ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка web-застосунку для готельного букінгу з функціями ребукінгу та передачі бронювання»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

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

_____ Максим ФІЛАРЕТОВ
(підпис)

Виконав: здобувач вищої освіти групи ПД-41

Максим ФІЛАРЕТОВ

Керівник: Ігор ГАМАНЮК
ст. викладач

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Філаретову Максиму Юрійовичу

1. Тема кваліфікаційної роботи: «Розробка web-застосунку для готельного букінгу з функціями ребукінгу та передачі бронювання»
керівник кваліфікаційної роботи старший викладач кафедри ІПЗ Ігор ГАМАНЮК, затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості щодо процесів онлайн-бронювання готелів, принципів побудови web-застосунків та архітектурних підходів у системах клієнт-сервер.

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

1. Провести аналіз існуючих систем готельного букінгу та виявити їхні недоліки щодо гнучкості зміни броні.
2. Дослідити технологічні рішення для реалізації механізмів ребукінгу (зміна дат/номерів) та передачі бронювань між користувачами.
3. Сформувати вимоги до web-застосунку з урахуванням безпеки транзакцій та інтеграції з API Amadeus.

4. Розробити архітектуру клієнт-серверного додатку на базі Java Spring (бекенд) та сучасних веб-технологій (фронтенд).
5. Реалізувати функції: динамічний пошук готелів з фільтрами, передача бронювання.
6. Провести тестування продуктивності системи та її відповідність вимогам безпеки.

5. Перелік графічного матеріалу: *презентація*
   1. Аналіз аналогів.
   2. Вимоги до застосунку.
   3. Програмні засоби реалізації.
   4. Діаграма варіантів використання.
   5. Діаграма діяльності.
   6. Діаграма послідовності.
   7. Структура бази даних.
   8. Діаграма класів створення бронювання.
   9. Архітектура додатку.
   10. Мапи web-застосунку.
   11. Екранні форми.
   12. Апробація результатів дослідження.

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури.	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів.	05.03-13.03.2025	
3	Дослідження технологічних рішень для реалізації механізмів ребукінгу (зміна дат/номерів) та передачі бронювань між користувачами.	14.03-15.03.2024	
4	Сформувати вимоги до web-застосунку з урахуванням безпеки транзакцій та інтеграції з API Amadeus.	16.03-21.03.2024	
5	Розробка архітектуру клієнт-серверного додатку на базі Java Spring (бекенд) та сучасних веб-технологій (фронтенд).	22.03-03.04.2024	
6	Реалізувати функції: динамічний пошук готелів з фільтрами, передача бронювання.	04.04-18.04.2024	
7	Провести тестування продуктивності системи та її відповідність вимогам безпеки.	19.04-28.04.2025	

8	Оформлення роботи: вступ, висновки, реферат.	29.04-11.05.2025	
9	Розробка демонстраційних матеріалів.	12.05-18.05.2025	
10	Попередній захист роботи.	19.05-30.05.2025	

Здобувач вищої освіти _____ Максим ФІЛАРЕТОВ
(підпис)

Керівник
кваліфікаційної роботи _____ Ігор ГАМАНЮК
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 53 стор., 1табл., 27 рис., 19 джерел.

Мета роботи – покращення процесів онлайн-бронювання готелів шляхом розробки web-застосунку з механізмами передачі бронювання між користувачами.

Об'єкт дослідження – процес онлайн-бронювання готелів.

Предмет дослідження – архітектура та алгоритми web-застосунку для готельного букінгу з підтримкою передачі бронювань.

Короткий зміст роботи: У роботі проаналізовано процеси та виклики сучасного онлайн-бронювання готелів, зокрема недоліки актуальних сервісів щодо відсутності можливості передачі бронювання між користувачами. Вивчено функціональні особливості існуючих платформ (Booking.com, Airbnb, Trivago), виділено їх обмеження з точки зору гнучкості повторного або делегованого бронювання.

Розглянуто архітектурні підходи до побудови web-застосунків з урахуванням вимог масштабованості, безпеки та інтеграції зі сторонніми сервісами (Amadeus API). Розроблено архітектуру застосунку на основі Java Spring Boot, PostgreSQL та REST API, реалізовано механізми реєстрації користувачів, здійснення та передачі бронювання, обробки рейтингів та відгуків.

Описано процес реалізації клієнтської частини на HTML, CSS та JavaScript із динамічною взаємодією через API. Запропоновано алгоритми перевірки актуальності бронювання, а також автоматизованого скасування з можливістю сповіщення готелів.

Проведено ручне тестування функціоналу з використанням Postman та браузера, проаналізовано результати обміну даними між клієнтом, сервером і базою. Представлено UML-діаграми, що демонструють логіку застосунку: діаграми класів, прецедентів, активностей та послідовності викликів.

Сферою використання застосунку є системи онлайн-бронювання житла, що вимагають гнучкого управління бронюванням у випадку зміни планів користувачів.

КЛЮЧОВІ СЛОВА: ОНЛАЙН-БРОНЮВАННЯ, JAVA, SPRING BOOT, REST API, ПЕРЕДАЧА БРОНЮВАННЯ, AMADEUS API, POSTGRESQL, UML, WEB-ЗАСТОСУНОК, HTML, JAVASCRIPT.

ЗМІСТ

ВСТУП

Актуальність: Розвиток туристичної індустрії та цифрових технологій значно впливає на спосіб організації подорожей у сучасному світі. Зокрема, бронювання готелів через онлайн-платформи стало звичним і зручним способом планування поїздок як для індивідуальних користувачів, так і для корпоративних клієнтів. В умовах глобалізації попит на інтегровані та безпечні сервіси бронювання зростає, а конкуренція серед готельних платформ стимулює впровадження нових функціональних рішень.

Одним із ключових факторів успішного бронювання є можливість швидко знаходити оптимальні варіанти бронювання, мати доступ до актуальної інформації про номери, ціни, знижки та мати гнучкі інструменти керування бронюваннями. Особливої актуальності набувають такі системи, які дозволяють інтеграцію з глобальними системами дистрибуції (наприклад, Amadeus), що відкриває доступ до тисяч готелів у всьому світі в режимі реального часу. Саме такі інструменти значно підвищують якість користувацького досвіду.

У рамках дипломної роботи було розроблено вебдодаток для бронювання готелів, який поєднує адміністративну панель, панель менеджера, а також унікальну функцію — передачу бронювання іншому користувачу. Додаток реалізований із використанням Java, фреймворку Spring з модулем Spring Security для захисту, PostgreSQL як системи управління базами даних, HTML, CSS і JavaScript для фронтенду, а також API Amadeus для доступу до пропозицій готелів.

Методи дослідження: на першому етапі було проведено аналіз існуючих платформ онлайн-бронювання, таких як Booking.com, Expedia, Airbnb, з метою виявлення їх основних функціональних і нефункціональних характеристик. Після формування технічного завдання було обрано стек технологій, що задовольняє цим вимогам: Java + Spring Boot для серверної частини, PostgreSQL як СУБД, HTML/CSS/JavaScript — для клієнтської частини, Amadeus API — для отримання даних про готелі, GitHub — як система контролю версій, Figma — для проєктування інтерфейсу.

Реалізація застосунку відбувалась поетапно: від створення базової структури проекту, через реалізацію авторизації та ролей, до інтеграції з Amadeus API та впровадження унікальної функції передачі бронювання. Особлива увага приділялась забезпеченню безпеки взаємодії користувачів із системою через Spring Security.

Наукова новизна роботи полягає у поєднанні класичних механізмів бронювання з можливістю передачі бронювання іншому користувачу, що забезпечує гнучкість у плануванні подорожей. Крім того, використання Amadeus API дає змогу працювати з великою кількістю реальних пропозицій від готелів по всьому світу без потреби формування власної бази даних готелів.

Практична значущість результатів полягає у можливості використання реалізованого web-застосунку як самостійного сервісу бронювання або як компонента більших туристичних систем. Додаток є масштабованим, адаптивним до мобільних пристроїв та забезпечує зручний інтерфейс для взаємодії користувачів різного рівня.

1 АНАЛІЗ ЗАСТОСУНКУ ДЛЯ БРОНЮВАННЯ ГОТЕЛІВ

1.1 Web-застосунок для бронювання готелів

Web-застосунок для бронювання готелів — це сучасний онлайн-сервіс, розроблений для зручного пошуку, перегляду та бронювання номерів у готелях по всьому світу. Системи цього типу об'єднують у собі широкий спектр функціональних можливостей, що дозволяють як звичайним користувачам, так і менеджерам готелів чи адміністраторам ефективно керувати процесом бронювання. Головною метою подібного застосунку є надання швидкого, безпечного та персоналізованого способу взаємодії з готельним бізнесом у цифровому середовищі.

Основу web-застосунку становить гнучка архітектура з підтримкою розмежування ролей користувачів (гості, менеджери, адміністратори), а також підключення до глобальної системи бронювання — Amadeus API, що забезпечує доступ до актуальних даних про готелі, номери, наявність, ціни тощо.

Основні компоненти web-застосунку для бронювання готелів включають:

– Авторизація та автентифікація: реалізована на основі Spring Security, що дозволяє безпечно ідентифікувати користувачів і розмежовувати доступ до функціоналу відповідно до ролі (адміністратор, менеджер, гість).

– Інтеграція з Amadeus API: надає можливість виводити в реальному часі актуальні пропозиції готелів, фільтрувати за параметрами, перевіряти наявність та бронювати номери.

– Панель адміністратора: дає змогу керувати всією системою, керувати правами користувачів, переглядати статистику та звіти.

– Панель менеджера: забезпечує менеджерам готелів інструменти для перегляду заявок на бронювання, обробки замовлень, зміни статусів та комунікації з гостями.

– Унікальна функція передачі бронювання: дозволяє одному користувачу передати своє бронювання іншому (наприклад, у разі змін у планах), що є новою гнучкою опцією в організації подорожей.

Клієнтська частина: реалізована за допомогою HTML, CSS та JavaScript з адаптивним дизайном для роботи на більшості пристроїв.

Основні цілі вебзастосунку:

– Полегшення процесу бронювання номерів у готелях.

– Забезпечення актуального пошуку через глобальну систему Amadeus.

– Надання ролі-орієнтованого функціоналу для різних типів користувачів.

– Підвищення безпеки обробки особистих даних і платежів.

– Впровадження додаткових можливостей (як-от передача бронювання) для розширення зручності користувачів.

Переваги вебзастосунку для бронювання готелів:

– Доступність: користувачі можуть здійснювати пошук і бронювання номерів у будь-який час і з будь-якого пристрою, підключеного до Інтернету.

– Гнучкість: розмежування доступу за ролями дозволяє адаптувати інтерфейс і функціонал до потреб конкретного користувача (гостя, менеджера, адміністратора).

– Інтеграція з реальними пропозиціями: завдяки Amadeus API система має

доступ до актуальної інформації про тисячі готелів по всьому світу.

– Автоматизація: зменшує потребу в ручній обробці бронювань, прискорює обробку замовлень.

– Безпека: завдяки Spring Security реалізовано захист даних користувачів, ролей і доступу до приватної інформації.

Потенційні недоліки або обмеження:

– Необхідність стабільного підключення до Інтернету як для кінцевих користувачів, так і для API-запитів до Amadeus.

– Залежність від зовнішнього API (Amadeus), що вимагає дотримання політики їхньої роботи та обмежень.

– Технічна складність розробки: інтеграція глобальних API, реалізація багаторольового доступу й безпеки вимагає значних ресурсів на етапі розробки та тестування.

– Самостійне керування бронюваннями може вимагати навчання або адаптації з боку користувачів, що не звикли до таких систем.

1.2 Специфіка розробки web-застосунку для бронювання готелів

Розробка web-застосунку для бронювання готелів має низку особливостей, які відрізняють цей тип програмного забезпечення від інших типових вебпроєктів. Специфіка полягає як у функціональних вимогах, так і в технічній архітектурі системи.

Інтеграція із зовнішніми сервісами:

– Застосунок повинен взаємодіяти з глобальною системою бронювання (Amadeus API), що забезпечує актуальні дані про готелі, вільні номери, ціни та додаткові послуги.

– Реалізація інтеграції вимагає обробки великої кількості запитів, автентифікації через токени та роботи з REST API.

Авторизація і безпека:

– Для забезпечення конфіденційності даних користувачів та бронювань необхідно реалізувати багаторівневу систему авторизації з розмежуванням прав доступу на основі ролей (гості, менеджери, адміністратори).

– Spring Security забезпечує надійний захист від несанкціонованого доступу, CSRF-атак, XSS та інших загроз.

Структура даних і БД:

– Необхідно проектувати базу даних PostgreSQL так, щоб вона підтримувала складні зв’язки між готелями, номерами, бронюваннями, користувачами, платежами тощо.

– Важливо враховувати оптимізацію запитів та масштабованість для обробки великої кількості записів.

Кросплатформеність і UI/UX:

– Інтерфейс має бути адаптованим під десктопи, планшети та смартфони.

– Важливо враховувати простоту навігації, швидке завантаження сторінок та логічну послідовність дій користувача.

Унікальна функція передачі бронювання:

– Однією з ключових інновацій стала реалізація функції передачі активного бронювання іншому користувачеві, що вимагає додаткової логіки перевірки прав і змін у базі даних.

Панелі для ролей:

– Панель адміністратора повинна включати управління користувачами, перегляд статистики та модерацію контенту.

– Панель менеджера — управління заявками, бронюваннями, повідомленнями.

– Користувач-гість має доступ до перегляду готелів, фільтрів, перегляду історії та створення бронювань.

Таким чином, розробка web-застосунку для бронювання готелів вимагає комплексного підходу з урахуванням як функціонального, так і технологічного рівня реалізації.

1.3 Цільова аудиторія

Web-застосунок для бронювання готелів орієнтований на різні групи користувачів, кожна з яких має власні цілі, завдання та очікування від системи. Сегментація користувачів дозволяє більш ефективно реалізовувати функціонал для кожної категорії.

Мандрівники та туристи — найбільша група кінцевих користувачів, які використовують систему для бронювання готелів під час відпусток, ділових поїздок, подорожей на відпочинок. Основні очікування: простота використання, швидкий пошук, безпечне бронювання, можливість зміни або передачі бронювання.

Менеджери готелів — представники готельного бізнесу, які керують доступністю номерів, обробляють заявки, підтверджують або скасовують бронювання. Для них важлива наявність зручного кабінету менеджера з функціями повідомлень, керування пропозиціями та перегляду аналітики.

Адміністратори системи — персонал, який керує структурою web-застосунку, правами доступу, користувачами та вмістом. Їх головна мета — забезпечення стабільної роботи системи, безпеки даних та доступу.

Корпоративні користувачі — компанії або агентства, які здійснюють групові бронювання або регулярно замовляють послуги проживання для своїх співробітників.

Завдяки гнучкій архітектурі застосунок може масштабуватися під потреби нових груп користувачів, включаючи B2B-клієнтів, агентів, партнерські програми тощо.

1.4 Дослідження існуючих аналогів

На сьогоднішній день одними з найпопулярніших вебсайтів для онлайн-бронювання готелів є Booking.com, Airbnb та Agoda. Розглянемо їх детальніше.

1.4.1 Booking.com

Booking.com — одна з найбільших онлайн-платформ для бронювання житла, яка дозволяє мандрівникам швидко знаходити готелі, апартаменти, хостели та інші види проживання. Платформа вирішує проблему зручного пошуку, порівняння цін та миттєвого бронювання, що значно спрощує планування подорожей.

Основні функції Booking.com включають:

– Пошук житла з фільтрами (ціна, рейтинг, тип проживання, зручності, розташування).
– Інтерактивна карта для візуального орієнтування.
– Реальні відгуки користувачів.
– Миттєве бронювання без потреби в погодженні з власником.
– Мобільний додаток для Android та iOS.
– Програма лояльності Genius.
– Багатомовна підтримка інтерфейсу.

Переваги:

– Великий вибір варіантів по всьому світу.
– Гнучкі умови скасування.
– Швидкий та інтуїтивний процес бронювання.
– Надійна служба підтримки.

Недоліки:

– Комісії для партнерів (власників житла).
– Перевантажений інтерфейс для новачків.
– Іноді приховані додаткові збори до завершення бронювання.

На рисунках 1.1 – 1.2 зображено сторінкі сайту Booking.com.

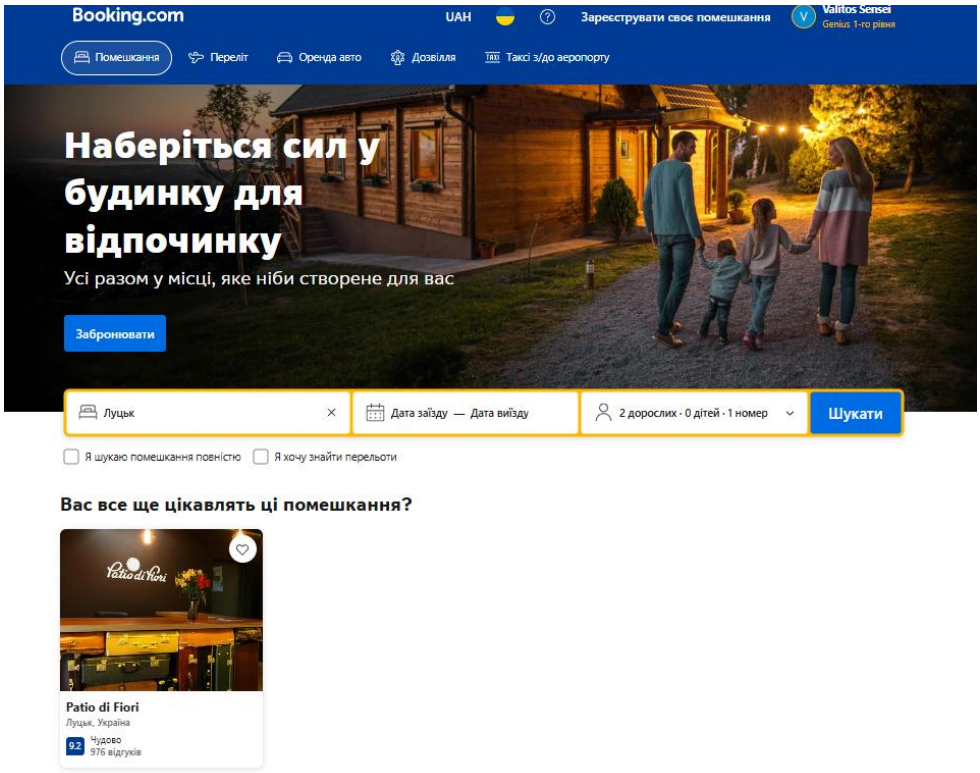


Рис. 1.1 Приклад сторінки пошуку житла на Booking.com

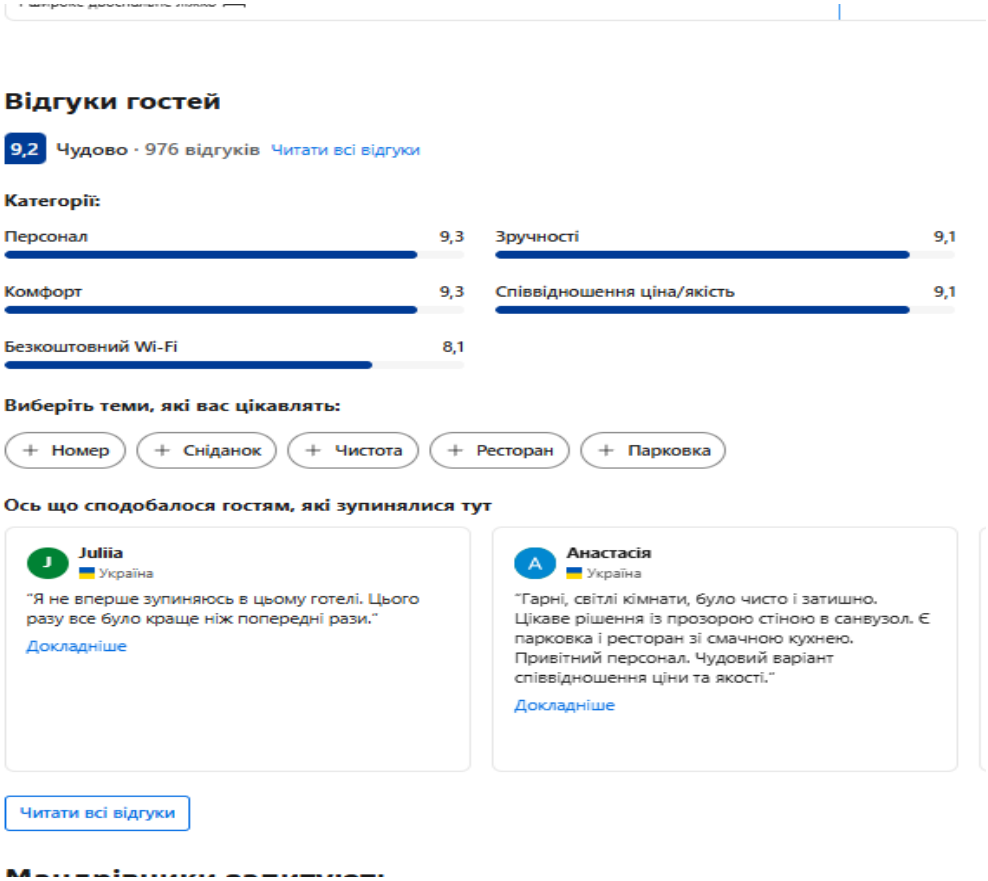


Рис. 1.2 Інтерфейс сторінки готелю з відгуками

1.4.2 Airbnb

Airbnb — це онлайн-платформа для оренди приватного житла напряму від власників. Вона пропонує мандрівникам унікальний досвід проживання у квартирах, будинках, віллах та навіть незвичайних локаціях, таких як будиночки на деревах чи човни.

Основні функції Airbnb:

- Можливість орендувати житло напряму від господаря.
- Інтерактивна карта з розміщенням об’єктів.
- Доступ до особистого спілкування з власником.
- Відгуки гостей після кожного перебування.
- Система рейтингів для господарів та гостей.
- Мобільний застосунок для зручного управління бронюванням.
- Категорії житла (наприклад, “домівки з басейном”, “підходить для тварин”).

Переваги:

- Більше індивідуальних та оригінальних варіантів.
- Часто дешевше за готелі при довгостроковому проживанні.
- Досвід проживання як “місцевий”.

Недоліки:

- Менше стандартів якості — залежить від конкретного господаря.
- Складніші правила скасування.
- Деякі оголошення можуть бути фейковими (хоча Airbnb бореться з цим).
- Комісія для гостей може бути досить високою.

На рисунках 1.3 - 1.4 зображено інтерфейс Airbnb.

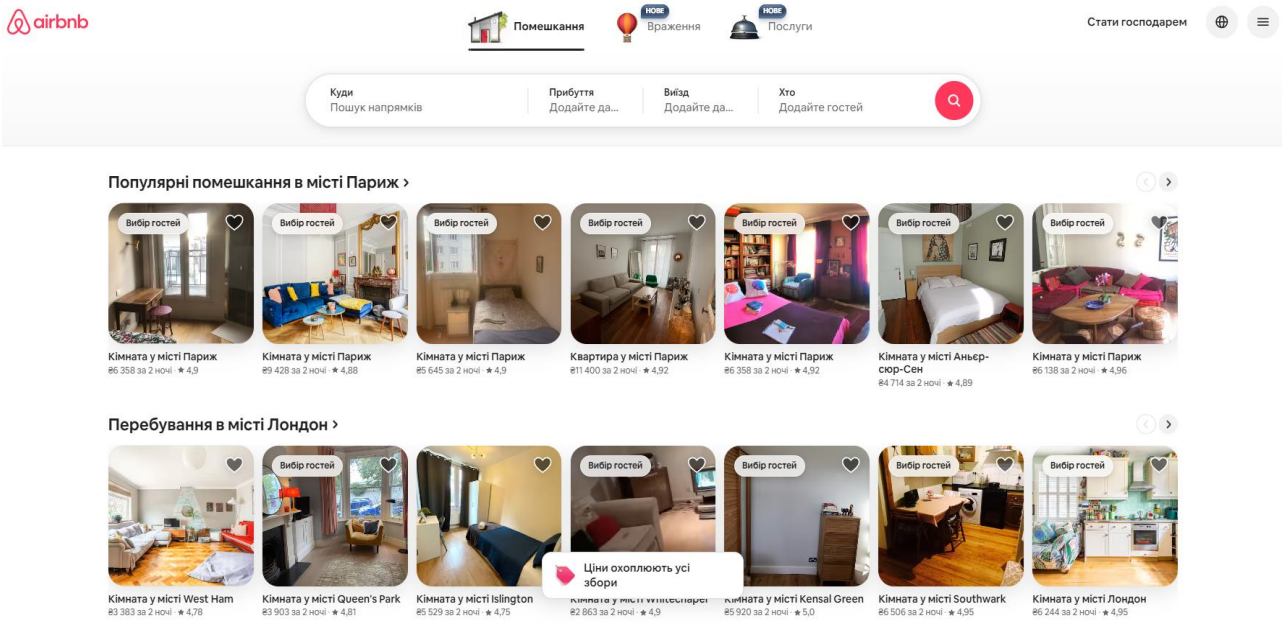


Рис. 1.3 Головна сторінка Airbnb з пошуком

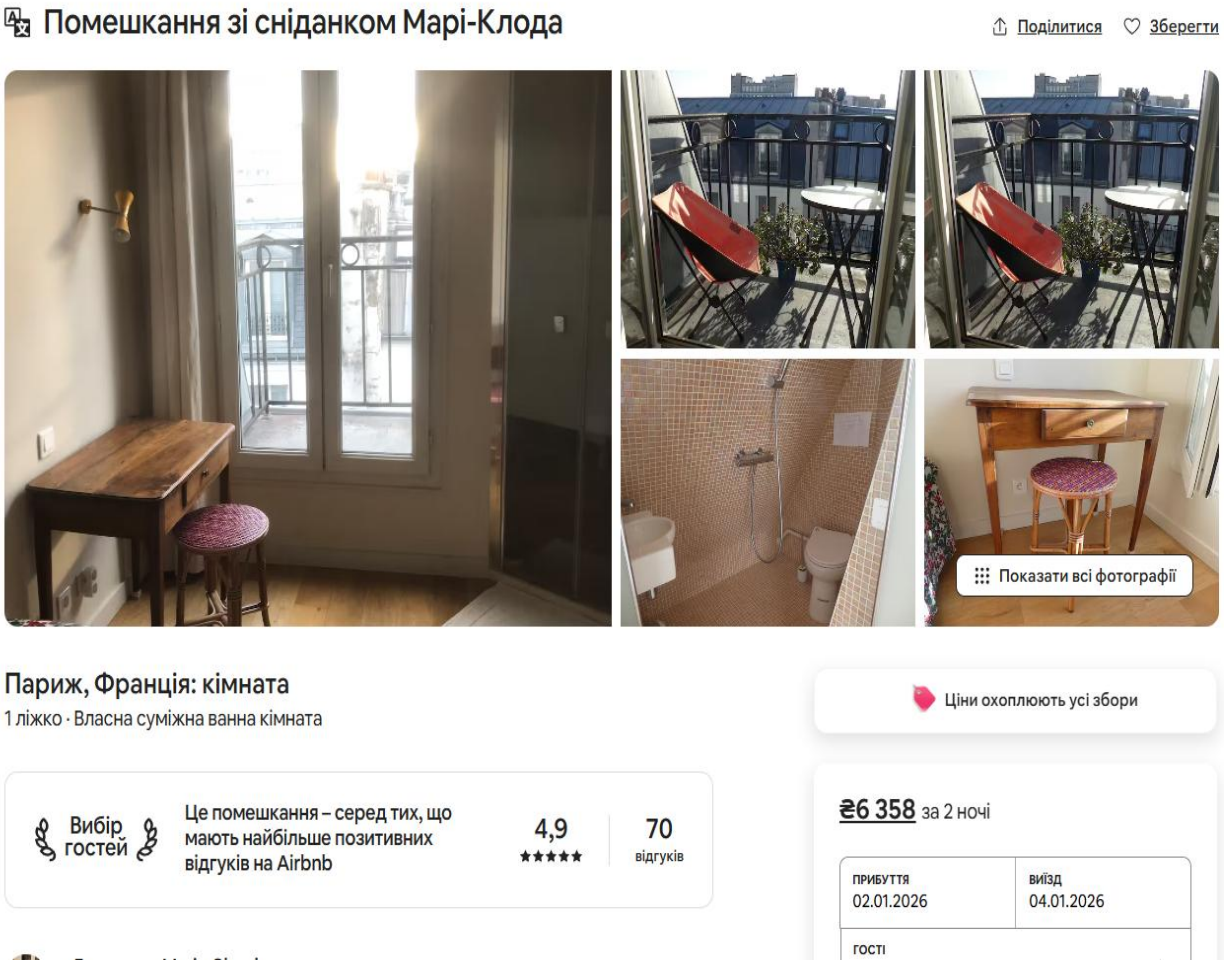


Рис. 1.4 Приклад помешкання з фото, описом і цінами

1.4.3 Agoda

Agoda — ще одна велика платформа для бронювання готелів, яка особливо популярна в Азії. Вона орієнтована як на туристів, так і на ділових мандрівників, і пропонує гнучкі умови та часто нижчі ціни завдяки промоакціям.

Основні функції Agoda:

- Пошук готелів, апартаментів та вілл з різними параметрами.
- Програма винагород (AgodaCash).
- Додаткові знижки для користувачів мобільного додатку.
- Функція «Таємні угоди» (Secret Deals).
- Розширені фільтри: за зірками, зручностями, відстанню тощо.
- Перегляд відгуків та фото користувачів.

Переваги:

- Вигідні пропозиції, особливо в країнах Південно-Східної Азії.
- Часто нижчі ціни, ніж у конкурентів.
- Широкий вибір місць проживання.
- Можливість бронювання без передоплати.

Недоліки:

- Менш зручний інтерфейс у порівнянні з Booking.com.
- Обмежений вибір у деяких регіонах Європи та Америки.
- Повільна технічна підтримка в окремих випадках.

На рисунках 1.5 - 1.6 зображено інтерфейс Agoda.

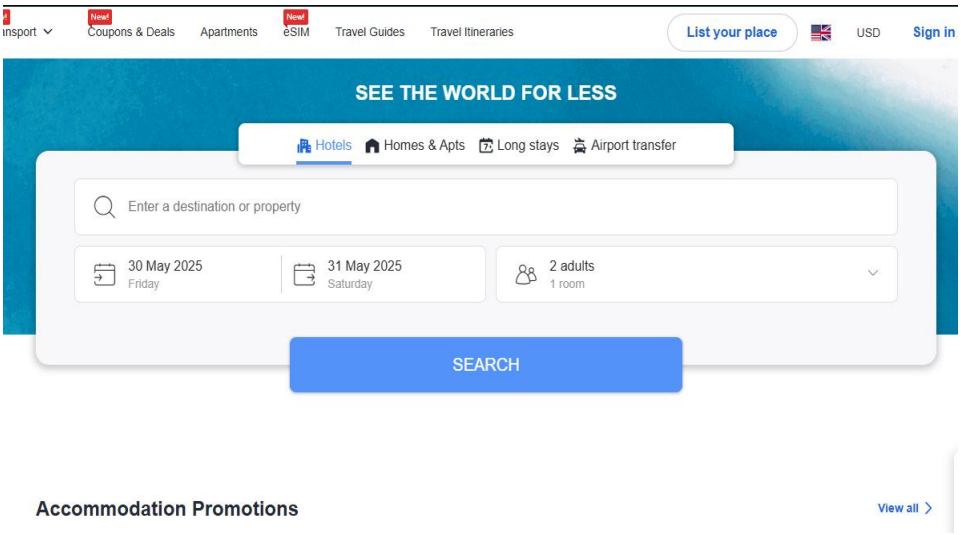


Рис. 1.5 Інтерфейс пошуку житла на Agoda

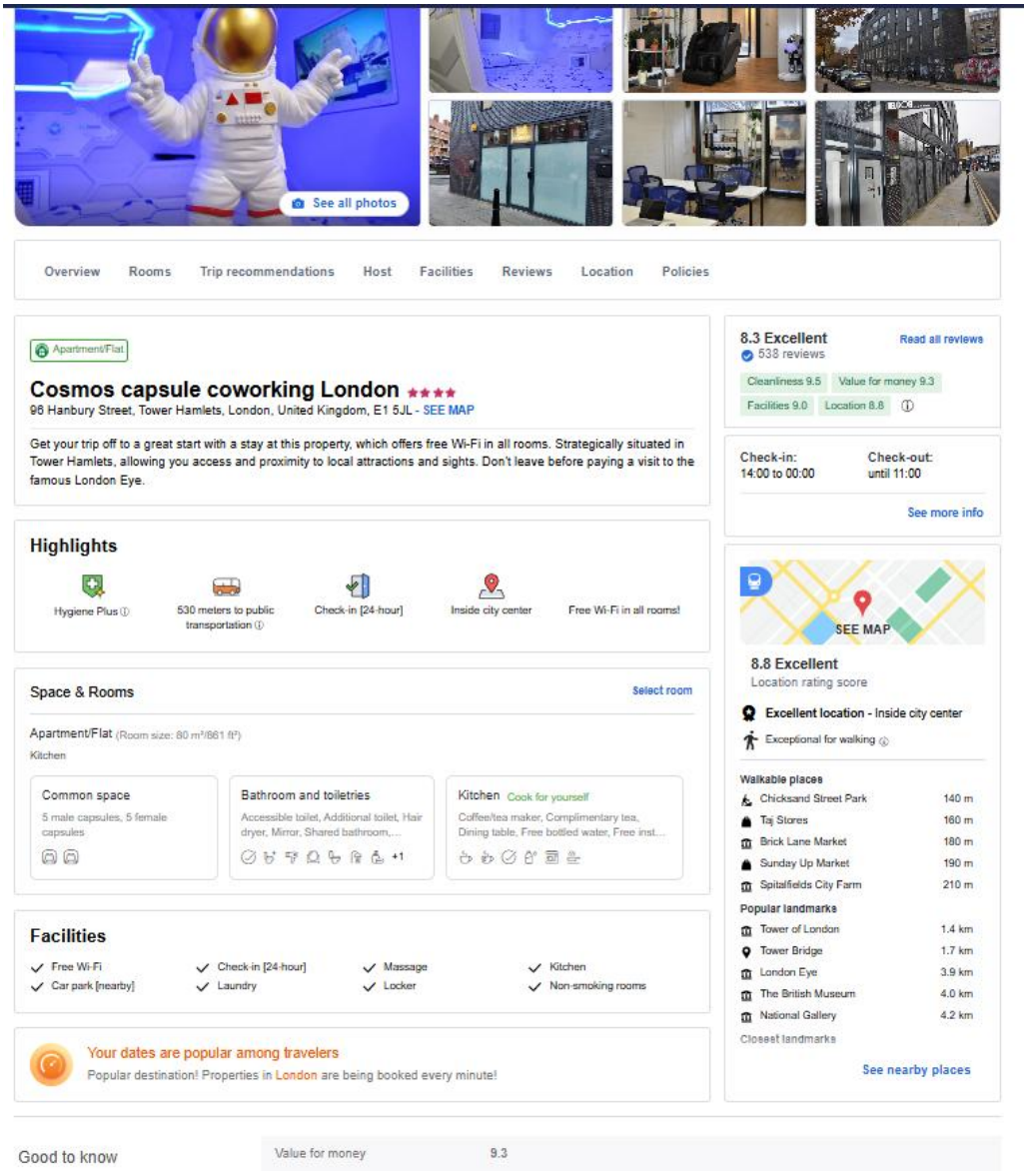


Рис. 1.6 Приклад сторінки з описом готелю та знижками

В таблиці 1.1 зведено результати аналізу існуючих застосунків для бронювання готелів.

Таблиця 1.1

Зведена таблиця аналізу існуючих платформ для бронювання готелів та їх порівняння

Показник	Booking.com	Airbnb	Agoda
Типи житла	Готелі, апартаменти, хостели, курорти, вілли та інші варіанти проживання	Приватні квартири, будинки, незвичайне житло (будиночки на деревах, човни тощо)	Готелі, апартаменти, вілли, курорти
Функція пошуку та фільтрація	Пошук за ціною, розташуванням, зручностями, типом житла, зірками	Пошук за ціною, типом житла, зручностями, відстанню від визначених локацій	Пошук за ціною, типом житла, зручностями, спеціальними пропозиціями
Цінова політика	Вартість залежить від типу житла та місця розташування, наявні знижки та акції	Ціни можуть змінюватися в залежності від господаря, наявність знижок	Конкурентні ціни, часто наявність акцій та знижок для мобільного додатка
Програма лояльності	Програма Genius з персональними знижками і перевагами для постійних користувачів	Відсутня	Програма винагород AgodaCash для знижок

Продовження таблиці 1.1

Зведена таблиця аналізу існуючих платформ для бронювання готелів та їх порівняння

Показник	Booking.com	Airbnb	Agoda
Мобільний додаток	Доступний для iOS і Android, з усіма функціями для бронювання та скасування	Доступний для iOS і Android, інтеграція з функціями чату та перегляду місць	Доступний для iOS і Android, з можливістю отримання додаткових знижок
Розмовна підтримка	Онлайн чат, телефонна підтримка на різних мовах	Чат з господарями для уточнення деталей, підтримка через додаток	Онлайн чат, телефонна підтримка, підтримка різних мов
Оцінка та відгуки користувачів	Рейтинг з відгуками після кожного перебування	Рейтинг від користувачів, коментарі господарів та гостей	Рейтинг з коментарями, оцінки гостей, відгуки про конкретне житло
Особливості	Широкий вибір варіантів по всьому світу, зручний інтерфейс, багатомовна підтримка	Унікальні варіанти житла, взаємодія з господарями, часто менш формальне житло	Широкий вибір, дешевші варіанти в Азії, вигідні пропозиції через мобільний додаток
Недоліки	Високі комісії для партнерів, інтерфейс може бути перевантажений	Досить високі комісії для гостей, може бути важко з'єднатися з господарем	Обмежений вибір у деяких регіонах, складнощі з інтерфейсом

1.5 Визначення вимог до застосунку

Проаналізувавши сутність застосунку для букінгу готелів, його специфіку, цільову аудиторію та аналогів, було визначено наступні вимоги до застосунку:

– Авторизація: реєстрація нових користувачів, вхід та вихід із системи для персоналізації користувацького досвіду.

– Бібліотека готелів та фільтрація пропозицій: зберігання та управління даними про готелі, включаючи їхні характеристики, рейтинги.

– Сторінка перегляду готелю: детальне відображення інформації про готель, з можливістю перегляду фотографій, опису номерів, відгуків користувачів, та іншої важливої інформації.

– Процес бронювання: можливість забронювати готель за кілька простих кроків, з автоматичним оновленням статусу заброньованих номерів та наданням електронних підтверджень.

– Швидкий та інтуїтивний інтерфейс користувача: оптимізовані сторінки. Інтерфейс повинен бути чітким, адаптивним та зручним у використанні на всіх пристроях.

– Безпека та захист даних: забезпечення конфіденційності табезпеки персональних даних користувачів, з використанням сучасних методів шифрування та захисту транзакцій.

– Система відгуків та рейтингів: можливість залишати відгуки про готелі, оцінювати якість сервісу, що дозволяє іншим користувачам робити обґрунтований вибір.

2 ЗАСОБИ РЕАЛІЗАЦІЇ

2.1 Середовище розробки

Інтегроване середовище розробки (IDE) — це програма, яка об'єднує різні інструменти, потрібні програмісту для створення програм. Зазвичай до складу такого середовища входить редактор коду, засоби для збирання проєкту, виявлення помилок (налагодження) та іноді можливість роботи з системами керування версіями. Це дає змогу зручно писати, перевіряти та вдосконалювати код в одному місці.

2.1.1 Intelij IDEA

IntelliJ IDEA — це зручне середовище для розробки програм, яке часто використовують під час написання коду на Java, Kotlin, Scala та деяких інших мовах. Зважаючи на те, що розробка застосунку для букінгу готелів вимагала високої продуктивності, стабільності та інтеграції з різними технологіями, вибір Intelij IDEA став оптимальним. Це середовище забезпечує потужні інструменти для відлагодження, профілювання, автоматизації тестів і взаємодії з базами даних, що робить його ідеальним для розробки складних програмних рішень, таких як системи бронювання.

Intelij IDEA має низку переваг для розробки застосунків, зокрема:

Підтримка багатьох мов програмування: окрім Java, IDE підтримує Python, Kotlin, JavaScript, TypeScript, SQL та інші мови. Це дозволяє легко інтегрувати різні технології для розробки web- і мобільних застосунків.

Інтеграція з базами даних: для роботи з базами даних, що використовуються для збереження інформації про готелі, бронювання та користувачів, Intelij IDEA має вбудовану підтримку SQL та зручні інструменти для роботи з базами даних, що суттєво прискорює процес розробки.

Підтримка систем контролю версій (Git, SVN): завдяки інтеграції з системами контролю версій, такими як Git, розробники можуть зручно відслідковувати зміни в коді та співпрацювати в команді.

Підтримка фреймворків для розробки web-застосунків: IntelliJ IDEA підтримує популярні веб-фреймворки, такі як Spring, що робить розробку сервера для букінгу готелів швидким і зручним процесом.

2.1.2 Переваги використання Intelij IDEA для розробки застосунку для букінгу готелів

Вибір Intelij IDEA для створення застосунку для букінгу готелів зумовлений її потужними інструментами для розробки та зручністю інтеграції з різними технологіями. Основні переваги включають:

Продуктивність і стабільність: Intelij IDEA забезпечує високу продуктивність, що дозволяє ефективно працювати з великими обсягами даних про готелі та бронювання.

Зручність налаштувань для web-застосунків: вбудовані інструменти для розробки веб-застосунків дозволяють швидко налаштувати сервер, підключити базу даних і забезпечити швидкий доступ до необхідної інформації.

Підтримка роботи з різними технологіями: IntelliJ IDEA легко поєднується з популярними інструментами для тестування, аналізу продуктивності та контролю якості, що робить її зручним вибором для розробки надійних і масштабованих застосунків.

2.2 Мови програмування

Мова програмування — це формалізована мова, призначена для комунікації інструкцій до машини, зокрема комп'ютера. Вона використовується для створення програм, які виконують алгоритми. Більшість мов програмування складається з інструкцій для комп'ютерів. Вони дозволяють розробникам ефективно створювати програми, виражаючи розрахунки та управління даними.

2.2.1 Java

Java — це об'єктно-орієнтована, високорівнева мова програмування, яка є однією з найбільш популярних для розробки великих та складних програмних рішень. Вона використовується для розробки як серверної частини, так і клієнтських додатків, включаючи веб-сайти та мобільні застосунки. Однією з основних переваг Java є її можливість працювати на різних платформах завдяки принципу "Write Once, Run Anywhere" (WORA), що забезпечує платформну незалежність.

У розробці застосунку для букінгу готелів Java використовувалася для серверної частини додатка, де потрібно обробляти запити користувачів, зберігати інформацію про готелі, бронювання, а також здійснювати взаємодію з базою даних для пошуку та фільтрації варіантів. Java є стабільною та ефективною мовою для створення таких серверних рішень завдяки своїй багатофункціональності, масштабованості та підтримці великої кількості бібліотек та фреймворків, зокрема Spring Framework.

2.2.2 HTML і CSS

HTML (HyperText Markup Language) — це основна мова, що використовується для побудови структури веб-сторінок. Вона допомагає організовувати вміст на сайті. А CSS (Cascading Style Sheets) відповідає за оформлення сторінок — з її допомогою задаються кольори, шрифти, розташування елементів та загальний вигляд інтерфейсу.

У розробці застосунку для букінгу готелів HTML і CSS використовувалися для створення адаптивного інтерфейсу користувача. HTML дозволяє визначати структуру сторінок, в той час як CSS забезпечує оформлення елементів інтерфейсу, таких як форми для пошуку готелів, вибір дат заїзду і виїзду, відображення доступних варіантів готелів. CSS також забезпечує мобільну адаптивність інтерфейсу, що є важливим для користувачів, які шукають готелі через мобільні пристрої.

2.2.3 JavaScript

JavaScript — це популярна мова програмування, яка часто застосовується для додавання інтерактивності на веб-сайтах, наприклад, для обробки подій, анімацій або динамічного оновлення контенту. JavaScript дозволяє здійснювати асинхронні запити, обробляти події користувача та змінювати контент сторінки без її перезавантаження. Завдяки цьому JavaScript робить веб-сторінки динамічними та інтерактивними.

У застосунку для букінгу готелів JavaScript використовувався для створення інтерактивних елементів на сторінках, таких як календарі для вибору дат заїзду та виїзду, фільтри для сортування результатів пошуку, а також обробки подій, коли користувач вибирає певні параметри, наприклад, тип готелю чи кількість гостей. JavaScript активно використовувався для покращення досвіду користувача, забезпечуючи швидку та безперебійну взаємодію з додатком без необхідності постійно перезавантажувати сторінки.

2.3 Веб-фреймворки

Веб-фреймворк — це спеціальний набір програмних інструментів, який допомагає розробникам створювати веб-додатки, зокрема веб-сервіси, сайти та API. Завдяки чітко визначеній структурі та готовим рішенням фреймворки дозволяють значно скоротити час на реалізацію функціоналу, забезпечити кращу підтримку коду та дотримання загальноприйнятих стандартів. Крім того, вони часто включають засоби для обробки запитів, маршрутизації, автентифікації, підключення до баз даних та іншого функціоналу, що робить розробку більш простою та ефективною.

2.3.1 Spring

Spring — це потужний, гнучкий і популярний Java-фреймворк, який активно використовується для створення корпоративних додатків. Він надає широкий набір інструментів та бібліотек для розробки веб-додатків, а також для інтеграції з базами даних, обробки запитів та безпеки. Однією з основних переваг Spring є його

підтримка архітектурного стилю MVC (Model-View-Controller), що робить розробку додатків більш структурованою і зрозумілою.

У рамках розробки застосунку для букінгу готелів Spring використовувався для побудови серверної частини додатка, обробки запитів користувачів, а також для роботи з базами даних. Зокрема, використання Spring Boot дозволяє швидко налаштовувати та розгортати додатки, а Spring Data забезпечує простоту взаємодії з реляційними базами даних. Це дозволяє ефективно управляти даними про готелі, бронювання, ціни та інші важливі аспекти функціонування платформи.

2.3.2 Amadeus API

Amadeus — це одна з провідних світових компаній, що надає API для розробки додатків у сфері туристичних послуг. У разі розробки застосунку для букінгу готелів, Amadeus API є потужним інструментом для інтеграції з глобальними системами бронювання, що дозволяє автоматизувати процеси пошуку готелів, порівняння цін, отримання інформації про наявність номерів, а також обробки платежів.

Використання Amadeus API в проекті дає змогу надати користувачам доступ до актуальної інформації щодо варіантів готелів по всьому світу, зручний інтерфейс для бронювання та оплату в межах одного додатка, а також забезпечує точність і актуальність даних.

2.4 Система управління базою даних

PostgreSQL, або Postgres, є однією з найпотужніших відкритих систем управління базами даних (СУБД), що використовуються для зберігання та обробки великих обсягів даних. Це об'єктно-реляційна СУБД, відома своєю надійністю та гнучкістю, а також здатністю ефективно працювати з великими наборами даних.

Для мого проекту, де основною метою є розробка застосунку для букінгу готелів, PostgreSQL була обрана для забезпечення зберігання і управління всіма даними, такими як інформація про готелі, номери, бронювання та користувачі. Система підтримує складні запити, зв'язки між таблицями через зовнішні ключі, індекси для прискорення пошуку, а також транзакції, що забезпечують цілісність та безпеку даних.

PostgreSQL також має потужну підтримку роботи з JSON, що дозволяє зберігати та обробляти дані у форматі JSON безпосередньо в базі. Це зручно для інтеграції з API та збереження налаштувань користувачів або параметрів бронювання, що надходять у вигляді JSON.

2.5 Система контролю версій

Git — це інструмент для керування версіями коду, який дає змогу розробникам зберігати історію змін, працювати над одним проєктом у команді та швидко повертатися до попередніх версій у разі потреби. Завдяки Git можна ефективно контролювати процес розробки, зменшувати ризики втрати даних і легко поєднувати роботу кількох учасників проєкту. Для мого проекту я використовував Git для зберігання і управління кодом на GitHub, що дозволяє здійснювати контроль версій, співпрацювати з іншими розробниками та забезпечує централізоване зберігання даних про проект.

GitHub, на базі Git, використовується для хостингу репозиторіїв, де зберігаються всі версії коду мого застосунку. Цей інструмент дозволяє створювати публічні або приватні репозиторії, виконувати пул-реквести для внесення змін, а також відстежувати зміни за допомогою гілок та комітів, що полегшує розробку та тестування.

2.6 Інструменти проєктування інтерфейсу користувача

Оскільки я не використовував специфічні інструменти для моделювання вигляду сайту, а напряму писав HTML, CSS та JavaScript, головною метою було створення простого та функціонального інтерфейсу. Цей підхід дозволив мені реалізувати бажаний вигляд та поведінку сайту безпосередньо в коді, забезпечивши адаптивність для різних пристроїв, включаючи десктопи та мобільні пристрої.

Для стилізації сторінок використовувався CSS, який дозволив створити привабливий та інтуїтивно зрозумілий інтерфейс для користувачів. JavaScript, у свою чергу, забезпечив інтерактивність, дозволяючи реалізувати динамічні елементи, такі як форми бронювання, обробка запитів та інші інтерактивні компоненти.

3 ПРОЄКТУВАННЯ WEB-ЗАСТОСУНКУ

3.1 Проєктування архітектури застосунку

Мій застосунок для букінгу готелів реалізовано за моделлю клієнт-сервер, що дозволяє ефективно розподіляти навантаження між клієнтською та серверною частинами. Це забезпечує покращену масштабованість, ефективність і зручність для користувачів.

Клієнтська частина (фронтенд) відповідає за взаємодію з користувачем через графічний інтерфейс. Фронтенд розроблений за допомогою стандартних веб-технологій, таких як HTML, CSS і JavaScript. Користувач взаємодіє з інтерфейсом для перегляду доступних готелів, бронювання номерів і здійснення платежів. Всі запити до сервера відправляються через HTTP-протокол. Для динамічного оновлення даних без перезавантаження сторінки використовуються технології AJAX і JSON. Це створює плавний користувацький досвід і забезпечує високу інтерактивність web-застосунку.

Серверна частина (бекенд) розроблена з використанням Java-фреймворку Spring для обробки запитів клієнта, управління бізнес-логікою та забезпечення роботи з базою даних. Сервер приймає запити від фронтенду, обробляє їх і взаємодіє з PostgreSQL для зберігання та вилучення даних про готелі, користувачів, бронювання і платіжні транзакції. Крім того, бекенд відповідає за аутентифікацію та авторизацію користувачів, а також забезпечує захист даних під час транзакцій.

Для покращення безпеки та взаємодії з користувачами я використовую Spring Security, що допомагає реалізувати механізми безпеки, зокрема управління правами доступу, захист від атак типу CSRF та авторизацію через JWT токени.

Інтеграція з сторонніми сервісами здійснюється через API, що дозволяє додавати додаткові функції, такі як перевірка доступності номерів через сторонні платформи або інтеграція з платіжними системами для здійснення оплати за допомогою сторонніх провайдерів.

На рисунку 3.1 зображено діаграму прецендантів.

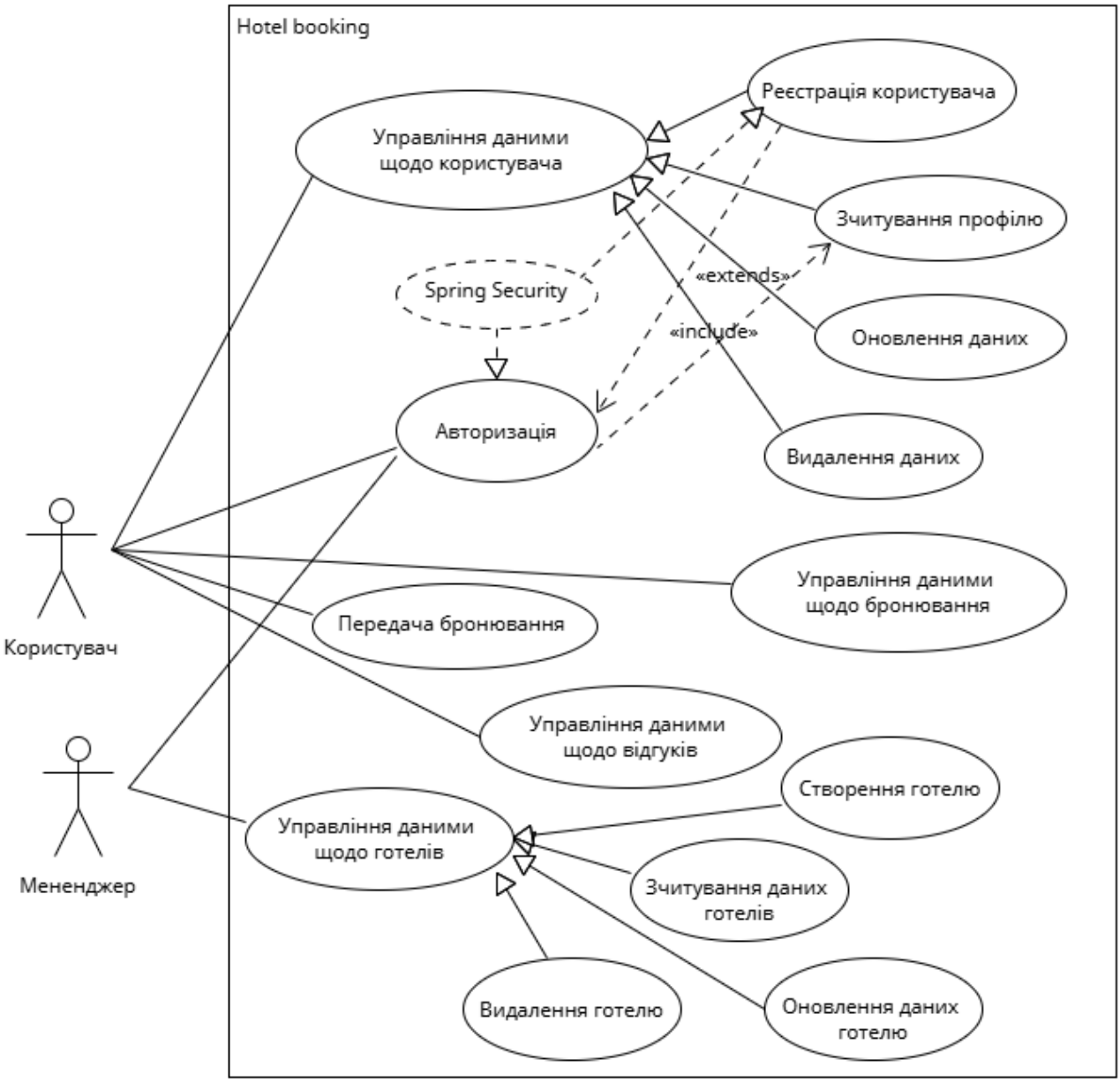


Рис. 3.1 Діаграма прецидентів

3.2 Архітектура клієнтської частини

Структура клієнтської частини застосунку для букінгу готелів організована за допомогою стандартних веб-технологій — HTML, CSS та JavaScript, без використання фреймворків, таких як Angular. Такий підхід забезпечує гнучкість і простоту реалізації функціоналу без складної інтеграції сторонніх бібліотек.

Для організації коду використовуються три основні компоненти:

Компоненти:

Кожен елемент інтерфейсу, наприклад, форма для пошуку готелів, кнопки бронювання, списки результатів пошуку, є окремим компонентом. Кожен компонент відповідає за свою частину інтерфейсу користувача та має відповідний HTML для розмітки, CSS для стилізації та JavaScript для управління поведінкою. Це дозволяє розробникам ефективно створювати та керувати елементами, що можуть бути повторно використані в різних частинах застосунку, зокрема на сторінках пошуку готелів, профілях користувачів та на сторінках бронювання.

Сторінки:

Сторінки формують окремі відображення, які користувач бачить у веб-додатку. Наприклад, сторінка пошуку готелів складається з таких компонентів, як поле для введення пошукових запитів, фільтри для сортування результатів, а також відображення списку доступних готелів. Кожна сторінка складається з різних елементів, що утворюють повний інтерфейс для користувача. Сторінки дозволяють зберігати чітку структуру навігації та зручний доступ до функціоналу.

Сервіси:

Сервіси відповідають за бізнес-логіку додатка та взаємодію з сервером. Вони обробляють запити користувачів, обирають потрібні дані з бази даних PostgreSQL, здійснюють запити до серверних API для перевірки наявності готелів та бронювання. Сервіси використовуються для обробки даних, взаємодії з платіжними системами та інших повторно використовуваних функцій, таких як відображення результатів пошуку або бронювання. Вони можуть бути викликані з компонентів для інтеграції з бекендом, забезпечуючи ефективне управління та розширюваність коду.

Загальну логіку запиту користувача зображено на рисунку 3.2.

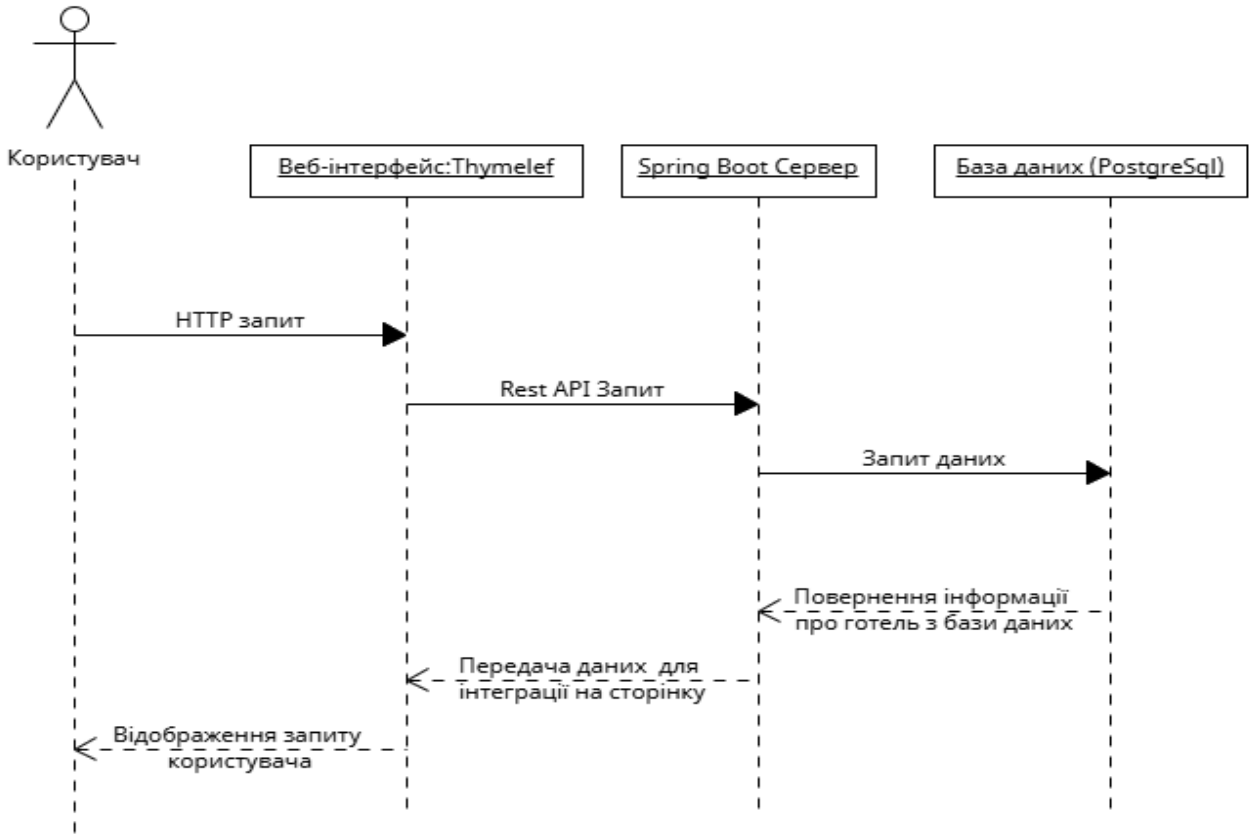


Рис 3.2 Діаграма запиту користувача

3.3 Архітектура серверної частини

Серверна частина web-застосунку для букінгу готелів реалізована з використанням фреймворку Spring Boot, який є частиною екосистеми Spring. Spring Boot забезпечує швидку конфігурацію, інверсію управління залежностями та гнучку архітектуру, що дозволяє ефективно реалізовувати масштабовані веб-додатки. Архітектура побудована на принципах шарової моделі, де кожен рівень відповідає за окремі завдання: обробку запитів, бізнес-логіку та взаємодію з базою даних.

Основні пакети серверної частини:

Controllers (Контролери):

Контролери відповідають за обробку HTTP-запитів від клієнта. Вони реалізовані за допомогою анотацій Spring MVC (@RestController, @GetMapping, @PostMapping тощо). Контролери отримують вхідні дані, передають їх у сервісний

шар для обробки, а потім формують відповіді, зазвичай у форматі JSON. Наприклад, запит на пошук готелів надходить у контролер, який звертається до відповідного сервісу, що взаємодіє з Amadeus API та повертає результати користувачу.

Services (Сервіси):

Сервісний шар містить бізнес-логіку застосунку. Тут відбувається обробка даних, взаємодія з Amadeus API для пошуку та бронювання готелів, логіка перевірки доступності номерів, обробка платежів тощо. Цей підхід дозволяє ізолювати логіку від контролерів та забезпечити зручне тестування й повторне використання коду.

Repositories (Репозиторії):

Для взаємодії з базою даних використовується Spring Data JPA. Репозиторії інкапсулюють SQL-запити та надають простий інтерфейс для зберігання та отримання даних з PostgreSQL. Наприклад, інформація про бронювання, користувачів або готелі зберігається у відповідних таблицях і доступна через JPA-репозиторії.

Entities (Сутності):

Це Java-класи, які відображають таблиці з бази даних PostgreSQL. Вони позначаються спеціальними анотаціями, такими як @Entity та @Table, що дає змогу Spring автоматично пов’язувати класи з відповідними таблицями та виконувати перетворення між об'єктами й базою даних (ORM — об'єктно-реляційне відображення). Наприклад, класи Hotel, Booking, User відповідають основним об'єктам у логіці програми.

На рисунку 3.3 зображено діаграму класів серверної логікі.

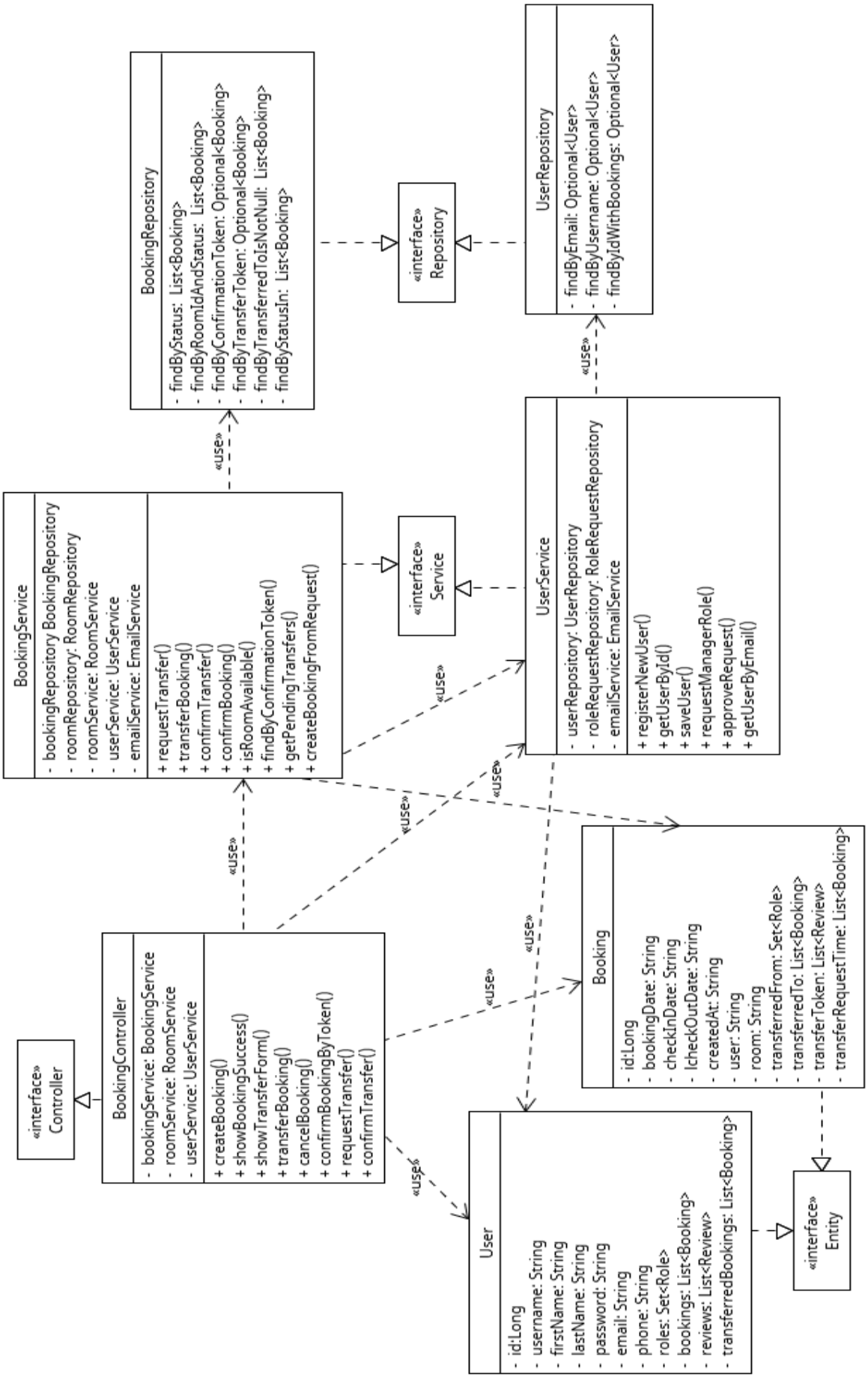


Рис 3.3 Діаграма класів для серверної логіки

3.4 Архітектура бази даних

Архітектура бази даних застосунку для букінгу готелів побудована на основі реляційної моделі з використанням системи управління базами даних PostgreSQL. Такий підхід дозволяє ефективно організувати, зберігати та обробляти великі обсяги структурованих даних, необхідних для роботи з користувачами, готелями, номерами, бронюваннями та взаємодією між ними.

База даних реалізована через набір таблиць, що відображають основні сутності предметної області:

– users — зберігає інформацію про зареєстрованих користувачів (email, ім’я, прізвище, логін, пароль, телефон), а також пов’язується з ролями користувача через таблицю user_roles, дозволяючи визначати тип доступу (звичайний користувач або адміністратор).

– hotels — таблиця містить дані про готелі: назва, місто, країна, контактна інформація, статус доступності. Також є зв’язок з користувачем-власником (owner_id), що дає змогу реалізувати розмежування прав.

– rooms — кожен готель може мати декілька номерів. Таблиця зберігає інформацію про тип номера, його номер, ціну, доступність та прив’язку до готелю (hotel_id).

– bookings — фіксує всі бронювання користувачів, включаючи дати заїзду та виїзду, статус, унікальні токени, а також реалізує важливу функцію передачі бронювання через поля transferred_from_id та transferred_to_id, що посилаються на інші записи в цій же таблиці. Це дозволяє відстежувати повний ланцюжок передачі.

– review — таблиця для зберігання відгуків користувачів про готелі з оцінками (rating) і коментарями (comment), що дозволяє формувати рейтинг готелів.

– role_request — реалізує механізм запитів на зміну ролі користувача (наприклад, перехід з гостя на власника готелю). Включає статуси затвердження або відхилення, дату запиту та коментарі адміністратора.

– hotel_photo_urls — зв’язана з готелями і містить URL-адреси фотографій готелю.

– user_roles — таблиця, що задає ролі користувачів (наприклад, “admin”, “guest”, “owner”), пов’язана з users за user_id.

Всі зв’язки між таблицями побудовані за реляційною схемою:

– Один до багатьох (1:N) — один користувач може мати багато бронювань; один готель — багато номерів; один номер — багато бронювань.

– Багато до одного (N:1) — кожне бронювання, номер або відгук пов’язані з відповідним готелем чи користувачем.

– Рекурсивний зв’язок — у таблиці booking реалізовано через transferred_from_id та transferred_to_id, які дозволяють відслідковувати передачі між користувачами.

На рисунку 3.4 зображено UML-діаграму класів яка чітко демонструє структуру таблиць, їх атрибути та зв’язки між ними. Зокрема, видно основні сутності: users, hotels, rooms, bookings, а також допоміжні — review, user_roles, role_request. Діаграма дозволяє наочно оцінити, як реалізується логіка взаємодії між користувачами, готелями та бронюваннями в системі.

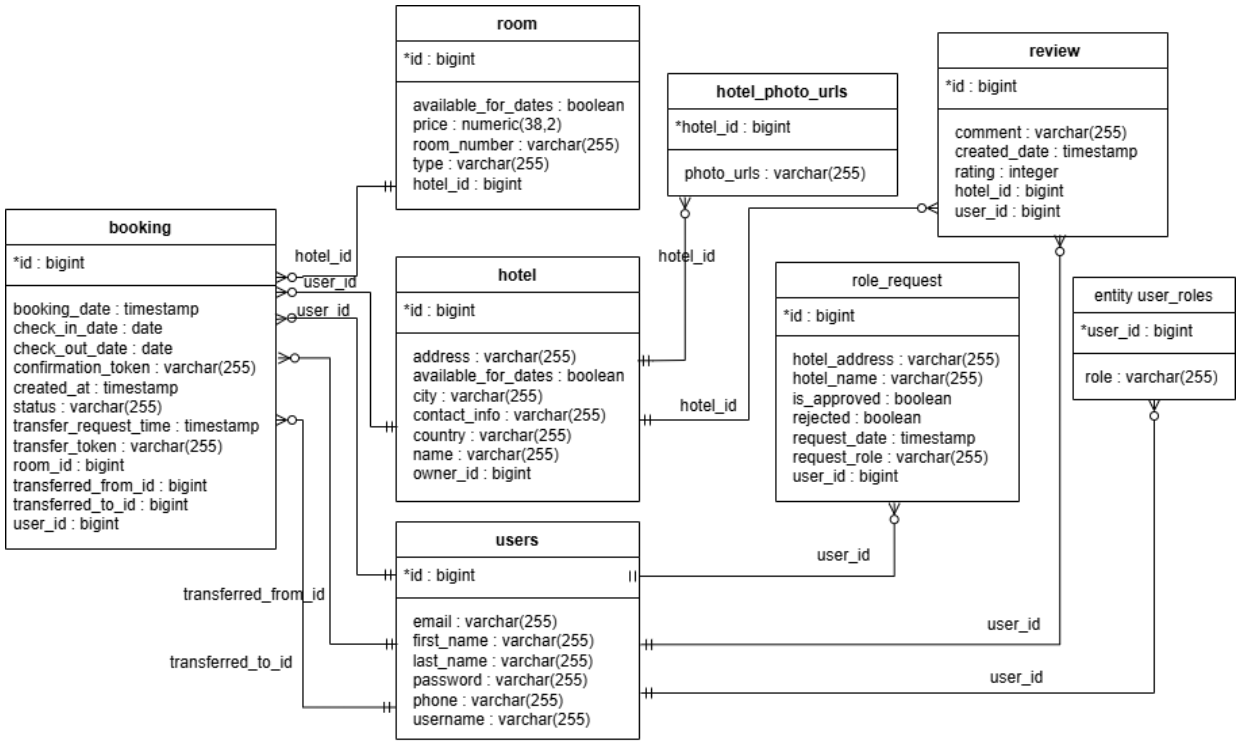


Рис. 3.4 Діаграму класів бази даних

4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

4.1 Дизайн інтерфейсу

У рамках розробки застосунку для онлайн-бронювання готелів дизайн інтерфейсу користувача реалізовувався без використання спеціалізованих інструментів для UI-прототипування, таких як Figma. Натомість дизайн створювався безпосередньо у середовищі розробки з використанням HTML, CSS та JavaScript — підхід, який дозволив одразу поєднувати дизайн із функціональністю.

Такий підхід мав низку переваг:

– Висока швидкість створення робочих макетів, які можна одразу перевірити у браузері.

– Можливість швидкого переходу від ескізу до інтерактивного інтерфейсу.

– Зменшення витрат часу на дублювання роботи (спочатку у дизайнерському інструменті, а потім у коді).

Основні етапи реалізації інтерфейсу:

Створення структури сторінок

HTML використовувався для побудови логічної структури сторінок. Були створені окремі шаблони для ключових розділів застосунку:

– Головна сторінка з полем пошуку готелів.
– Сторінка результатів пошуку з фільтрами, картками готелів та пагінацією.
– Сторінка з деталями готелю та кнопкою для переходу до бронювання.
– Особистий кабінет користувача з історією замовлень та обраними готелями.

Візуальне оформлення

CSS (зокрема Flexbox та Grid) забезпечив адаптивну верстку і сучасний вигляд інтерфейсу. Було реалізовано:

– Стандартну світлу тему з нейтральними кольорами, які не відволікають користувача від основного контенту.

– Використання шрифтів із відкритим ліцензуванням (наприклад, Google Fonts).

– Просту систему класів для повторного використання стилів (наприклад, кнопки, заголовки, картки готелів).

Додавання динаміки

JavaScript (у поєднанні з Angular) використовувався для реалізації інтерактивності:

Асинхронний пошук готелів із автозаповненням та запитами до Amadeus API.

Відображення списку результатів з можливістю фільтрації за ціною, рейтингом, розташуванням.

Обробка кліків, переходів, додавання готелів до обраного та взаємодія з сервером у режимі реального часу.

Адаптивність

Особлива увага приділялася адаптивному дизайну, який забезпечує коректне відображення інтерфейсу на пристроях з різною шириною екрану (мобільні телефони, планшети, десктопи).

Замість класичного макетування у Figma, дизайн інтерфейсу формувався у процесі написання коду. Основна увага приділялася не візуальній деталізації, а простоті користування, інтуїтивній навігації та швидкому доступу до основних функцій.

На рисунках 4.1 – 4.2 зображено інтерфейс розробленого застосунку.

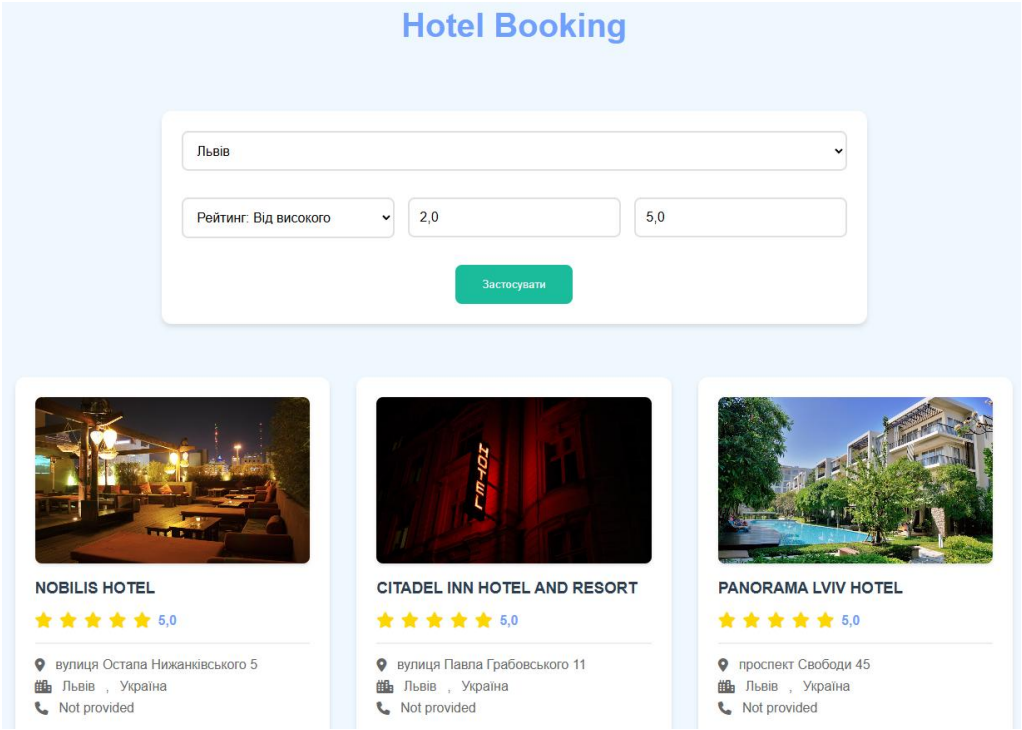


Рис. 4.1 Екранна форма реалізованої сторінки пошуку готелів

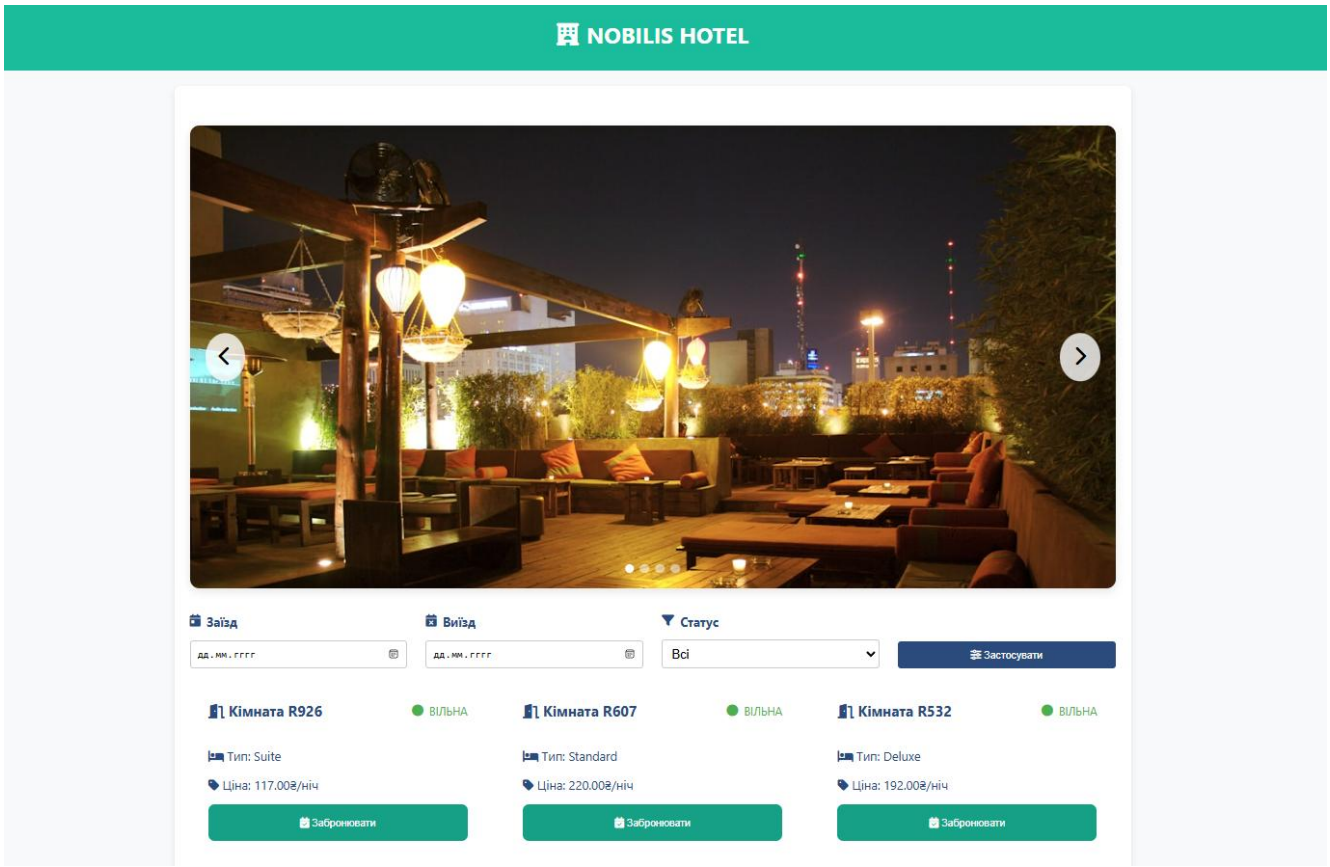


Рис. 4.2 Екранна форма сторінки перегляду деталей готелю та бронювання

4.2 Реалізація клієнтської частини

Клієнтська частина web-застосунку для бронювання готелів була реалізована з використанням стандартного веб-стека: HTML, CSS та JavaScript. Основний акцент робився на простоті, функціональності та взаємодії з серверною частиною, що працює на основі Java Spring. Розгортання застосунку не виконувалося — робота велася у локальному середовищі розробки.

4.2.1 Структура клієнтської частини

Клієнтська частина організована у вигляді HTML-файлів для кожної основної сторінки:

- Головна сторінка з пошуковим полем та коротким описом можливостей;
- Сторінка результатів пошуку, де відображаються знайдені готелі;
- Сторінка з деталями готелю та можливістю ініціювати бронювання;
- Особистий кабінет користувача, де можна переглянути історію бронювань.

Кожна HTML-сторінка супроводжується окремим CSS-файлом для стилізації та JavaScript-файлом для обробки подій, динаміки та взаємодії з API.

Використання CSS та JavaScript:

- Інтерфейс був стилізований за допомогою адаптивного CSS з використанням медіазапитів, Flexbox та Grid.

Це забезпечило коректне відображення інтерфейсу на різних пристроях, зокрема мобільних.

JavaScript використовувався для:

- обробки подій користувача (пошук, кнопки, введення);
- динамічного оновлення DOM (наприклад, відображення списку результатів пошуку без перезавантаження сторінки);
- асинхронного обміну даними з сервером за допомогою fetch-запитів до бекенду.

Взаємодія з сервером

– Зв’язок між клієнтом та серверною частиною, реалізованою на Java Spring, забезпечується через REST API. Клієнтська частина відправляє запити, зокрема:

– на пошук готелів через API Amadeus (виконувалося на сервері, а клієнт лише ініціював запит);

– на отримання деталей обраного готелю;

– на здійснення бронювання;

– на авторизацію та реєстрацію користувача.

Усі дані з сервера надходили у форматі JSON і оброблялися JavaScript на стороні клієнта.

Навігація між сторінками

Оскільки застосунок не є односторінковим (SPA), навігація реалізована класичним способом — переходом між окремими HTML-сторінками за допомогою тегів <a>. При цьому для покращення UX деякі елементи (наприклад, форми) надсилалися асинхронно, без повного оновлення сторінки.

Інтеграція з Amadeus API

Оскільки дані про готелі отримуються із зовнішнього джерела — API Amadeus, клієнтська частина ініціює запити через бекенд. Такий підхід дозволив:

– приховати ключі доступу до API;

– обробляти отриману інформацію на сервері;

– повертати клієнту лише релевантні дані.

Тестування та налагодження

Застосунок тестувався у браузері у процесі розробки зображено на рисунках 4.3 – 4.4.Налагодження клієнтської частини проводилося через інструменти розробника браузера (Chrome DevTools) , що дозволяло оперативно перевіряти коректність стилів, структури DOM та виконання JavaScript.

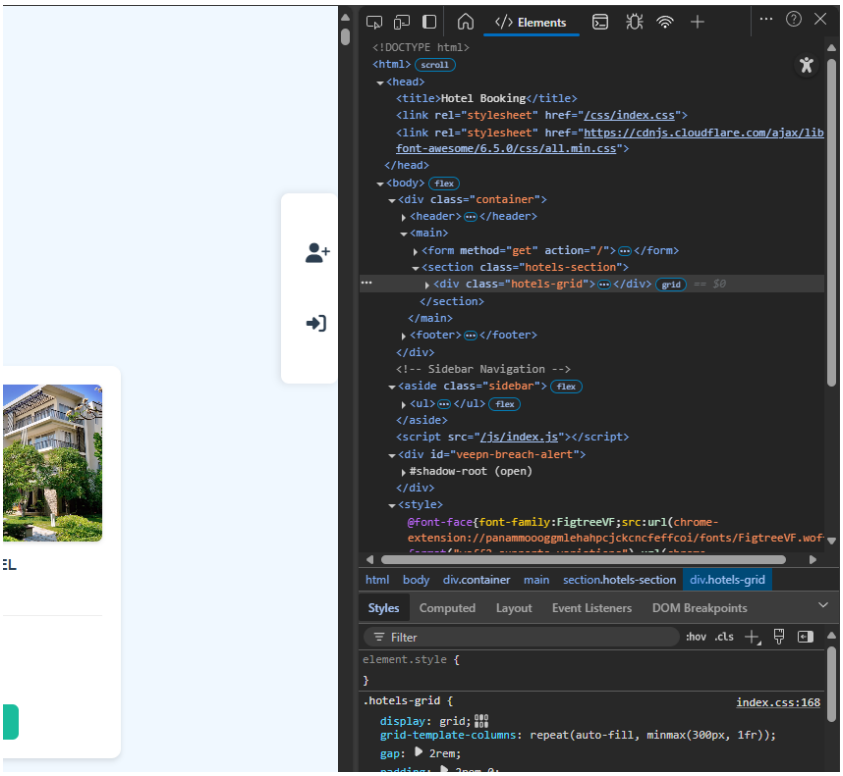


Рис. 4.3 Екранна форма реалізації сторінки пошуку готелів

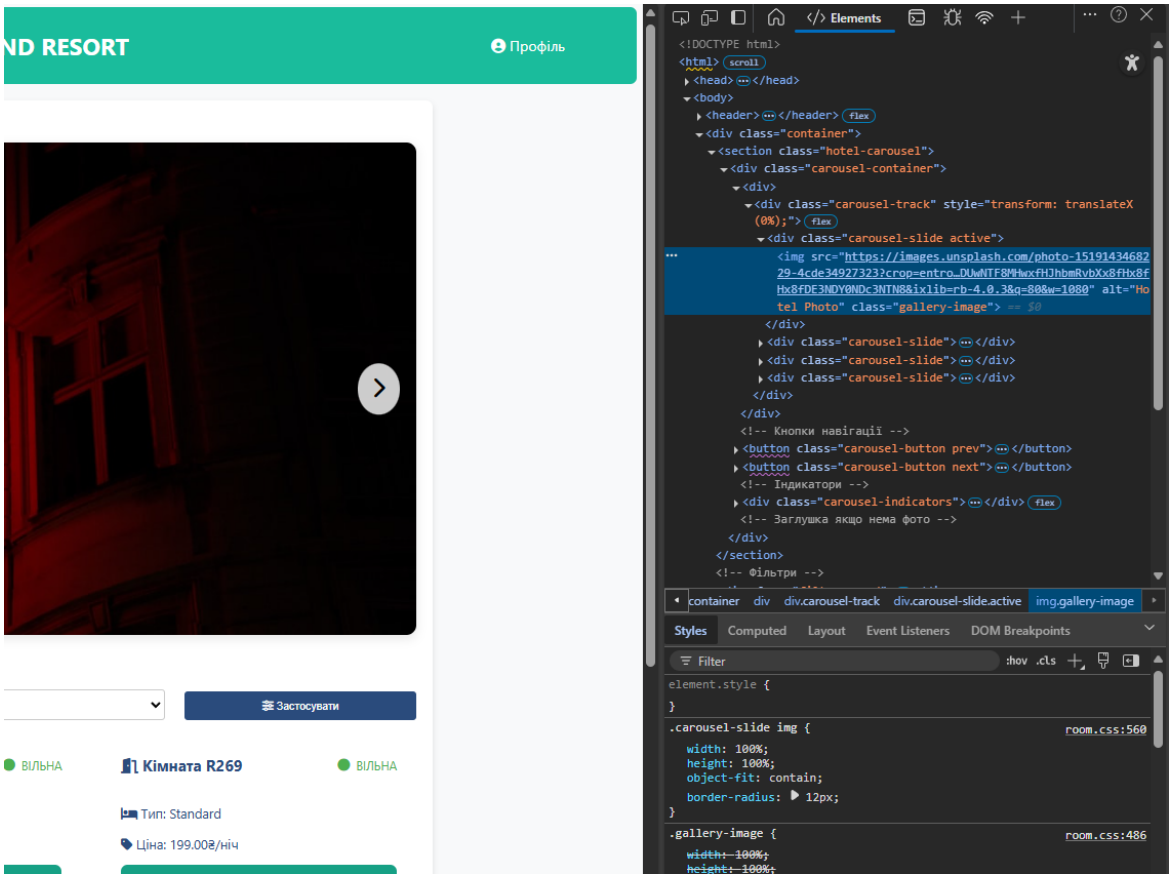


Рис. 4.4 Екранна форма реалізації сторінки перегляду деталей готелю

4.3 Реалізація серверної логіки передачі бронювання

Однією з унікальних функцій розробленого web-застосунку є можливість передачі активного бронювання іншому користувачу. Така функціональність рідко реалізується в існуючих комерційних сервісах (наприклад, Booking чи Airbnb), але є актуальною у випадках зміни планів користувача або бажання передати право користування готельним номером іншій особі.

Загальна логіка

Функція передачі реалізована на рівні серверної частини за допомогою Java + Spring Boot. Основні етапи:

– Користувач ініціює передачу бронювання, передаючи email отримувача;

– Сервер створює новий запис бронювання, копіюючи дані з оригінального, та створює спеціальний токен для передачі;

– У полі transferred_from_id нового запису вказується ідентифікатор попереднього бронювання, а у старому — transferred_to_id;

– Статус старого бронювання оновлюється на "передано", а нового — на "активне".

Структура коду

Контролер (BookingController) на рисунку 4.5 приймає запит на передачу:

```
@PostMapping("/{bookingId}/transfer")
public String transferBooking(
        @PathVariable Long bookingId,
        @RequestParam("newUserId") Long newUserId,
        RedirectAttributes redirectAttributes
) {
    try {
        bookingService.transferBooking(bookingId, newUserId);
        redirectAttributes.addFlashAttribute( attributeName: "success",  attributeValue: "Бронювання успішно передано");
    } catch (RuntimeException ex) {
        redirectAttributes.addFlashAttribute( attributeName: "error", ex.getMessage());
    }
    return "redirect:/profile";
}
```

Рис. 4.5 Фрагмент коду для реалізації методу transferBooking()

Сервіс (BookingService) на рисунку 4.6 реалізує бізнес-логіку:

```
@Transactional  1 usage
public void transferBooking(Long bookingId, Long newUserId) {
    Booking booking = bookingRepository.findById(bookingId)
            .orElseThrow(() -> new RuntimeException("Бронювання не знайдено"));

    User currentUser = booking.getUser();
    User newUser = userService.getUserById(newUserId);

    booking.setTransferredFrom(currentUser);
    booking.setTransferredTo(newUser);

    currentUser.getBookings().remove(booking);
    newUser.getBookings().add(booking);
    booking.setUser(newUser);

    bookingRepository.save(booking);
    userService.saveUser(currentUser);
    userService.saveUser(newUser);
}
```

Рис 4.6 Фрагмент коду для реалізації методу transferBooking()

Сутність (Booking.java) на рисунку 4.7 включає поля:

```
@ManyToOne  3 usages
@JoinColumn(name = "transferred_from_id")
private User transferredFrom; // Хто передав бронювання

@ManyToOne  3 usages
@JoinColumn(name = "transferred_to_id")
private User transferredTo; // До кого передається
```

Рис 4.7 Фрагмент коду для передачі бронювання

Вони дозволяють побудувати ланцюжок передач бронювання, якщо таких кілька.

Завдяки використанню Spring Data JPA, усі зміни автоматично відображаються в таблиці booking. Кожен запис пов’язаний з user_id, room_id, а також містить посилання на первинне або наступне бронювання в ланцюжку передачі.

4.4 Синхронізація готелів через Amadeus API

Для забезпечення актуальності та повноти даних у застосунку було реалізовано синхронізацію з API Amadeus — одним із найпопулярніших провайдерів туристичних і готельних сервісів. Синхронізація дозволяє отримувати реальні дані про готелі, їх розташування, контактну інформацію, відгуки, доступні номери тощо.

Функція імпорту даних реалізована на стороні серверної частини у вигляді окремого сервісу.Основні завдання синхронізації:

– Надіслати запит до Amadeus API за містом (код міста);

– Отримати список готелів (масив об’єктів Hotel[]);

– Витягнути з кожного готелю необхідну інформацію: назву, фото, кімнати, контакти, геокоординати;

– За координатами — отримати фактичне місто, країну та адресу через окрему службу геокодування;

– Перетворити об'єкт стороннього API на власний клас Hotel, який зберігається в базі даних;

– Зберегти об'єкти у відповідні таблиці через JPA-репозиторії.

Ключові методи синхронізації:

– syncHotelsFromAmadeus(String cityCode) — основна точка входу. Вона надсилає запит до Amadeus, отримує відповідь та зберігає готелі у локальну базу;

– getHotels() — метод-запит до Amadeus API;

– getGeoCode() — отримання координат (широта/довгота), які далі використовуються для визначення міста і країни;

– getCityCountry(lat, lon) — допоміжна служба геолокації, яка повертає інформацію для заповнення поля address.

Усі отримані об'єкти Amadeus перетворюються у внутрішні сутності через методи setName(), setAddress(), setCity(), setCountry(), setRooms() тощо. Таким

чином, сторонній формат адаптується до структури локального застосунку, що дозволяє уніфіковано обробляти дані з різних джерел.

На рисунку 4.8 зображена UML-діаграма яка демонструє повний цикл викликів між класами Service, Amadeus і ApiHotel. Видно, що запит запускається методом syncHotelsFromAmadeus(cityCode), після чого послідовно витягуються всі дані про готель, включаючи координати. Далі ці координати перетворюються на адресу, і створюється новий локальний об'єкт Hotel.

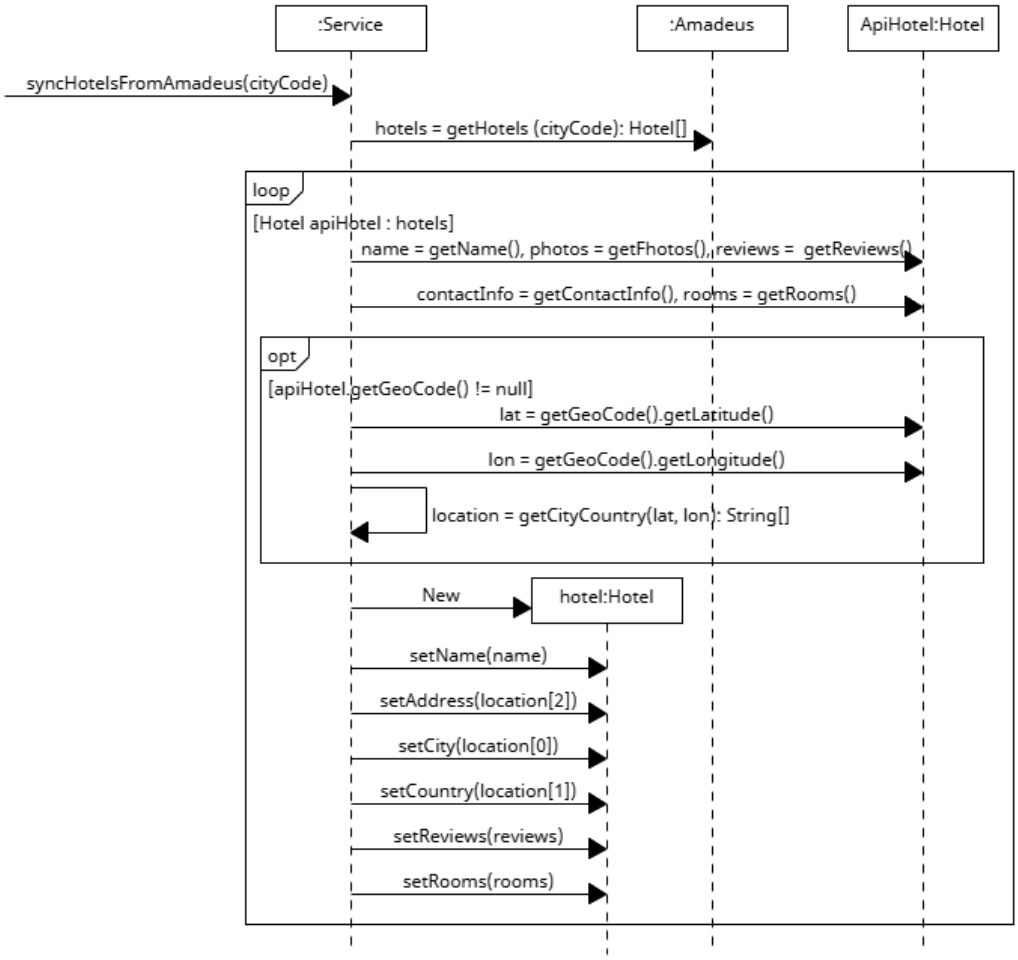


Рис. 4.8 Діаграма послідовності синхронізації даних з Amadeus API

Отримані об’єкти Hotel, Room, Review зберігаються в базі PostgreSQL через репозиторії Spring Data JPA. Це дозволяє уникати дублювання записів і гарантує актуальність інформації у web-застосунку. Синхронізацію можна запускати періодично або вручну для конкретного міста.

Для імпорту актуальної інформації про готелі у застосунку реалізовано спеціальний сервіс AmadeusHotelService, який відповідає за взаємодію з API Amadeus. Сервіс забезпечує отримання та трансформацію даних про готелі у внутрішні сутності системи, з подальшим збереженням до локальної бази даних.

Основна логіка реалізована в методі syncHotelsFromAmadeus(String cityCode), який ініціює запит до Amadeus API на основі коду міста. У відповіді повертається масив об'єктів, кожен з яких містить ключову інформацію про готель. На основі отриманих координат викликається окрема служба геокодування, яка дозволяє визначити точну адресу, місто та країну розташування готелю. Потім зібрані дані формують об'єкт внутрішньої сутності Hotel, який містить також інформацію про фото, контакти, доступні кімнати та відгуки користувачів.

Зібрані дані адаптуються до структури програми за допомогою сеттерів зображено на рисунку 4.9 (setName(), setAddress(), setCity() тощо), після чого зберігаються в базу даних через HotelRepository.

```
public void syncHotelsFromAmadeus(String cityCode) throws Exception {  1 usage
    Amadeus amadeus = Amadeus.builder( clientId: "oKxXTG1SA5Wi6LkKmt6cciDHPv65AdEA",  clientSecret: "gsS0NPnEewVbR2oG").build();

    User systemUser = userRepository.findByUsername("system-bot").orElseThrow(()->new Exception("System user not found"));

    Hotel[] hotels = amadeus.referenceData.locations.hotels.byCity.get(
            Params.with("cityCode", cityCode)
                    .and("radius", 50)
                    .and("radiusUnit", "KM")
                    .and("hotelSource", "ALL")
    );
```

Рис. 4.9 Фрагмент коду синхронізації даних з Amadeus API

Крім основного методу синхронізації, додаткові допоміжні методи відповідають за:

– взаємодію з API геокодування для визначення точної адрес зображено на риснку 4.10 (getCityCountryAndAddressFromCoordinates);

```
private String[] getCityCountryAndAddressFromCoordinates(double lat, double lon) {  1 usage
    try {
        String url = "https://nominatim.openstreetmap.org/reverse?format=json&lat=" + lat + "&lon=" + lon;
        HttpRequest request = HttpRequest.newBuilder()
                .uri(URI.create(url))
                .header( name: "User-Agent",  value: "hotel-booking-app")
                .build();

        HttpResponse<String> response = HttpClient.newHttpClient().send(request, HttpResponse.BodyHandlers.ofString());
        String body = response.body();

        String city = extractJsonValue(body,  prefix: "\"city\":\"",  suffix: "\"");
        if (city == null) city = extractJsonValue(body,  prefix: "\"town\":\"",  suffix: "\"");
        if (city == null) city = extractJsonValue(body,  prefix: "\"village\":\"",  suffix: "\"");
        if (city == null) city = "Unknown City";

        String country = extractJsonValue(body,  prefix: "\"country\":\"",  suffix: "\"");
        if (country == null) country = "Unknown Country";

        String road = extractJsonValue(body,  prefix: "\"road\":\"",  suffix: "\"");
        String houseNumber = extractJsonValue(body,  prefix: "\"house_number\":\"",  suffix: "\"");

        String address = "";
        if (road != null) address += road;
        if (houseNumber != null) address += " " + houseNumber;
        if (address.isBlank()) address = "Unknown Address";

        return new String[]{city, country, address};
```

Рис. 4.10 Фрагмент коду синхронізації вулиць з Nominatim Reverse Geocoding API

– генерацію посилань на фото для готелів зображено на риснку 4.11 (через Unsplash API або резервні варіанти);

```
private List<String> generateHotelPhotos(int count) {  1 usage
    List<String> photos = new ArrayList<>();
    String accessKey = "hygmPhoerF9qfIg5gpYrsjdG6pMnIOsNt1zzATI4-CM";
    try {
        String apiUrl = "https://api.unsplash.com/photos/random?query=hotel&count=" + count + "&client_id=" + accessKey;
        HttpRequest request = HttpRequest.newBuilder()
                .uri(URI.create(apiUrl))
                .build();

        HttpResponse<String> response = HttpClient.newHttpClient().send(request, HttpResponse.BodyHandlers.ofString());
        JsonArray jsonArray = JsonParser.parseString(response.body()).getAsJsonArray();

        for (JsonElement element : jsonArray) {
            JsonObject photo = element.getAsJsonObject();
            String url = photo.getAsJsonObject( memberName: "urls").get("regular").getAsString();
            photos.add(url);
        }
```

Рис. 4.11 Фрагмент коду синхронізації зображень з Unsplash API

– адаптацію отриманих об'єктів до внутрішньої структури застосунку зображено на рисунку 4.11.

```
org.booking.hotelbooking.Entity.Hotel hotel = new org.booking.hotelbooking.Entity.Hotel();
hotel.setName(name);
hotel.setAddress(address);
hotel.setCity(city);
hotel.setCountry(country);
hotel.setContactInfo("Not provided");

List<String> photos = generateHotelPhotos( count: 3 + random.nextInt( bound: 3)); // 3-5 фото
hotel.setPhotoUrls(photos);
```

Рис. 4.11 Фрагмент коду адаптації отриманих об'єктів

Ця реалізація забезпечує автоматичне оновлення актуальної інформації у системі, зберігаючи при цьому єдину архітектуру та гнучкість обробки даних із зовнішніх джерел.

4.5 Автоматизована обробка бронювань

Для підвищення надійності та зручності у користуванні застосунком, було реалізовано автоматизований процес перевірки актуальності бронювань. Щодня о опівночі система запускає скрипт, який перевіряє, чи не настав граничний термін підтвердження бронювання.

Логіка перевірки:

– Щоденно о 12:00 сервер ініціює перевірку бронювань зі статусом “очікує підтвердження” або “нове”;

– Для кожного бронювання перевіряється, чи поточна дата співпадає з останнім днем дії бронювання;

– Якщо гість не підтвердив бронювання, воно автоматично скасовується, і система надсилає відповідні повідомлення готелю та гостю;

Цей процес реалізований за допомогою планувальника завдань (@Scheduled) у Spring Boot зображено на рисунку 4.12.

```
@Scheduled(cron = "0 0 0 * * ?") // Щодня о півночі at 00:00
public void markCompletedBookings() {
    List<Booking> bookings = bookingRepository.findByStatusIn(
            List.of(BookingStatus.CONFIRMED, BookingStatus.PENDING)
    );

    LocalDate today = LocalDate.now();
    bookings.forEach( Booking booking -> {
        if (booking.getCheckOutDate().isBefore(today)) {
            booking.setStatus(BookingStatus.COMPLETED);
            bookingRepository.save(booking);
        }
    });
}
}
```

Рис. 4.12 Фрагмент коду для реалізації автоматичної обробки бронювань

На рисунку 4.13 зображена UML-діаграма логіки перевірки: зчитування дати, перевірка умов, автоматичне скасування або підтвердження бронювання, та перехід до розрахунків.

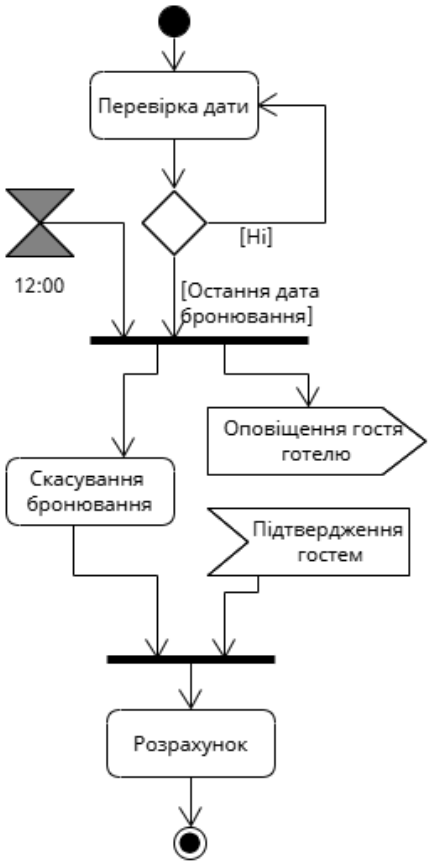


Рис. 4.13 UML-діаграма активності обробки прострочених бронювань

4.8 Реалізація функціональності передачі бронювання

Однією з унікальних функцій, реалізованих у межах web-застосунку для бронювання готелів, стала можливість передачі активного бронювання іншому користувачу. Така функція дозволяє користувачу, у разі неможливості скористатися послугою, передати бронювання кімнати іншій особі, що є особливо актуальним у випадках зміни планів або непередбачених обставин.

Реалізація цієї функціональності відбувається у кілька логічних етапів, кожен з яких реалізовано окремим методом у класі BookingService, що відповідає за всю бізнес-логіку, пов’язану з бронюваннями.

4.8.1 Ініціація запиту на передачу

Передача бронювання починається з виклику методу requestTransfer, у який передається ідентифікатор поточного бронювання та електронна адреса нового користувача. Система здійснює перевірку наявності як самого бронювання, так і користувача за вказаним email. Після цього генерується унікальний токен передачі, що зберігається у полі transferToken відповідного об'єкта Booking. Також фіксується час створення запиту (transferRequestTime) і тимчасово встановлюється майбутній власник у полі transferredTo.

Після збереження змін у базі даних, новому користувачу надсилається електронний лист із посиланням для підтвердження передачі. Це посилання містить токен і веде на відповідну кінцеву точку бекенду, яка обробляє підтвердження. Фрагмент коду зображено на рисунку 4.13.

```java
@Transactional
public void requestTransfer(Long bookingId, String newUserEmail) {
    Booking booking = bookingRepository.findById(bookingId)
            .orElseThrow(() -> new RuntimeException("Бронювання не знайдено"));

    User newUser = userService.getUserByEmail(newUserEmail);

    String transferToken = UUID.randomUUID().toString();
    booking.setTransferToken(transferToken);
    booking.setTransferRequestTime(LocalDateTime.now());
    booking.setTransferredTo(newUser);

    bookingRepository.save(booking);

    String confirmationLink = "http://localhost:8080/bookings/transfer/confirm?token=" + transferToken;
    emailService.sendTransferConfirmationEmail(newUser.getEmail(), confirmationLink);
}
```

Рис. 4.13 Фрагмент коду для реалізації запиту на передачу бронювання

4.8.2 Підтвердження передачі

Коли новий користувач переходить за отриманим посиланням, активується метод confirmTransfer, який перевіряє дійсність переданого токена. Окремо перевіряється, чи не минуло понад 24 години з моменту створення запиту на передачу — після цього терміну запит вважається недійсним.

Якщо всі перевірки проходять успішно, система змінює власника бронювання, оновлюючи поля:

– user — новий активний власник;

– transferredFrom — попередній власник;

– transferredTo — новий користувач (очікуване значення, яке після підтвердження стає актуальним);

– transferToken — обнуляється для недопущення повторного використання.

Також відповідно оновлюються списки бронювань у обох користувачів, після чого всі зміни зберігаються в базі даних.Фрагмент коду зображено на рисунку 4.14.

```
@Transactional  1 usage
public void confirmTransfer(String token) {
    Booking booking = bookingRepository.findByTransferToken(token)
            .orElseThrow(() -> new RuntimeException("Невірний токен"));

    long hoursPassed = ChronoUnit.HOURS.between(booking.getTransferRequestTime(), LocalDateTime.now());
    if (hoursPassed > 24) {
        throw new RuntimeException("Час на підтвердження передачі минув");
    }

    User oldUser = booking.getUser();
    User newUser = booking.getTransferredTo();

    oldUser.getBookings().remove(booking);
    newUser.getBookings().add(booking);
    booking.setUser(newUser);
    booking.setTransferredFrom(oldUser);
    booking.setTransferToken(null);

    userService.saveUser(oldUser);
    userService.saveUser(newUser);
    bookingRepository.save(booking);
}
```

Рис. 4.14 Фрагмент коду для реалізації підтвердження передачі

4.8.3 Загальний огляд реалізованого механізму передачі

Функціональність передачі бронювання була повністю інтегрована в загальну архітектуру web-застосунку як з технічної, так і з користувацької точки зору. Цей механізм реалізовано у вигляді окремого сервісного шару (BookingService), що дотримується принципу розділення відповідальностей: логіка передачі не дублюється і не конфліктує з іншими функціями, такими як створення, підтвердження чи скасування бронювання.

З боку користувача передача бронювання можлива з особистого кабінету, де передбачено інтерфейс для введення email нового власника. Після підтвердження запиту користувач отримує повідомлення про очікування підтвердження новим власником, що створює прозору логіку взаємодії. У свою чергу, новий користувач отримує email з унікальним посиланням, яке веде на сторінку підтвердження, після чого відбувається оновлення статусу бронювання.

На рівні бекенду система забезпечує цілісність даних шляхом транзакційної обробки всіх змін у базі даних. Тобто, кожна операція — створення запиту на

передачу, підтвердження або її скасування — виконується як атомарна транзакція, що унеможливлює появу «висячих» або неповністю переданих бронювань.

Крім цього, впроваджено механізм автоматичного контролю строку дії запиту. Якщо новий користувач не підтвердив передачу бронювання протягом 24 годин, токен стає недійсним, а бронювання залишається у попереднього власника. Це запобігає помилковим або випадковим передачам та гарантує, що лише своєчасно підтверджені бронювання впливають на систему.

Таким чином, передача бронювання реалізована як повноцінна частина життєвого циклу бронювання, що працює у тісній взаємодії з іншими підсистемами — зокрема, з обліком користувачів, поштою, перевіркою доступності номерів та обробкою статусів. Це значно підвищує гнучкість і надає користувачам більше контролю над власними замовленнями.

4.6 Завантаження проєкту на GitHub

У процесі розробки web-застосунку для бронювання готелів було використано систему контролю версій Git. Основна мета — збереження проміжних версій коду, можливість відновлення попереднього стану у разі помилок, а також централізоване зберігання проєкту на GitHub.

Замість класичної ініціалізації Git у кінці проєкту, репозиторій використовувався з самого початку — для збереження ключових етапів розробки, таких як:

- реалізація REST API на сервері за допомогою Java Spring;
- створення HTML-сторінок та стилів;
- реалізація взаємодії з API Amadeus;
- покрокове оновлення логіки взаємодії між клієнтом і сервером.

Зміни завантажувалися до GitHub із використанням таких базових команд:

- git add . — додавання змінених файлів;
- git commit -m "Оновлено пошук готелів через Amadeus API" — створення коміту;

– git push — завантаження змін у віддалений репозиторій.

Система Git використовувалася переважно для збереження стабільних версій проєкту після кожного етапу перевірки. Окремих гілок не створювалося, оскільки розробка здійснювалася індивідуально, у межах кваліфікаційного проєкту.Завантажений проект зображено на рисунку 4.15.

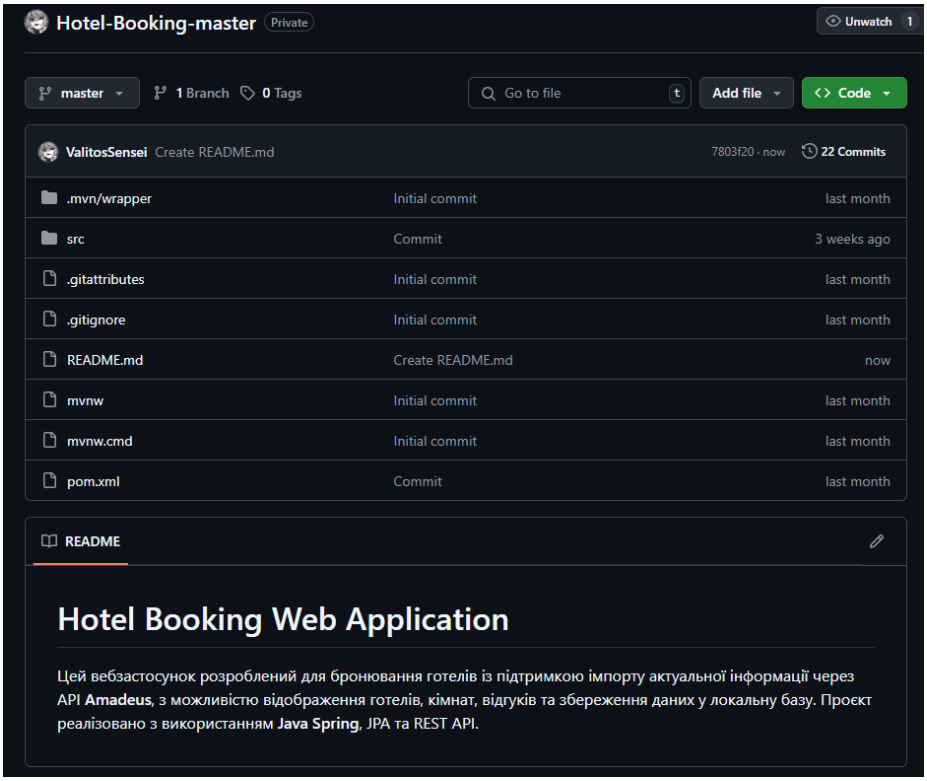


Рис. 4.15 Приклад завантаженого проекту на GitHub

4.7 Тестування застосунку

У рамках реалізації та перевірки працездатності web-застосунку не проводилось автоматизоване модульне тестування у вигляді JUnit чи аналогічних інструментів. Натомість застосовувався практичний підхід: тестування функціональності вручну за допомогою інструментів Postman і вебпереглядача.

Тестування охоплювало такі етапи:

– Тестування REST API (через Postman):

– Надсилання GET-запитів для пошуку готелів за допомогою інтеграції з Amadeus API;

– Перевірка параметрів запитів (місто, дати, кількість осіб);

– Тестування відповіді на некоректні запити (наприклад, відсутність результатів, неправильний формат дати);

– Тестування POST-запитів при бронюванні готелю та взаємодії з базою даних.

Тестування клієнтської частини (через браузер):

– Перевірка завантаження сторінок та коректності верстки;

– Візуальна перевірка елементів інтерфейсу: полів введення, кнопок, навігації;

– Обробка некоректного введення користувачем (наприклад, порожні поля у формі);

– Відображення отриманих з бекенду результатів у HTML-таблицях.

Журнали логування:

– Для відстеження помилок використовувалося базове логування у Spring Boot;

– Повідомлення у консоль дозволяли оперативно виявляти проблеми на сервері (наприклад, помилки при зверненні до Amadeus API або помилки авторизації).

ВИСНОВОК

Результатом виконаної роботи стала розробка web-застосунку для готельного букінгу з можливістю повторного бронювання (ребукінгу) та передачі бронювання іншому користувачу, що дозволяє підвищити гнучкість взаємодії між мандрівниками та готелями. Реалізація даного програмного забезпечення базується на сучасних інструментах і фреймворках, які забезпечують масштабованість, безпеку, надійність і розширюваність системи.

Було виконано наступні задачі:

– Проведено аналіз предметної галузі: охарактеризовано сферу онлайн бронювання, виявлено ключові проблеми існуючих рішень, такі як відсутність передачі бронювання, складність при повторному створенні подібного бронювання та обмеженість у налаштуванні. На основі цього було визначено цільову аудиторію (мандрівники, туристи, адміністратори готелів) і сформульовано функціональні та нефункціональні вимоги до майбутньої системи.

– Виконано аналіз існуючого програмного забезпечення: розглянуто сервіси Booking.com, Airbnb та Trivago, визначено їх недоліки, що стали підґрунтям для обґрунтування необхідності створення нового рішення.

– Обрано засоби реалізації: Java як основна мова програмування, Spring Boot як фреймворк для побудови backend-частини, PostgreSQL як СУБД для збереження даних, Amadeus API як зовнішнє джерело інформації про готелі та бронювання, GitHub як система контролю версій, IntelliJ IDEA як середовище розробки, Postman для тестування REST API та Figma для створення прототипу інтерфейсу користувача.

– Створено архітектурну модель системи: побудовано UML-діаграми (класів, розгортання, прецедентів), які описують структуру та логіку роботи застосунку.

– Реалізовано прототип web-застосунку: налаштовано структуру проєкту, створено модулі реєстрації користувачів та готелів, пошуку, бронювання, передачі бронювання, а також систему фільтрації, рейтингу та сповіщень. Реалізовано

кешування результатів пошуку, щоденне резервне копіювання даних, захист форм від CSRF-атак і хешування паролів за допомогою BCrypt.

Таким чином, розроблений web-застосунок не лише виконує функції стандартного онлайн-бронювання, але й вирішує реальні проблеми, з якими стикаються користувачі у щоденній практиці. Отримані результати будуть використані у кваліфікаційній бакалаврській роботі.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Філаретов М. Ю., Захист інформації у web-застосунку букінг готелів // Матеріали XII Всеукраїнської науково-практичної конференції молодих учених «Інформаційні технології – 2025», 15.05.25, КСУ імені Бориса Грінченка, Київ, Україна, С. 336-337.
2. Філаретов М. Ю., Гаманюк І. М. Вкористання діаграми послідовності для моделювання процесів веб застосунку для бронювання готелів // Матеріали VI Всеукраїнську науково-технічну конференцію «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 24.04.25,ДУІКТ, Київ, Україна, С. 136-138.

ПЕРЕЛІК ПОСИЛАНЬ

1. Walls C. Spring in Action, 6th Edition. Manning Publications, 2022. 520 p. URL:https://www.manning.com/books/spring-in-action-sixth-edition (date of access: 23.05.2025).

2. PostgreSQL Global Development Group. PostgreSQL 17.5 Documentation. URL: https://www.postgresql.org/docs/17 (date of access: 23.05.2025).

3. Amadeus for Developers. Self-Service Travel API Catalog. URL: https://developers.amadeus.com/self-service (date of access: 23.05.2025).

4. OWASP Foundation. OWASP Top Ten Web Application Security Risks. URL: https://owasp.org/www-project-top-ten/ (date of access: 23.05.2025).

5. Pautasso C., Zimmermann O., Leymann F. RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision. Proceedings of the 17th International Conference on World Wide Web, 2008. P. 805–814. URL: https://www.researchgate.net/publication/221022734_RESTful_Web_Services_vs_Big_Web_Services_-_Making_the_Right_Architectural_Decisions (date of access: 23.05.2025).

6. Sill A. Cloud Security: A Comprehensive Guide to Secure Cloud Computing. Wiley, 2010. 384 p. URL: https://www.wiley.com/en-us/Cloud%2BSecurity%3A%2BA%2BComprehensive%2BGuide%2Bto%2BSecure%2BCloud%2BComputing-p-x000530232 (date of access: 23.05.2025).

7. TechTarget. What is a Booking Engine? URL: https://www.techtarget.com/whatis/definition/booking-engine (date of access: 23.05.2025).

8. Cloudbeds. What is a Hotel Booking Engine? (+Top 9 Systems). URL:https://www.cloudbeds.com/articles/hotel-booking-engine-guide/ (date of access: 23.05.2025).

9. Hotel Tech Report. What is a Booking Engine? Complete Guide to Hotel Booking Systems. URL: https://hoteltechreport.com/news/hotel-booking-engine-guide (date of access: 23.05.2025).

10. Amazon.com. Usability Engineering: Nielsen, Jakob. URL: https://www.amazon.com/Usability-Engineering-Jakob-Nielsen/dp/0125184069 (date of access: 23.05.2025).

11. Amazon.com. Cloud Security: A Comprehensive Guide to Secure Cloud Computing. URL: https://www.amazon.com/Cloud-Security-Comprehensive-Secure-Computing/dp/0470589876 (date of access: 23.05.2025).

12. Шевченко О. В., Журавель В. П. Технології розробки веб-додатків: навч. посібник. – Київ: КНЕУ, 2021. – 234 с.

13. Oracle. Java Platform, Standard Edition Documentation. URL: https://docs.oracle.com/javase/8/docs/ (date of access: 25.05.2025).

14. JetBrains. IntelliJ IDEA Overview. URL: https://www.jetbrains.com/idea/features/ (date of access: 25.05.2025).

15. Mozilla Developer Network. JavaScript Guide. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide (date of access: 25.05.2025).

16. W3C. HTML & CSS Standards. URL: https://www.w3.org/standards/webdesign/htmlcss (date of access: 25.05.2025).

17. Spring.io. Spring Framework Documentation. URL: https://spring.io/projects/spring-framework (date of access: 25.05.2025).

18. Ярошенко С. В. Безпека веб-додатків: навч. посібник. – Харків: ХНУРЕ, 2020. – 148 с.

19. DigitalOcean. Git Version Control Essentials. URL: https://www.digitalocean.com/community/tutorials/git-essential-commands (date of access: 25.05.2025).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка web-застосунку для готельного букінгу з функціями ребукінгу та передачі бронювання

Виконав студент4 курсу
групи ПД-41
Філаретов Максим Юрійович
Керівник роботи
старший виклад кафедри ІПЗ Гаманюк Ігор Михайлович

Київ – 2025

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** – покращення процесів онлайн-бронювання готелів шляхом розробки web-застосунку з механізмами передачі бронювання між користувачами .
- **Об’єкт дослідження** – процес онлайн-бронювання готелів.
- **Предмет дослідження** – архітектура та алгоритми web-застосунку для готельного букінгу з підтримкою передачі бронювань.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Провести аналіз існуючих систем готельного букінгу та виявити їхні недоліки щодо гнучкості зміни броні.
2. Дослідити технологічні рішення для реалізації механізмів ребукінгу (зміна дат/номерів) та передачі бронювань між користувачами.
3. Сформувати вимоги до web-застосунку з урахуванням безпеки транзакцій та інтеграції з API Amadeus.
4. Розробити архітектуру клієнт-серверного додатку на базі Java Spring (бекенд) та сучасних веб-технологій (фронтенд).
5. Реалізувати функції: динамічний пошук готелів з фільтрами, передача бронювання.
6. Провести тестування продуктивності системи та її відповідність вимогам безпеки.

3

АНАЛІЗ АНАЛОГІВ

Показник	Booking.com	Airbnb	Trivago	Розроблений застосунок
Передача бронювання	-	-	-	+
Фільтри пошуку	+	+	-	+
Рейтинг готелів	+	-	+	+
Повідомлення на email	+	-	+	+
Автоматичне оновлення статусу бронювання	-	-	-	+
Сповіщення про зміни	+	+	-	+

4

ВИМОГИ ДО ЗАСТОСУНКУ

Функціональні вимоги:

1. Реєстрація користувачів/готелів з верифікацією email.
2. Пошук готелів за параметрами (місто, рейтинг).
3. Здійснення бронювання.
4. Сповіщення:автоматична відправка email:
5. Передача бронювання: автоматичне оновлення статусу бронювання в системі.
6. Фільтрація за статусом (активні, завершені, передані).
7. Опрацювання відгуків та рейтингів готелів.

Нефункціональні вимоги:

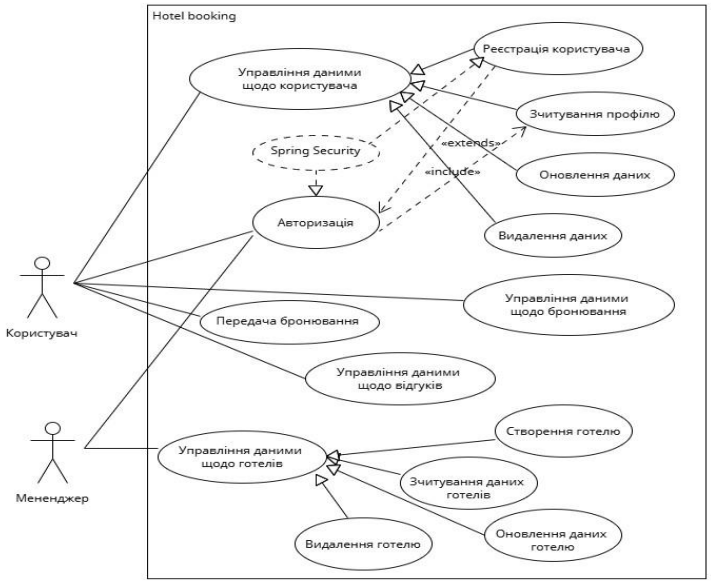
1. Захист від CSRF-атак (токени у формах).
2. Хешування паролів за допомогою BCrypt.
3. Кешування результатів пошуку.
4. Резервне копіювання даних щодоби.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

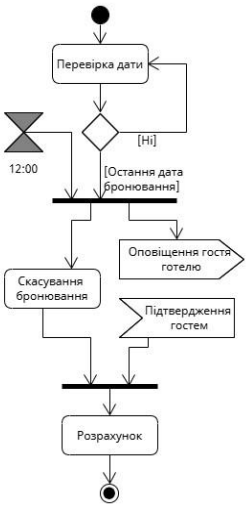
6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



7

ДІАГРАМИ ДІЯЛЬНОСТІ

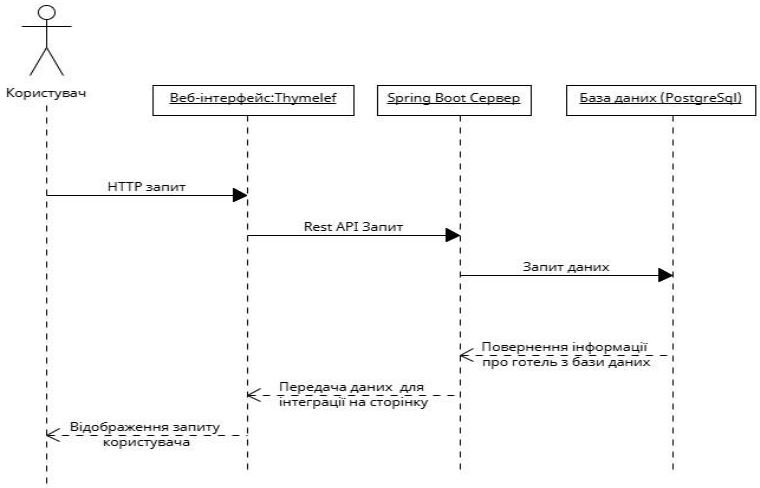


Завершення бронювання

Створення бронювання

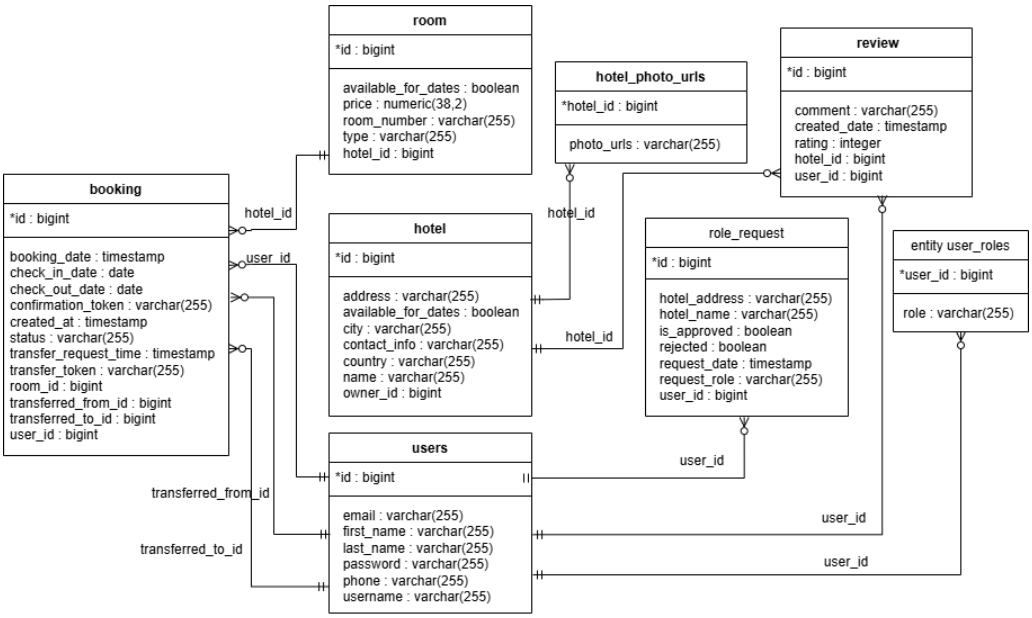
8

ДІАГРАМА ПОСЛІДОВНОСТІ



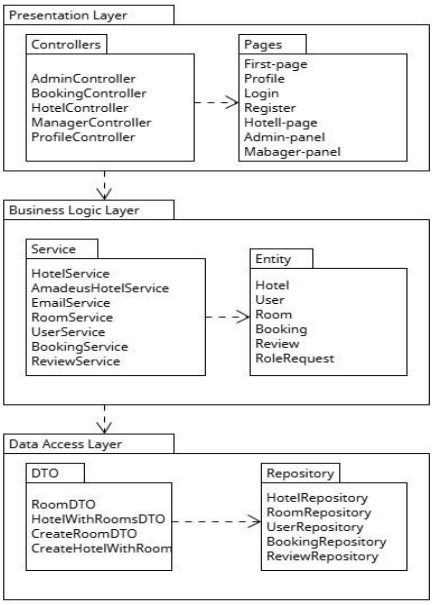
9

СТРУКТУРА БАЗИ ДАННИХ



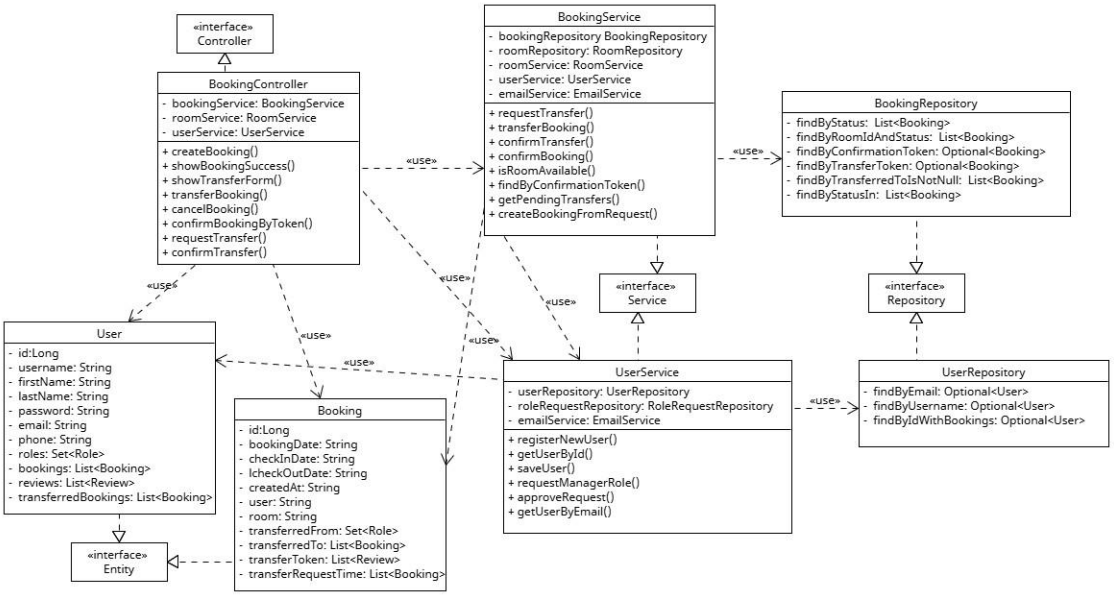
10

АРХІТЕКТУРА ДОДАТКУ



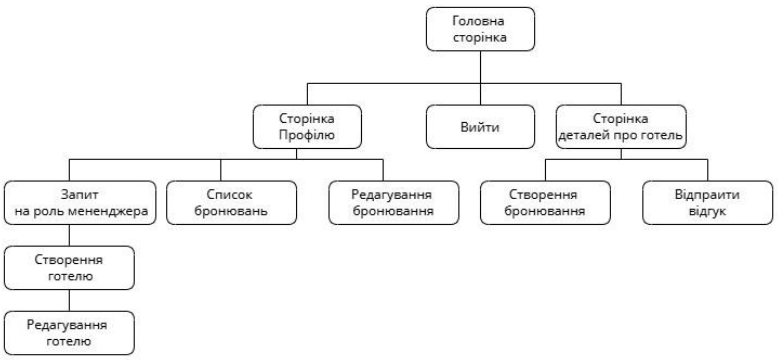
11

ДІАГРАМА КЛАСІВ СТВОРЕННЯ БРОНЮВАННЯ



12

МАПИ ВЕБ-ЗАСТОСУНКУ

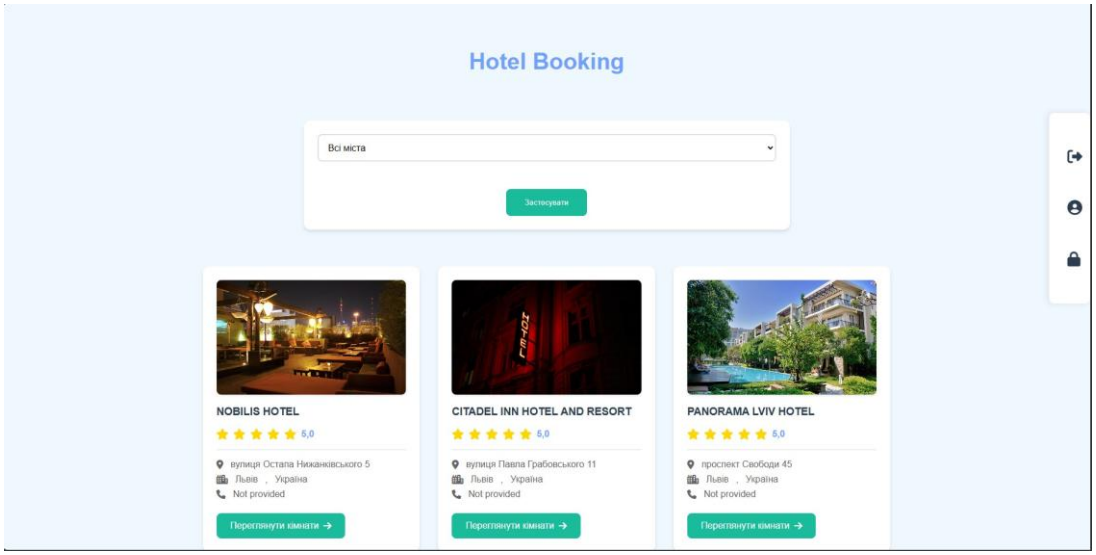


Авторизований користувач

Неавторизований користувач

13

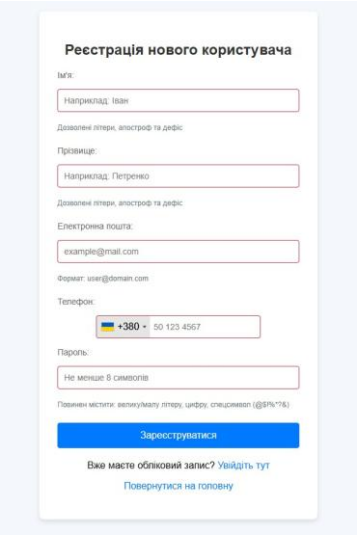
ЕКРАННІ ФОРМИ



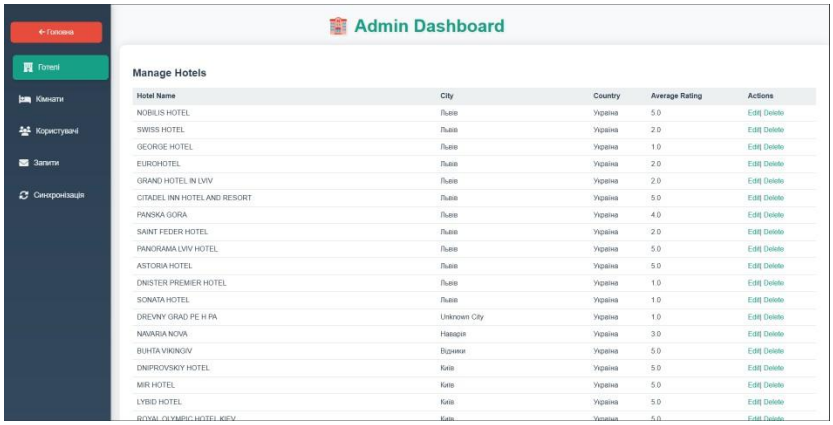
Головна сторінка

14

ЕКРАННІ ФОРМИ



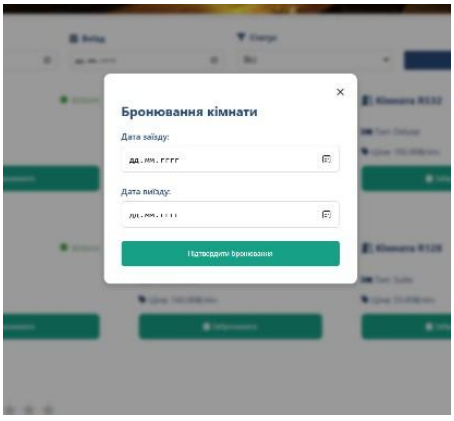
Форма реєстрації



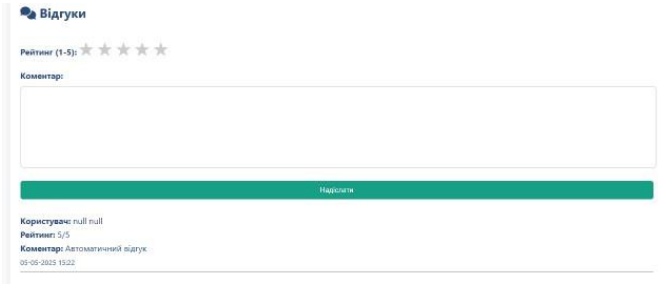
Сторінка адмін-панелі

15

ЕКРАННІ ФОРМИ



Форма
Бронювання



Форма з відгуками

16

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. 1. Філаретов М. Ю., Захист інформації у web-застосунку букінг готелів Матеріали XII Всеукраїнської науково-практичної конференції молодих учених «Інформаційні технології – 2025», 15.05.25, КСУ імені Бориса Грінченка, Київ, Україна, с. 336-337.
2. 2. Філаретов М. Ю., Гаманюк І. М. Використання діаграми послідовності для моделювання процесів веб застосунку для бронювання готелів // Матеріали VI Всеукраїнську науково-технічну конференцію «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 24.04.25,ДУІКТ, Київ, Україна, с. 136-138.

17

ВИСНОВКИ

1. Проведено аналіз існуючих систем готельного букінгу та виявленно їх недолікі.
2. Визначено засоби реалізації застосунку(Java, Spring, Amadeus, VC, Intelije, PostgreSQL).
3. Сформовано вимоги до web-застосунку, що стали основою для проектування та реалізації функцій застосунку.
4. Розроблено трьохшарову архітектуру застосунку, а саме відокремлено бізнес аналітику від презентаційного шару і шару доступа до даних. Ця незалежність сприятиме, при необхідності, легкої заміни технічних рішень презентаційного шару і бази даних.
5. Реалізовано функції застосунку. Отримано можливість здійснення онлайн-бронювання номерів в готелях.
6. Проведено тестування застосунку. Застосунок відповідає визначеним вимогам.

18

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
@Controller
@RequestMapping("/admin")
public class AdminController {

    private final HotelService hotelService;
    private final UserService userService;
    private final RoomService roomService;
    private final RoleRequestRepository
roleRequestRepository;

    @Autowired
    private EmailService emailService;


    @Autowired
    public AdminController(HotelService hotelService,
                           UserService userService,
                           RoomService roomService,
RoleRequestRepository roleRequestRepository) {
        this.hotelService = hotelService;
        this.userService = userService;
        this.roomService = roomService;
        this.roleRequestRepository =
roleRequestRepository;
    }

    @GetMapping
    public String adminPanel(Model model,
     @AuthenticationPrincipal
org.springframework.security.core.userdetails.User
securityUser) {

        List<Hotel> hotels = hotelService.listHotel();
        List<Room> rooms = roomService.getAllRooms();
        List<RoleRequest> requests =
roleRequestRepository.findByIsApprovedFalseAndReje
ctedFalse();

        model.addAttribute("hotels", hotels);
        model.addAttribute("rooms", rooms);
        model.addAttribute("requests", requests); // Додано
        model.addAttribute("searchedUsers",
Collections.emptyList());

        return "adminPanel";
    }

    @GetMapping("/editHotel/{hotelId}")
    public String showEditHotel(@PathVariable Long
hotelId, Model model) {
        Hotel hotel = hotelService.getHotelById(hotelId);
        model.addAttribute("hotel", hotel);
        return "edit-hotel";
    }

    @PostMapping("/editHotel/{hotelId}")
    public String editHotel(@PathVariable Long hotelId,
                            @ModelAttribute Hotel hotel,
                            RedirectAttributes redirectAttributes) {
        hotelService.UpdateHotel(hotelId, hotel);
        redirectAttributes.addFlashAttribute("success",
"Hotel updated successfully");
        return "redirect:/admin";
    }

    @GetMapping("/deleteHotel/{hotelId}")
    public String deleteHotel(@PathVariable Long
hotelId, RedirectAttributes redirectAttributes) {
        hotelService.deleteHotel(hotelId);
        redirectAttributes.addFlashAttribute("success",
"Hotel deleted successfully");
        return "redirect:/admin";
    }


    @GetMapping("/editRoom/{roomId}")
    public String showEditRoomForm(@PathVariable
Long roomId, Model model) {
        Room room = roomService.getRoomById(roomId);
        model.addAttribute("room", room);
        return "edit-room";
    }

    @PostMapping("/editRoom/{roomId}")
    public String editRoom(@PathVariable Long roomId,
@ModelAttribute Room roomData, RedirectAttributes
redirectAttributes) {
        Room room = roomService.getRoomById(roomId);


room.setRoomNumber(roomData.getRoomNumber());
        room.setType(roomData.getType());
        room.setPrice(roomData.getPrice());

room.setAvailableForDates(roomData.isAvailableForDat
es());

        room.setHotel(room.getHotel());

        roomService.saveRoom(room);
        redirectAttributes.addFlashAttribute("success",
"Кімнату оновлено");
        return "redirect:/admin";
    }

    @GetMapping("/deleteRoom/{roomId}")
    public String deleteRoom(@PathVariable Long
roomId, RedirectAttributes redirectAttributes) {
        roomService.deleteRoom(roomId);
        redirectAttributes.addFlashAttribute("success",
"Кімнату видалено");
        return "redirect:/admin";
    }
```

```
@PostMapping("/approve-request/{id}")
public String approvedRequest(
        @PathVariable Long id,
        RedirectAttributes redirectAttributes
) {
    RoleRequest request =
roleRequestRepository.findById(id)
            .orElseThrow(() -> new
RuntimeException("Запит не знайдено"));

    User user = request.getUser();
    user.getRoles().add(Role.ROLE_MANAGER);
    userService.saveUser(user);

    userService.approveRequest(id);

    String hotelName = request.getHotelName();

    userService.approveRequest(id);

    try {

emailService.sendRequestApprovedEmail(user.getEmail
(), hotelName);
        redirectAttributes.addFlashAttribute("success",
"Роль менеджера надано та лист відправлено");
    } catch (Exception e) {
        redirectAttributes.addFlashAttribute("error",
"Помилка відправки листа: " + e.getMessage());
    }

    return "redirect:/admin";
}


    @GetMapping("/searchUser")
    public String searchUser(@RequestParam String
email, Model model) {
        try {
            User user = userService.getUserByEmail(email);
            model.addAttribute("searchedUsers",
Collections.singletonList(user));
        } catch (RuntimeException e) {
            model.addAttribute("searchedUsers",
Collections.emptyList());
        }
        return "adminPanel :: #userResults";
    }

    @PostMapping("/grant-admin/{userId}")
    public String grantAdmin(@PathVariable Long
userId, RedirectAttributes redirectAttributes) {
        User user = userService.getUserById(userId);
        user.getRoles().add(Role.ROLE_ADMIN);
        userService.saveUser(user);
        redirectAttributes.addFlashAttribute("success",
"Admin role granted");
        return "redirect:/admin";
    }

    @PostMapping("/reject-request/{id}")
    public String rejectRequest(
            @PathVariable Long id,
            @RequestParam(required = false) String
comment,
            RedirectAttributes redirectAttributes
    ) {
        userService.rejectRequest(id, comment);
        redirectAttributes.addFlashAttribute("success",
"Запит відхилено");
        return "redirect:/admin";
    }

}
@Controller
@RequestMapping("/manager")
public class ManagerController {

    private final HotelService hotelService;
    private final UserService userService;
    private final RoomService roomService;

    @Autowired
    public ManagerController(HotelService hotelService,
UserService userService, RoomService roomService) {
        this.hotelService = hotelService;
        this.userService = userService;
        this.roomService = roomService;
    }

    @GetMapping("/hotels/{hotelId}")
    public String manageHotel(
            @PathVariable Long hotelId,
            @AuthenticationPrincipal
org.springframework.security.core.userdetails.User
securityUser,
            Model model
    ) {
        if (securityUser == null) {
            return "redirect:/login";
        }

        User user =
userService.getUserByEmail(securityUser.getUsername(
));
        if
(!user.getRoles().contains(Role.ROLE_MANAGER)) {
            return "redirect:/profile?error=Доступ
заборонено";
        }


        Hotel hotel =
hotelService.getHotelByIdAndOwnerId(hotelId,
user.getId());
        if (hotel == null) {
            return "redirect:/profile?error=Готель не
знайдено або доступ заборонено";
        }

        model.addAttribute("hotel", hotel);
        return "manage-hotel";
    }
```

```
    @GetMapping("/editHotel/{hotelId}")
    public String showEditHotelForm(
            @PathVariable Long hotelId,
            @RequestParam("userId") Long userId,
            Model model
    ) {
        Hotel hotel =
hotelService.getHotelByIdAndOwnerId(hotelId, userId);
        model.addAttribute("hotel", hotel);
        model.addAttribute("userId", userId);
        return "manager-edit-hotel";
    }

    @PostMapping("/editHotel/{hotelId}")
    public String editHotel(
            @PathVariable Long hotelId,
            @AuthenticationPrincipal
org.springframework.security.core.userdetails.User
securityUser,
            @ModelAttribute("hotel") Hotel hotelData,
            RedirectAttributes redirectAttributes
    ) {
        User user =
userService.getUserByEmail(securityUser.getUsername(
));
        Hotel existingHotel =
hotelService.getHotelByIdAndOwnerId(hotelId,
user.getId());
        existingHotel.setName(hotelData.getName());
        existingHotel.setAddress(hotelData.getAddress());
        existingHotel.setCity(hotelData.getCity());
        existingHotel.setCountry(hotelData.getCountry());

existingHotel.setContactInfo(hotelData.getContactInfo())
;


        hotelService.UpdateHotel(hotelId,existingHotel);
        redirectAttributes.addAttribute("userId",
user.getId());
        return "redirect:/manager/hotels/{hotelId}";
    }

    @GetMapping("/deleteHotel/{hotelId}")
    public String deleteHotel(
            @PathVariable Long hotelId,
            @AuthenticationPrincipal
org.springframework.security.core.userdetails.User
securityUser,
            RedirectAttributes redirectAttributes
    ) {
        hotelService.deleteHotel(hotelId);
        User user =
userService.getUserByEmail(securityUser.getUsername(
));
        redirectAttributes.addAttribute("userId",
user.getId());
        return "redirect:/profile";
    }


    @GetMapping("/editRoom/{roomId}")
    @ResponseBody
    @CrossOrigin(origins = "http://localhost:8080")
    public ResponseEntity<Room>
getRoomForEdit(@PathVariable Long roomId) {
        Room room = roomService.getRoomById(roomId);
        room.setHotel(null);
        return ResponseEntity.ok(room);
    }

    @PostMapping("/editRoom/{roomId}")
    public String editRoom(
            @PathVariable Long roomId,
            @ModelAttribute Room roomData,
            @AuthenticationPrincipal
org.springframework.security.core.userdetails.User
securityUser,
            RedirectAttributes redirectAttributes
    ) {
        System.out.println("Отримано запит на оновлення
кімнати ID: " + roomId);
        System.out.println("Новий номер: " +
roomData.getRoomNumber());
        Room room = roomService.getRoomById(roomId);

room.setRoomNumber(roomData.getRoomNumber());
        room.setType(roomData.getType());
        room.setPrice(roomData.getPrice());

room.setAvailableForDates(roomData.isAvailableForDat
es());
        roomService.saveRoom(room);
        User user =
userService.getUserByEmail(securityUser.getUsername(
));
        redirectAttributes.addAttribute("hotelId",
room.getHotel().getId());
        redirectAttributes.addAttribute("userId",
user.getId());
        return "redirect:/manager/hotels/{hotelId}";
    }

    @GetMapping("/hotels/{hotelId}/rooms/create")
    public String showCreateRoomForm(
            @PathVariable Long hotelId,
            @AuthenticationPrincipal
org.springframework.security.core.userdetails.User
securityUser,
            Model model
    ) {
        Room room = new Room();
        User user =
userService.getUserByEmail(securityUser.getUsername(
));

room.setHotel(hotelService.getHotelByIdAndOwnerId(h
otelId, user.getId()));
        model.addAttribute("room", room);
        model.addAttribute("userId", user.getId());
        return "manager-create-room";
    }


    @PostMapping("/hotels/{hotelId}/rooms/create")
```

```
    public String createRoom(
            @PathVariable Long hotelId,
            @AuthenticationPrincipal
org.springframework.security.core.userdetails.User
securityUser,
            @ModelAttribute("room") Room room,
            RedirectAttributes redirectAttributes
    ) {
        User user =
userService.getUserByEmail(securityUser.getUsername(
));
        Hotel hotel =
hotelService.getHotelByIdAndOwnerId(hotelId,
user.getId());
        room.setHotel(hotel);
        roomService.saveRoom(room);

        redirectAttributes.addAttribute("userId",
user.getId());
        return "redirect:/manager/hotels/{hotelId}";
    }

    @GetMapping("/create-hotel")
    public String showCreateHotelForm(
            @AuthenticationPrincipal
org.springframework.security.core.userdetails.User
securityUser,
            Model model
    ) {
        User user =
userService.getUserByEmail(securityUser.getUsername(
));
        CreateHotelWithRoomsDTO dto = new
CreateHotelWithRoomsDTO();
        dto.getRooms().add(new CreateRoomDTO());
        model.addAttribute("hotelDTO", dto);
        model.addAttribute("userId", user.getId());
        return "create-hotel";
    }

    @PostMapping("/create-hotel")
    public String createHotel(
            @ModelAttribute("hotelDTO")
CreateHotelWithRoomsDTO hotelDTO,
            @AuthenticationPrincipal
org.springframework.security.core.userdetails.User
securityUser,
            RedirectAttributes redirectAttributes
    ) {
        if (securityUser == null) {
            redirectAttributes.addFlashAttribute("error",
"Будь ласка, увійдіть у систему");
            return "redirect:/login";
        }

        User user =
userService.getUserByEmail(securityUser.getUsername(
));
        if (user == null) {
            redirectAttributes.addFlashAttribute("error",
"Користувач не знайдений");
            return "redirect:/profile";
        }


        hotelDTO.setUserId(user.getId());

        try {
            List<String> photoUrls = new ArrayList<>();
            if (hotelDTO.getPhotos() != null) { // Додано
перевірку на null
                for (MultipartFile file : hotelDTO.getPhotos())
{
                    if (!file.isEmpty()) {
                        String fileName =
System.currentTimeMillis() + "_" +
file.getOriginalFilename();
                        Path uploadPath = Paths.get("uploads/" +
fileName);

Files.createDirectories(uploadPath.getParent());
                        Files.copy(file.getInputStream(),
uploadPath,
StandardCopyOption.REPLACE_EXISTING);
                        photoUrls.add("/uploads/" + fileName);
                    }
                }
            }
            hotelDTO.setPhotoUrls(photoUrls);

            hotelService.createHotelWithRooms(hotelDTO);
            redirectAttributes.addFlashAttribute("success",
"Готель створено!");
        } catch (Exception ex) {
            redirectAttributes.addFlashAttribute("error",
"Помилка завантаження файлів: " + ex.getMessage());
        }

        return "redirect:/profile";
    }

    @GetMapping("/deleteRoom/{roomId}")
    public String deleteRoom(
            @PathVariable Long roomId,
            @AuthenticationPrincipal
org.springframework.security.core.userdetails.User
securityUser,
            @RequestParam("hotelId") Long hotelId,
            RedirectAttributes redirectAttributes
    ) {
        roomService.deleteRoom(roomId);
        User user =
userService.getUserByEmail(securityUser.getUsername(
));
        redirectAttributes.addAttribute("hotelId", hotelId);
        redirectAttributes.addAttribute("userId",
user.getId());
        return "redirect:/manager/hotels/{hotelId}";
    }

    @PostMapping("/hotels/{hotelId}/photos")
    public String uploadPhotos(
            @PathVariable Long hotelId,
            @RequestParam("photos") MultipartFile[] files,
```

```
        @AuthenticationPrincipal
org.springframework.security.core.userdetails.User
securityUser,
        RedirectAttributes redirectAttributes) throws
IOException {

    User user =
userService.getUserByEmail(securityUser.getUsername(
));
    Hotel hotel =
hotelService.getHotelByIdAndOwnerId(hotelId,
user.getId());
    List<String> newPhotos =
processUploadedFiles(files);

    List<String> updatedPhotos = new
ArrayList<>(hotel.getPhotoUrls());
    updatedPhotos.addAll(newPhotos);

    hotel.setPhotoUrls(updatedPhotos);
    hotelService.UpdateHotel(hotelId, hotel);

    redirectAttributes.addFlashAttribute("success",
"Фото успішно додані");
    return "redirect:/manager/hotels/" + hotelId;
  }


  @PostMapping("/hotels/{hotelId}/photos/delete")
  public String deletePhoto(
        @PathVariable Long hotelId,
        @RequestParam String photoUrl,
        @AuthenticationPrincipal
org.springframework.security.core.userdetails.User
securityUser,
        RedirectAttributes redirectAttributes) {

    System.out.println("Deleting photo: " + photoUrl);
// Логування
    User user =
userService.getUserByEmail(securityUser.getUsername(
));
    Hotel hotel =
hotelService.getHotelByIdAndOwnerId(hotelId,
user.getId());

    List<String> updatedPhotos = new
ArrayList<>(hotel.getPhotoUrls());
    updatedPhotos.remove(photoUrl);

    hotel.setPhotoUrls(updatedPhotos);
    hotelService.UpdateHotel(hotelId, hotel);

    redirectAttributes.addFlashAttribute("success",
"Фото видалено");
    return "redirect:/manager/hotels/" + hotelId;
  }

  private List<String>
processUploadedFiles(MultipartFile[] files) throws
IOException {
    List<String> photoUrls = new ArrayList<>();
    for (MultipartFile file : files) {
      if (!file.isEmpty()) {
        String fileName = UUID.randomUUID() + "_"
+ file.getOriginalFilename();
        Path uploadPath = Paths.get("uploads/" +
fileName);

Files.createDirectories(uploadPath.getParent());
        Files.copy(file.getInputStream(), uploadPath,
StandardCopyOption.REPLACE_EXISTING);
        photoUrls.add("/uploads/" + fileName);
      }
    }
    return photoUrls;
  }
}

@Controller
@RequestMapping("/profile")
public class ProfileController {

  private final UserService userService;
  private final HotelService hotelService;
  private final HotelRepository hotelRepository;

  @Autowired
  public ProfileController(UserService userService,
                HotelService hotelService,
HotelRepository hotelRepository) {
    this.userService = userService;
    this.hotelService = hotelService;
    this.hotelRepository = hotelRepository;
  }

  @GetMapping
  @Transactional
  public String showProfile(@AuthenticationPrincipal
org.springframework.security.core.userdetails.User
securityUser,
                Model model) {
    if (securityUser == null) {
      return "redirect:/login";
    }

    try {
      User fullUser =
userService.getUserByEmail(securityUser.getUsername(
));
      model.addAttribute("user", fullUser);

      boolean isManager =
fullUser.getRoles().contains(Role.ROLE_MANAGER);
      model.addAttribute("isManager", isManager);

      if (isManager) {
        List<Hotel> userHotels =
hotelService.getHotelsByManagerId(fullUser.getId());
        model.addAttribute("userHotels", userHotels);
      }

      return "profile";
    } catch (RuntimeException ex) {
```

```
            model.addAttribute("error", ex.getMessage());
            return "redirect:/login";
        }
    }

    @PostMapping("/request-manager")
    public String requestManagerRole(
            @AuthenticationPrincipal
org.springframework.security.core.userdetails.User
securityUser,
            @RequestParam String hotelName,
            @RequestParam String hotelAddress,
            RedirectAttributes redirectAttributes,
            HttpServletRequest request
    ) {
        try {
            User user =
userService.getUserByEmail(securityUser.getUsername(
));
            userService.requestManagerRole(user.getId(),
hotelName, hotelAddress);
            redirectAttributes.addFlashAttribute("message",
"Запит відправлено! Очікуйте підтвердження.");
            System.out.println("Відправлено повідомлення:
" + "Запит відправлено! Очікуйте підтвердження.");

request.getSession().setAttribute("showMessageModal",
"Запит відправлено! Очікуйте підтвердження.");
        } catch (RuntimeException ex) {
            redirectAttributes.addFlashAttribute("error",
ex.getMessage());
            System.out.println("Відправлено помилку: " +
ex.getMessage());
        }
        return "redirect:/profile";
    }
}
```