0;
}
.details-container strong {
    color: #333;
}
.success-message {
    color: #155724;
    background-color: #d4edda;
    border: 1px solid #c3e6cb;
    padding: 10px;
    border-radius: 5px;
    margin-bottom: 20px;
    text-align: center;
}
.error-message {
    color: #721c24;
    background-color: #f8d7da;
    border: 1px solid #f5c6cb;
    padding: 10px;
    border-radius: 5px;
    margin-bottom: 20px;
    text-align: center;
}
```

application.properties

```
spring.application.name=hotel-management-api
spring.datasource.url=jdbc:postgresql://localhost:5432/hotel-management-api
spring.datasource.username=postgres
spring.datasource.password=postgres
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
spring.mvc.throw-exception-if-no-handler-found=true
```

ДОДАТОК Б

ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ

ВІДГУК

на кваліфікаційну роботу бакалавра

на тему:

«Розробка RESTful вебдодатку для управління готельним комплексом та бронювання номерів»

студента групи 121-22ск-1 Старченко Максима Павловича

Керівник економічного розділу

к.е.н., доц. каф. ПЕП та ПУ **Л.В.Касьяненко**

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	**Опис**
Пояснювальні документи	
ДП_Звіт_121-22ск-1_Старченко.docx	Пояснювальна записка до кваліфікаційної роботи. Word
ДП_Звіт_121-22ск-1_Старченко.pdf	Пояснювальна записка до кваліфікаційної роботи. PDF
Програма	
hotel-management-api.zip	Архів ZIP. Містить вихідний код проєкту та скомпільований .jar файл.
Презентація	
ДП_Презентація_121-22ск-1_Старченко.pptx	Презентація кваліфікаційної роботи