МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВОЛИНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЛЕСІ УКРАЇНКИ
Кафедра комп'ютерних наук та кібербезпеки

На правах рукопису

ЛАЩУК МАКСИМ ТАРАСОВИЧ

ПРОЄКТУВАННЯ ТА РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ АВТОМАТИЗОВАНОГО ПРОЦЕСУ БРОНЮВАННЯ НОМЕРІВ БАЗИ ВІДПОЧИНКУ

(на прикладі бази «Гарт»)

Спеціальність: 122 Комп'ютерні науки
Освітньо-професійна програма: Комп'ютерні науки та інформаційні технології
Робота на здобуття освітнього ступеня «магістр»

Науковий керівник:
ГОЛОВІН МИКОЛА БОРИСОВИЧ,
кандидат фізико-математичних наук,
доцент кафедри комп'ютерних наук
та кібербезпеки

РЕКОМЕНДОВАНО ДО ЗАХИСТУ
Протокол №_____
засідання кафедри комп'ютерних наук
та кібербезпеки
від _____ 2025 р.
Завідувач кафедри
(_____) Гришанович Т.О.

ЛУЦЬК 2025

ЗМІСТ

ВСТУП

В умовах цифрової трансформації та зростаючої конкуренції в індустрії гостинності ефективність управління операційними процесами є ключовим фактором успіху. Багато закладів відпочинку, зокрема регіональні бази, досі покладаються на застарілі або ручні методи управління бронюванням, такі як телефонні дзвінки, паперові журнали або електронні таблиці. Такий підхід збільшує адміністративне навантаження на персонал та призводить до фінансових втрат через помилки людського фактора, зокрема подвійне бронювання, а також унеможливлює гнучке реагування на ринковий попит. Відсутність автоматизованої системи обмежує можливості впровадження сучасних практик управління дохідністю, зокрема динамічного ціноутворення. Таким чином, розробка інтелектуального вебзастосунку, що автоматизує процес бронювання та інтегрує алгоритми оптимізації цін, є актуальною практичною задачею. На прикладі бази відпочинку «Гарт» дана робота спрямована на підвищення операційної ефективності та максимізацію прибутку за рахунок використання сучасних комп’ютерних технологій.

Метою роботи є проєктування та розробка вебзастосунку для автоматизованого процесу бронювання номерів на прикладі бази відпочинку «Гарт».

Для досягнення поставленої мети було визначено наступні задачі:

1. Провести аналіз предметної області та сучасних підходів до розроблення вебзастосунків для визначення основних тенденцій, переваг і недоліків існуючих рішень.

2. Обґрунтувати вибір моделі життєвого циклу програмного забезпечення та технологічного середовища, що забезпечує ефективність процесу розроблення.

3. Реалізувати серверну частину системи з урахуванням вимог до безпеки, цілісності даних та ефективності бізнес-логіки.

4. Забезпечити інтеграцію засобів автоматизованого інформування користувачів щодо результатів взаємодії із системою.

5. Створити інтерфейс користувача з адаптивним дизайном, який забезпечує зручність взаємодії та ефективну візуалізацію даних.

6. Провести тестування розробленої системи для перевірки її працездатності, надійності та відповідності поставленим вимогам.

Об’єктом дослідження є процес організації та управління бронюванням і формуванням цінової політики у закладах відпочинку.

Предметом дослідження є архітектурні підходи, методи, алгоритми та програмні засоби, що забезпечують розроблення автоматизованого вебзастосунку для системи бронювання.

Практичне значення отриманих результатів. Розроблений програмний засіб готовий до впровадження і має практичну цінність для бази «Гарт», оскільки дозволяє автоматизувати рутинні операції, виключити помилки подвійного бронювання. Для адміністраторів створено зручний візуальний інструмент для моніторингу та управління номерним фондом. Робота може слугувати архітектурним шаблоном для створення аналогічних систем автоматизації в малих та середніх закладах гостинності в Україні.

Апробація результатів роботи. Результати даної кваліфікаційної роботи були представлені на наукових конференціях:

IX Міжнародна студентська конференція «Актуальні питання та перспективи проведення наукових досліджень», м. Рівне, 28 листопада 2025 р.

XVI Міжнародна науково-практична конференція «Практичні та теоретичні питання розвитку науки та освіти» м. Львів, 22-23 листопада 2025 р.

За результатами конференції було опубліковано тези в збірниках матеріалів даних конференцій.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ СИСТЕМИ БРОНЮВАННЯ НОМЕРІВ БАЗИ ВІДПОЧИНКУ

1.1 Аналіз предметної області систем бронювання

Електронні послуги істотно трансформували процеси онлайн-комунікації, зокрема з огляду на підвищення доступності інформації, що сприяє полегшенню процесу порівняння, забезпеченню прозорості ринків, розширенню інформаційного різноманіття та підвищенню інтерактивності вебресурсів [1]. Сфера онлайн-бронювання набула значного поширення та ефективності в різних галузях діяльності, охоплюючи резервування готельних номерів, бібліотечних аудиторій, придбання квитків у кінотеатри та інші заклади.

У роботі [2] обґрунтовано значення впровадження системи бронювання приміщень у межах ширшої стратегії створення спільних робочих просторів. Розглянуто підходи до ефективного управління такими системами та проаналізовано можливості використання сервісу Google Календар як інструменту для бронювання приміщень. Установлено, що цей сервіс може виступати безкоштовною, хоча й спрощеною системою бронювання, проте потребує більшого обсягу технічного обслуговування та адміністрування порівняно з комерційними або спеціалізованими відкритими рішеннями. У дослідженні [3] розглянуто модернізацію системи бронювання аудиторій університетської бібліотеки, спрямовану на забезпечення зручності для студентів і мінімізацію участі працівників бібліотеки у процесах керування бронюванням і доступом до приміщень. Розвиток системи здійснювався на основі аналізу користувацьких відгуків, що дало змогу підвищити її ефективність і відповідність реальним потребам.

Відгуки користувачів становлять один із ключових чинників під час вибору житла з позиції споживача. Клієнти, як правило, виходять із припущення, що об’єкт із більшою кількістю позитивних оцінок забезпечить аналогічний позитивний досвід користування послугами проживання. У роботі [4] акцентовано увагу на значущості клієнтських відгуків щодо видів тимчасового

житла. Проведено аналітичне дослідження відгуків споживачів, яке підтвердило їхню роль як вагомого критерію при виборі та бронюванні закладів проживання.

У роботі [5] розглянуто вплив візуального оформлення та юзабіліті-дизайну фірмових мобільних застосунків для бронювання готелів на клієнтський досвід. Проаналізовано не лише прямий вплив дизайну на зручність і задоволення користувачів, а й модифікуючу роль мети перебування у процесі взаємодії з цифровими сервісами.

Можна зробити висновок, що функціональність і дизайн програмного забезпечення для сервісів бронювання виступають важливою складовою ефективності бізнес-процесів у сфері гостинності та безпосередньо впливають на рівень клієнтської лояльності.

На сучасному ринку бронювання житла функціонують численні міжнародні платформи, що активно надають свої послуги як в Україні, так і за її межами (зокрема, *Booking.com*, *Airbnb*, *Expedia*). Проте такі системи не завжди враховують економічні, фінансові та нормативні особливості окремих країн, що знижує їхню ефективність у локальному контексті.

Серед основних проблем, з якими стикаються місцеві орендодавці, варто виокремити:

– високий рівень комісійних відрахувань на користь міжнародних сервісів;

– складність налаштування та технічного супроводу власних вебресурсів;

– відсутність повноцінної адаптації під національну валюту та локальні платіжні системи;

– залежність бізнес-процесів від іноземних платформ і зовнішньої інфраструктури.

Попри зазначені обмеження, загальна тенденція розвитку галузі визначається процесами цифровізації та автоматизації, які сприяють переходу від традиційних моделей бронювання до інтерактивних онлайн-рішень. Сучасні користувачі орієнтуються на швидкість, зручність та інтуїтивність процесу, очікуючи отримання послуги у кілька кліків без зайвих операцій.

Для представників малого та середнього бізнесу актуальною є потреба у створенні доступних, технологічно простих і водночас ефективних інструментів, що не вимагають значних фінансових витрат або спеціалізованих технічних знань.

Можна констатувати недостатній рівень наявних готових рішень, орієнтованих на український ринок. Це зумовлює необхідність розроблення програмного продукту, який забезпечить простоту використання, адаптованість до місцевих умов та сприятиме підвищенню ефективності діяльності власників об’єктів розміщення малого та середнього сегменту.

Інформаційна система бронювання – це сукупність апаратних та програмних засобів, даних та бізнес-логіки, що забезпечують пошук доступності, прийом та підтвердження замовлень (бронювань), керування інвентарем номерів, обробкою платежів і взаємодію з каналами продажу. Сучасні системи бронювання функціонують як багаторівневі платформи, які поєднують локальні рішення, модулі online-бронювання, channel-менеджери та API-інтеграції для зовнішніх платформ [6].

Для невеликих і середніх баз відпочинку це створює потребу у використанні інтегрованих, але водночас масштабованих інформаційних рішень. Найбільш ефективною моделлю є поєднання простого механізму онлайн-бронювання для прийому прямих заявок, системи управління об’єктом розміщення (PMS) для контролю процесів заселення та спеціалізованого channel-менеджера для синхронізації з агрегаторами і зовнішніми системами бронювання [7], [8].

Класифікацію інформаційних систем бронювання доцільно здійснювати за кількома незалежними ознаками, що відображають архітектурні, функціональні, цільові та технологічні особливості даних застосунків. Такий підхід дозволяє систематизувати існуючі рішення та визначити тенденції розвитку сучасних інтелектуальних платформ у сфері гостинності (Таблиця 1.1).

Таблиця 1.1.

Класифікація інформаційних систем бронювання [9]-[12].

Ознака класифікації	Тип системи	Характеристика
Архітектура розгортання	Локальні (on-premises)	Функціонують автономно на сервері підприємства; не залежать від підключення до Інтернету; потребують технічного обслуговування та ручного оновлення.
	Хмарні (SaaS)	Реалізовані як сервіси за підпискою; забезпечують швидке масштабування, автоматичне оновлення та інтеграцію з каналами продажу.
Функціональна роль	PMS (Property Management System)	Система управління номерним фондом, заселенням, гостями, фінансами та звітністю.
	Booking Engine	Модуль онлайн-бронювання, інтегрований у вебсайт або доступний через віджет; забезпечує прийом і підтвердження бронювань.
	Channel Manager	Інструмент синхронізації тарифів і наявності номерів між PMS і зовнішніми агрегаторами.
	GDS/OTA-інтегратори	Платформи підключення до глобальних дистрибутивних мереж або онлайн-агрегаторів, що розширюють канали збуту.
Цільова аудиторія	Корпоративні рішення	Призначені для великих готельних мереж; характеризуються високим рівнем кастомізації та інтеграції з внутрішніми ERP-системами.
	Рішення для малого бізнесу	Орієнтовані на невеликі бази відпочинку; мають спрощений інтерфейс, базовий набір функцій і доступну вартість.
	Агрегатори (OTA)	Платформи, що надають єдиний інтерфейс бронювання для багатьох закладів одночасно.

Рівень інтелектуалізації	Традиційні системи	Використовують статичну бізнес-логіку без елементів адаптивного аналізу даних.
	Інтелектуальні системи	Впроваджують автоматизовані алгоритми, зокрема динамічне ціноутворення, рекомендаційні модулі та чат-боти для клієнтської підтримки.

У сучасних умовах розвитку цифрових технологій спостерігається тенденція переходу від локальних традиційних систем до хмарних інтелектуалізованих рішень. Такі системи забезпечують високу гнучкість, адаптивність і можливість інтеграції з глобальними каналами бронювання, що підвищує ефективність управління доходами та рівень сервісу для клієнтів.

1.2. Принципи функціонування систем бронювання номерів баз відпочинку

Система бронювання номерів у закладах відпочинку являє собою інтегроване програмне середовище, призначене для автоматизації процесів перевірки доступності номерів, створення та підтвердження бронювань, опрацювання фінансових транзакцій, синхронізації з зовнішніми каналами продажів та ведення внутрішнього обліку клієнтів у системах управління готельними процесами.

Ефективність функціонування таких систем є ключовим чинником забезпечення операційної стабільності та прибутковості діяльності закладів відпочинку [13].

Системи бронювання номерів баз відпочинку реалізують комплекс програмно-інформаційних механізмів, спрямованих на автоматизацію процесів пошуку, вибору, резервування та оплати місць розміщення. Їхня робота ґрунтується на низці фундаментальних принципів, що забезпечують точність, доступність та надійність інформації для користувачів та адміністраторів.

1. Централізоване зберігання та доступ до даних

Централізоване сховище в контексті CRS (Central Reservation System) – це логічно єдиний репозиторій даних, який агрегує й синхронізує інформацію про інвентар (номери), тарифи, історію бронювань, клієнтів та фінансові транзакції з усіх каналів дистрибуції (веб-портал, мобільний застосунок, агентські системи, PMS). Така централізація виступає архітектурною основою для забезпечення єдності господарської діяльності, полегшуючи оновлення, звітність і подальші інтеграції з зовнішніми системами [14].

У центрі моделі даних для бази відпочинку зазвичай виокремлюють наступні ключові сутності та їхні обов'язкові атрибути [15]:

– Room / Unit (Інвентар номерів) – ідентифікатор, тип (наприклад, стандарт/люкс), місткість, оснащення, географічне розташування на сервісі, статус доступності.

– Rate / Tariff (Тарифи) – базова ціна, сезонні коригування, правила зміни ціни, правила скасування, пакети послуг.

– Availability / Inventory (Наявність) – календар доступності, блокування/резервації, правила мінімального/максимального перебування.

– Booking (Бронювання) – унікальний ID бронювання, дата створення, період перебування, пов'язана одиниця інвентарю, послуги, статус (попереднє/підтверджене/скасоване).

– Customer / Guest (Клієнт) – ідентифікатори, контактна інформація, історія взаємодій, програма лояльності.

– Payment / Transaction (Платіж) – метод оплати, статус платежу, повернення, зв'язок з бронюванням.

Реляційні моделі часто формалізуються у вигляді ER-діаграм з нормалізованими таблицями для уникнення аномалій оновлення; у NoSQL-підходах частина даних (наприклад, денний інвентар або денні тарифи) може зберігатися як вкладені документи для швидкого читання. Як приклад на рисунку 1.1. показано діаграма ER для управління готелем [15].

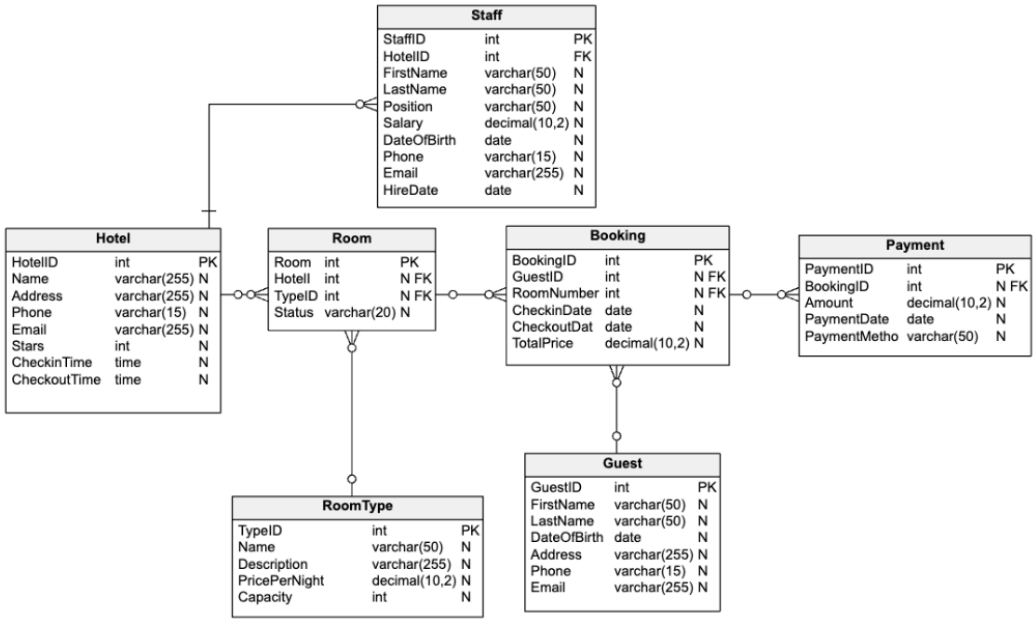


Рисунок 1.1 – Діаграма ER для управління готелем

Переваги централізації проявляються у підвищенні узгодженості, уніфікації та зменшенні дублювання даних. Централізоване зберігання інформації дає змогу уникати розбіжностей між різними каналами доступу, наприклад між веб-інтерфейсом і call-центром. Узгодженість забезпечується завдяки застосуванню єдиного джерела достовірних даних, коли всі оновлення тарифів та інвентарю виконуються через централізований API або бекенд-систему [16].

Додаткову цілісність надає нормалізація даних і використання обмежень у базі даних, таких як унікальність записів і зовнішні ключі, що мінімізує дублювання та усуває несумісні дані. Важливим елементом також є механізми контролю версій і журналювання, які забезпечують фіксацію та аналіз історії змін, а за потреби – відновлення попередніх станів.

Застосування цих принципів знижує когнітивне навантаження на адміністративний персонал і сприяє підвищенню точності аналітичних процесів.

Забезпечення узгодженості даних та запобігання випадкам подвійного бронювання є ключовою вимогою до систем керування інвентарем, у яких одночасно взаємодіє багато клієнтів. Центральною операцією виступає атомарне резервування одиниці інвентарю на визначений період, що потребує використання спеціалізованих механізмів контролю конкурентного доступу [17].

Одним із базових засобів є транзакційні моделі з гарантіями ACID, які забезпечують повне виконання операції, тим самим підтримуючи коректний стан інвентарю та виключаючи часткові оновлення [18].

У практиці також застосовують різні підходи до блокування. Песимістичне блокування передбачає ізоляцію запису на час виконання операції, що є ефективним у випадках високої конкуренції за один і той самий ресурс. Оптимістичне блокування ґрунтується на контролі версій і повторному виконанні транзакцій у разі виявлення конфліктів, тому воно більш ефективне у системах, де домінують операції читання.

До засобів бізнес-логіки належать тимчасові резервування без оплати, так звані hold-операції, які автоматично анулюються після завершення визначеного інтервалу часу. Додатково використовують обмеження кількості одночасних сесій та механізми чергування транзакцій для впорядкованої обробки конкурентних запитів [19].

У розподілених системах забезпечення узгодженості потребує застосування координованих транзакцій та алгоритмів узгодження на кшталт двофазного коміту. Поширені також спеціалізовані сервіси резервування та використання кешів із терміном дії або атомарних операцій у key-value сховищах, які дають змогу мінімізувати логічні конфлікти під час високонавантажених транзакційних потоків.

Нижче наведено приклад сценарію створення бронювання номерів бази відпочинку (Рис. 1.2). Сценарій демонструє послідовність взаємодії між користувачем, інтерфейсом вебзастосунку, бекенд-сервісами та базою даних.

Процес створення бронювання складається з таких етапів:

1. Користувач передає параметри пошуку (дати, кількість осіб, тип номеру).
2. Frontend відправляє запит до Booking API.
3. Booking API передає запит до AvailabilityService для перевірки доступності номерів.

4. AvailabilityService звертається до бази даних (RoomRepository, BookingRepository).

5. Після підтвердження наявності, користувач надсилає запит на створення бронювання.

6. Booking API викликає BookingService, який створює запис у БД та генерує підтвердження.

7. Система повертає результат у вигляді успішного бронювання.

У структурі представленої системи кожен компонент виконує специфічну функцію. Користувач виступає ініціатором операцій пошуку та оформлення бронювання. Фронтенд забезпечує інтерфейс взаємодії з користувачем і формує відповідні HTTP-запити до серверної частини. Компонент BookingAPI виконує роль центрального керуючого модуля бекенд-системи, спрямовуючи запити до відповідних сервісів. AvailabilityService відповідає за перевірку доступності номерів, тоді як BookingService реалізує логіку створення бронювання та керування транзакційними операціями. Усі дані зберігаються у центральній реляційній базі даних, реалізованій на основі PostgreSQL.

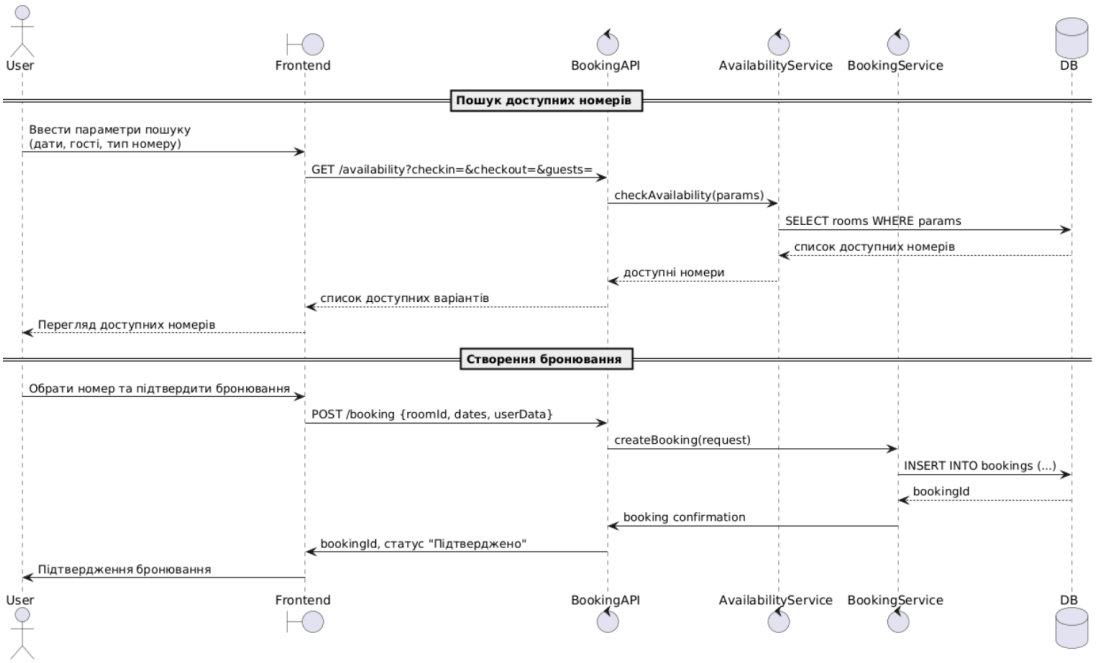


Рисунок 1.2 – Процес створення бронювання

Етап перевірки доступності передбачає виконання AvailabilityService відповідних SQL-запитів, що включають визначення номерів із заданими

параметрами та відсіювання тих, що вже мають активні бронювання на вибраний період.

Після підтвердження доступності ініціюється процес створення бронювання. BookingService виконує його у межах транзакції, що охоплює додавання нового запису до таблиці *booking* та подальше фіксування змін. Після успішного завершення операції формується й повертається об’єкт бронювання як відповідь на запит.

2. Принцип актуальності та цілісності інформації

Принцип актуальності та цілісності даних визначає здатність системи бронювання забезпечувати достовірність інформації щодо доступності номерів у кожен момент часу та підтримувати узгодженість усіх операцій, пов’язаних із бронюваннями. Для інформаційних систем такого типу критично важливо, щоб жодні два користувачі не змогли забронювати один і той самий номер у перетині дат, а всі дії з даними були атомарними, відновлюваними та несуперечливими.

У більшості сучасних архітектур систем бронювання центральне місце посідають реляційні системи керування базами даних. Їхня робота ґрунтується на транзакційній моделі, що визначається властивостями ACID, які гарантують коректність оброблення операцій.

Створення бронювання виконується як єдина неподільна операція. Усі її етапи – перевірка доступності, додавання запису до бази даних та підтвердження змін – мають завершитися успішно. У разі виникнення помилки транзакція повністю відхиляється. Такий підхід виключає можливість появи частково зафіксованих бронювань.

Система гарантує, що база даних переходить лише між станами, які відповідають усім правилам цілісності. Це включає, зокрема, заборону перетину дат у бронюваннях для одного номера. У PostgreSQL дотримання цих правил підтримується механізмами перевірок цілісності, тригерами та спеціально визначеними обмеженнями.

Паралельні транзакції виконуються таким чином, щоб їхні проміжні стани не впливали одна на одну. У системах бронювання поширеним є використання

рівня READ COMMITTED, тоді як для операцій, що вимагають максимального захисту від конфліктів, застосовують рівні REPEATABLE READ або SERIALIZABLE, які дозволяють мінімізувати ризик одночасного створення конфліктних бронювань.

Після підтвердження транзакції її результати залишаються збереженими навіть у разі виникнення технічних збоїв. Це є критичним, оскільки процес бронювання часто включає фінансові операції та взаємодію зі сторонніми сервісами.

Як зазначають дослідники [20], ACID-модель продовжує залишатися фундаментальним стандартом для транзакційних систем, що працюють у режимі реального часу.

Проблематика паралельного виконання операцій у системах бронювання є однією з ключових у сучасних дослідженнях інформаційних систем. На основі напрацювань [21] сформовано перелік підходів, які вважаються найбільш ефективними для мінімізації конфліктів.

1. Блокування рядків у межах транзакції (Row-Level Locks). У PostgreSQL для забезпечення виняткового доступу використовується блокування вибраного ресурсу за допомогою запиту:

```
SELECT * FROM rooms WHERE id = X FOR UPDATE.
```

Цей механізм гарантує, що номер не може бути змінений іншими транзакціями до завершення поточної операції.

2. Виявлення перетину дат у транзакції. Система виконує перевірку на наявність бронювань, що перекриваються з указаним періодом. Операція здійснюється з фіксацією блокування, наприклад:

```
SELECT * FROM bookings
WHERE room_id = X
  AND daterange(checkin, checkout) && daterange($dates)
FOR SHARE;
```

Це забезпечує узгоджене читання стану та попереджає появу конфліктних записів.

3. Оптимістичне блокування. Цей підхід є доцільним для високонавантажених архітектур, зокрема документноорієнтованих баз даних або мікросервісних систем. Кожен запис має версію, яка змінюється під час оновлення. Перед фіксацією транзакції система перевіряє відповідність версій, що дозволяє виявити зміну стану ресурсу іншим процесом [20].

4. Механізми чергування транзакцій і повторного виконання. У практиці великих платформ застосовується автоматичне повторення транзакцій у разі виявлення конфлікту або взаємного блокування. Такий підхід забезпечує стабільність обробки запитів в умовах високої конкурентності та зменшує частоту відмов системи [21].

На рисунку 1.3 подано структуровану та деталізовану UML діаграма активності, що описує механізм перевірки доступності номерів у системі бронювання. Діаграма відтворює бізнес-логіку на операційному рівні та демонструє послідовність дій, за допомогою яких система опрацьовує запит користувача щодо визначення доступних номерів у заданому часовому інтервалі.

На початковому етапі система приймає від фронтенд-компонента набір параметрів, що включає дати заїзду та виїзду, тип номеру і кількість гостей. Після отримання даних виконується їхня валідація, яка охоплює перевірку коректності формату дат, узгодженість календарного діапазону (умова *check-in* < *check-out*) та відповідність зазначеної кількості гостей допустимій місткості обраної категорії номерів.

Після успішної перевірки параметрів система здійснює вибірку номерів, що відповідають визначеним критеріям. На цьому етапі виконується фільтрація за типом номеру, максимальною місткістю та іншими характеристиками, якщо такі параметри передбачені структурою даних або бізнес-логікою системи.

3. Інтеграція платіжних сервісів і захист транзакцій

Функціонування сучасних систем бронювання неможливе без надійної підтримки електронних платежів, оскільки цей компонент визначає ефективність фінансових операцій та рівень довіри користувачів. Інтеграція з платіжними сервісами дає змогу реалізувати автоматизовану обробку транзакцій,

забезпечити своєчасну фіксацію платежів і сформувати прозору логіку взаємодії між клієнтом та адміністрацією закладу. Сучасні дослідження з електронної комерції підкреслюють, що якість інтеграції платіжної інфраструктури безпосередньо впливає на користувацький досвід і конверсію бронювань [22].

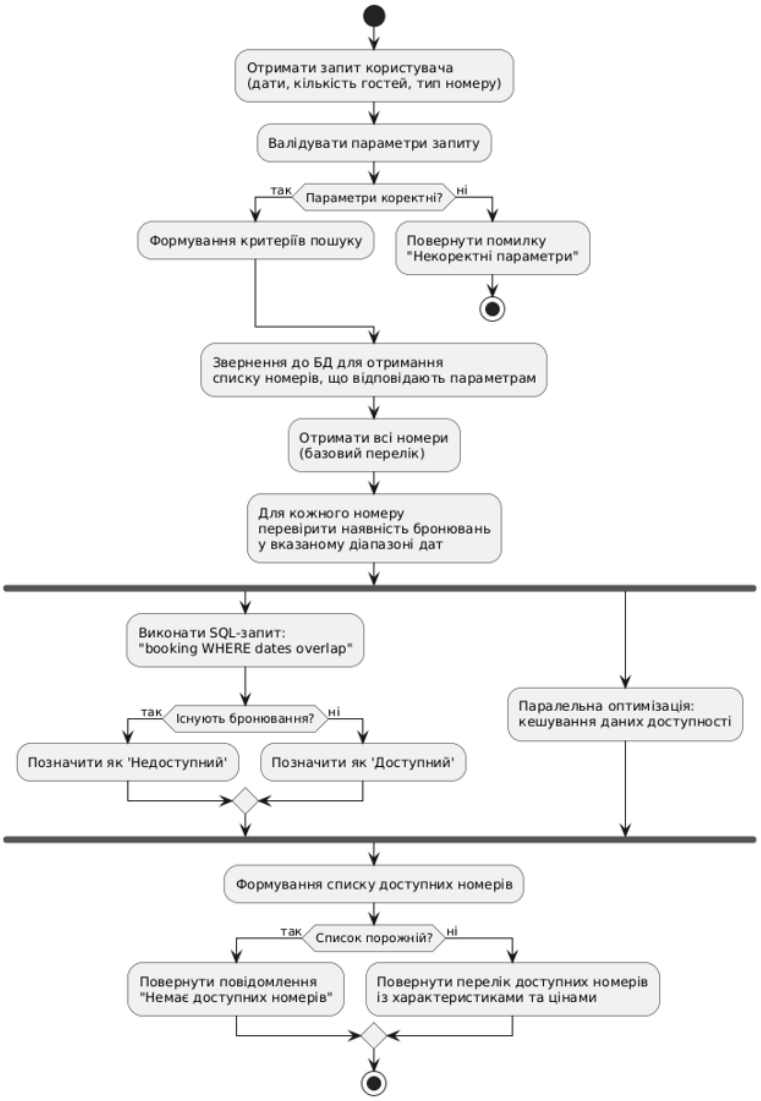


Рисунок 1.3 – UML діаграма активності механізму перевірки доступності номерів

Платіжні шлюзи, такі як LiqPay, Stripe або WayForPay, забезпечують стандартизований механізм обробки платежів у режимі реального часу та підтримують різні моделі оплати. Ці сервіси здійснюють авторизацію, списання коштів, підтвердження успішності операцій, а також формують захищені канали зворотного зв’язку з бекендом системи бронювання. Подібні платформи функціонують на принципах токенізації, що дає змогу уникнути передачі чутливих реквізитів карт у відкритому вигляді. Наукові праці з фінансових

технологій підтверджують, що токенізація значно знижує ризик компрометації платіжних даних та мінімізує наслідки потенційних кіберінцидентів [23].

Важливою складовою інтеграції є забезпечення безпеки транзакцій. Вебсервіси повинні використовувати захищені канали передачі даних, які базуються на протоколах HTTPS і TLS. Це гарантує стійкість до атак перехоплення трафіку та знижує ймовірність втручання у фінансові операції. Додаткові вимоги стосуються перевірки автентичності webhook-повідомлень від платіжних провайдерів, оскільки від цих сигналів залежить коректність оновлення статусів платежів у системі. Дослідження з галузі кібербезпеки демонструють, що багато інцидентів у фінансових вебсистемах спричинені недостатньою перевіркою цілісності комунікацій між платіжним сервісом і бекендом застосунку [24].

Захист транзакцій передбачає відповідність вимогам стандарту PCI DSS, який регламентує правила обробки та зберігання платіжної інформації. Виконання цих вимог унеможливлює зберігання CVV-кодів, повних номерів карт у незашифрованому вигляді та інших елементів, що можуть загрожувати безпеці користувачів. Стандарт PCI DSS також визначає необхідність регулярного аудиту, впровадження систем виявлення вторгнень та використання ролей доступу для обмеження взаємодії персоналу з платіжними даними. Дослідження останніх років [22]-[25] підкреслюють, що відповідність PCI DSS значно зменшує ймовірність фінансових інцидентів і підвищує загальний рівень кіберстійкості системи.

Загалом інтеграція платіжних сервісів у системи бронювання є багатокомпонентним процесом, який поєднує технічні, безпекові та організаційні аспекти. Результатом стає прозора, стандартизована та безпечна інфраструктура оплати, що формує основу довіри користувачів і забезпечує надійну роботу сервісу.

4. Принцип масштабованості та стійкості

Принцип масштабованості й стійкості систем бронювання передбачає, що навіть за пікових навантажень – наприклад у пікові туристичні сезони або під час

акцій – система має коректно виконувати свої функції без деградації продуктивності або втрати доступності. Це досягається завдяки горизонтальному масштабуванню, коли в систему додаються нові інстанції серверів, а не просто збільшується потужність одного вузла. Такий підхід дозволяє динамічно розподіляти трафік та забезпечувати високу доступність, оскільки відмова однієї або декількох інстанцій не призводить до повного виходу системи з ладу. Розподіл навантаження виступає ключовим механізмом у цьому контексті, оскільки оптимізує пропускну здатність і мінімізує затримки у відповіді для користувачів. Однією зі стратегій, що довела свою ефективність в системах з великою кількістю серверів, є так звані – Join-the-Shortest-Queue або JSQ-подібні схеми, які в умовах великих кластерів дають можливість мінімізувати час очікування без надмірного комунікаційного навантаження між диспечерами та серверними чергами [25].

Щоб гарантувати безперервність обслуговування, необхідні механізми резервного копіювання й аварійного відновлення. Регулярне створення бекапів та реплікація баз даних забезпечують ізоляцію даних від апаратних збоїв або втрат, а наявність резервних інстанцій серверів дозволяє переадресовувати трафік у випадку падіння однієї з них [26].

Принципи функціонування систем бронювання номерів охоплюють технічні, інформаційні та організаційні аспекти, що забезпечують точність операцій, безпеку даних і зручність для кінцевих користувачів. Дотримання цих принципів дозволяє створювати ефективні, масштабовані та надійні вебзастосунки для туристичної галузі.

1.3.Вимоги, які ставляться до інструментів бронювання

Інструменти бронювання у сфері розміщення та рекреації повинні забезпечувати безперервність роботи системи, точність даних і комфорт користувача, що досягається реалізацією механізмів оновлення доступності в реальному часі, синхронізації між внутрішніми системами та зовнішніми каналами продажу, адаптивного ціноутворення та безпечної обробки платежів.

Наявність реального часу оновлення доступності є критичною умовою зменшення ризику подвійних бронювань і підвищення якості управління доходами, оскільки системи, що працюють з оперативними даними, дозволяють миттєво реагувати на зміни попиту і коригувати тарифи на підставі поточної заповнюваності та прогностичних моделей. Докази ефективності підходів real-time availability представлені у дослідженнях, що розглядають оптимізацію попиту й управління бронюваннями у режимі реального часу [27].

Автоматична синхронізація між PMS, CRS і OTA-каналами зменшує невідповідності в доступності і тарифах та підвищує узгодженість даних у мультиканальному середовищі, що особливо важливо для уникнення операційних помилок під час інтенсивних періодів продажу. Техніки синхронізації, серед яких подієва архітектура та вебхуки, показали свою практичну придатність для забезпечення низької латентності оновлень і надійної доставки змін стану інвентарю [28].

Застосування динамічного ціноутворення, яке враховує сезонність, часові вікна бронювань, поведінкові патерни покупців і політики знижок, сприяє підвищенню дохідності та оптимізації заповнюваності, при цьому його ефективність підтверджена емпіричними дослідженнями у сфері управління доходами та аналітичними моделями прогнозування цін. Використання алгоритмів машинного навчання та алгоритмічних підходів дозволяє адаптувати ціни швидко і коректно на основі вхідних даних і зовнішніх факторів [29].

Інтеграція платіжних модулів повинна відповідати вимогам безпеки, оскільки безпечна обробка транзакцій є фундаментальною умовою довіри користувачів і відповідності нормативам. Використання токенізації, надійних каналів передачі даних на базі HTTPS/TLS та обробка подій через захищені вебхуки з перевіркою підписів забезпечують мінімізацію ризиків компрометації платіжних даних. Необхідність відповідності стандартам PCI DSS і регулярного аудиту безпеки підкріплюється як практичними настановами індустрії, так і дослідженнями, що аналізують ефективність заходів з кіберзахисту платіжних систем. На рисунку 1.4 показано екосистему стандартів безпеки PCI [30].

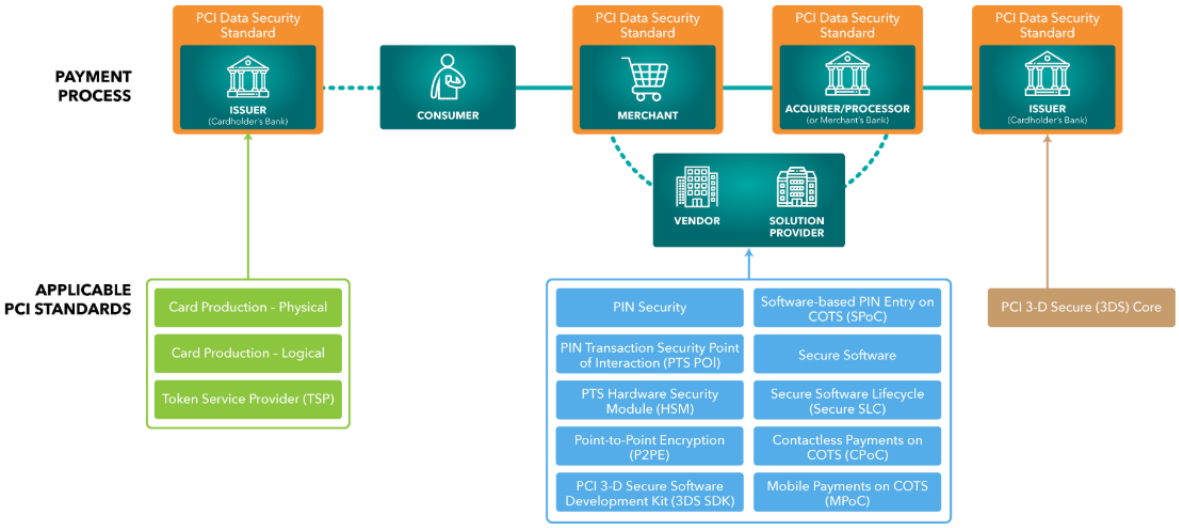


Рисунок 1.4 – Екосистема стандартів безпеки PCI

Підтримка багатомовності, багатовалютності і мобільних інтерфейсів підвищує доступність сервісу для міжнародної аудиторії і покращує користувацький досвід, що прямо впливає на конверсію і повторні бронювання. Аналітичні дослідження мобільних онлайн туристичних агенств і поведінки користувачів у мобільних сценаріях показують, що адаптовані мобільні інтерфейси значно збільшують ймовірність завершення бронювання на пристрої користувача [31], [32].

Вимоги до інструментів бронювання формуються на перетині технічних, функціональних і нормативних аспектів і втілюються через реалізацію оперативного оновлення доступності, надійної синхронізації між системами, адаптивного ціноутворення і сертифікованих платіжних рішень, що разом забезпечує сталість бізнес-процесів і високий рівень задоволеності клієнтів.

Сучасні технічні вимоги до систем бронювання передбачають забезпечення високої доступності й відмовостійкості, адже в умовах сезону або масових акцій навіть невеликий простої може призвести до значних втрат доходів. Хмарні архітектури, які підтримують горизонтальне масштабування та географічно розподілені репліки, дозволяють досягти рівнів доступності, близьких до 99,9 %, та гарантують резервне копіювання й аварійне відновлення даних при збоях [33].

Щодо інтеграцій, системи бронювання повинні реалізовувати підхід API-first, бо це дає змогу легко під’єднуватися до інших сервісів – CRM, ERP або систем керування ресурсами – і забезпечує слабку зв’язність компонентів, що підвищує гнучкість та масштабованість рішення. У хмарному контексті API-архітектура є однією з кращих практик, оскільки вона підтримує мікросервісну модель та автоматичне масштабування.

Що стосується захисту даних, сучасні дослідження пропонують використання TLS 1.3 для передавання чутливої інформації у хмарних сервісах, а також запровадження шифрування баз даних з використанням складних алгоритмів. Наприклад, нові підходи гомоморфного шифрування дозволяють проводити операції над зашифрованими даними без їх декодування, що значно підвищує конфіденційність у хмарних БД.

У частині масштабованості система має підтримувати автоматичне додавання ресурсів під час пікових навантажень, що особливо актуально для SaaS-рішень. Використання горизонтального масштабування, контейнеризації та оркестрації дозволяє динамічно реагувати на зміну трафіку й забезпечити стабільність роботи без простоїв.

Щодо безпеки, сучасні системи бронювання мусять відповідати стандартам PCI DSS для обробки платіжних даних, а також впроваджувати механізми аутентифікації через OAuth 2.0 чи JWT, щоб захищати API-доступ. Ще одним важливим аспектом є управління життєвим циклом даних: політики зберігання, архівації й видалення даних користувачів після завершення їх дії відповідно до регламентів GDPR або локальних аналогів. У дослідженнях з безпеки даних зазначено, що системи, які реалізують повний цикл даних, істотно знижують ризик витоків і підвищують довіру користувачів.

Експлуатаційні та організаційні вимоги до таких інструментів включають зручний інтерфейс для адміністраторів баз відпочинку, надання можливості навчання і техпідтримки, а також модуль аналітики з показниками заповнюваності, доходів і каналів бронювання. Хмарні SaaS-системи значно спрощують оновлення компонентів без зупинки роботи, що важливо для

підприємств гостинності з високим навантаженням та вимогою до безперервності.

Підсумовуючи, технічні вимоги до систем бронювання формуються на перетині хмарної архітектури, кібербезпеки, API-дизайну та управління даними. Їх реалізація є ключовою для забезпечення масштабованості, стійкості, безпеки і зручності як для клієнтів, так і для операторів баз відпочинку.

1.4. Огляд сучасних систем онлайн-бронювання

Сучасні онлайн-системи бронювання – це модульні, API-орієнтовані платформи, побудовані як хмарні SaaS-сервіси з багатим ринком. Вони поєднують систему бронювання, PMS, платіжні інтеграції і аналітичні / RMS-модулі. Головні вимоги – реальний час оновлення, безпека платежів і даних, масштабованість та можливість інтеграції зовнішніх сервісів; майбутнє за інтенсивним використанням AI для прогнозування попиту та персоналізації.

Нижче наведено огляд актуальних платформ бронювання для готелів, житла, закладів харчування та супутніх сервісів.

Booking.com [34] є однією з найбільш масштабних і технологічно розвинених онлайн-платформ для бронювання житла та суміжних туристичних послуг. Її створення у 1996 році в Амстердамі стало відправною точкою формування нового формату електронних сервісів бронювання, що інтегрують різні типи туристичних ресурсів у межах єдиного цифрового простору. Організаційно платформа функціонує як частина міжнародної корпорації Booking Holdings Inc., що забезпечує доступ до широкої інфраструктури цифрового туризму (Рис. 1.5).

Платформа підтримує понад сорок мов, включно з українською, та орієнтована на підвищення доступності подорожей для широкого кола споживачів. Стратегічна місія сервісу полягає у створенні цифрового середовища, що сприяє безперешкодному плануванню подорожей і взаємодії між користувачами та постачальниками послуг.

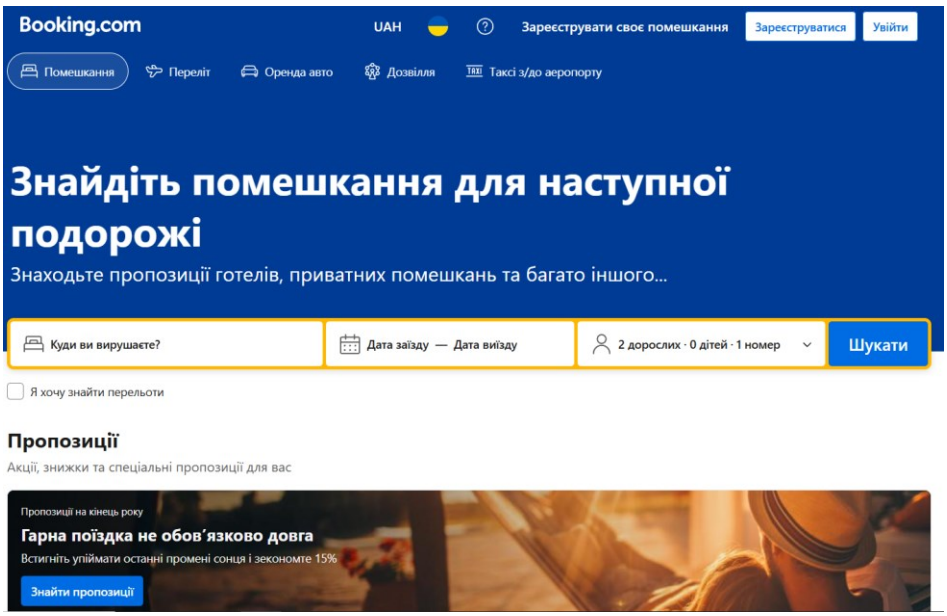


Рисунок 1.5 – Онлайн-платформ Booking.com

Система об'єднує понад двадцять вісім мільйонів об'єктів розміщення різних типів, що розташовані у більш ніж двохстах двадцяти країнах і територіях світу. Значну частину пропозицій становлять апартаменти, будинки та інші альтернативні форми житла, що відповідають тенденціям розвитку спільної економіки. Окрім послуг бронювання житла, платформа забезпечує доступ до транспортних сервісів, включаючи авіаперевезення, прокат автомобілів і різноманітні тури та активності, що представлені на багатьох регіональних ринках.

Booking.com характеризується високим рівнем зручності користування, що обумовлено інтуїтивним інтерфейсом, інтеграцією мобільних застосунків і оптимізованим процесом бронювання. Важливим компонентом екосистеми є масштабна система користувацьких відгуків, яка налічує понад триста мільйонів записів і виконує роль соціального доказу, підвищуючи прозорість і довіру до об'єктів розміщення. Платформа підтримує гнучкі варіанти скасування бронювань і надає інструменти для управління пропозиціями партнерам – власникам готелів і туристичних комплексів. Значну увагу приділено використанню аналітичних технологій, інструментів штучного інтелекту та автоматизації, що сприяють оптимізації роботи як для користувачів, так і для постачальників послуг.

Booking.com посідає провідне місце серед сервісів, що використовуються українськими туристами для організації внутрішніх і міжнародних подорожей. Українські готелі та рекреаційні об’єкти активно інтегрують свої пропозиції у платформу для підвищення видимості на міжнародному ринку та залучення іноземних відвідувачів. На внутрішньому ринку сервіс конкурує з локальними платформами, проте зберігає конкурентні переваги завдяки розгалуженій глобальній інфраструктурі, високій впізнаваності бренду та широкому діапазону інструментів підтримки.

Hotels24.ua [35] позиціонується як національний сервіс онлайн-бронювання, що агрегує пропозиції тимчасового розміщення по всій Україні та забезпечує комунікацію між гостями й власниками об’єктів. Платформа функціонує як каталог і система прийому бронювань, охоплюючи тисячі закладів різних типів, що формує широку мережу для внутрішнього туризму та регіональних потоків відпочивальників (Рис. 1.6).

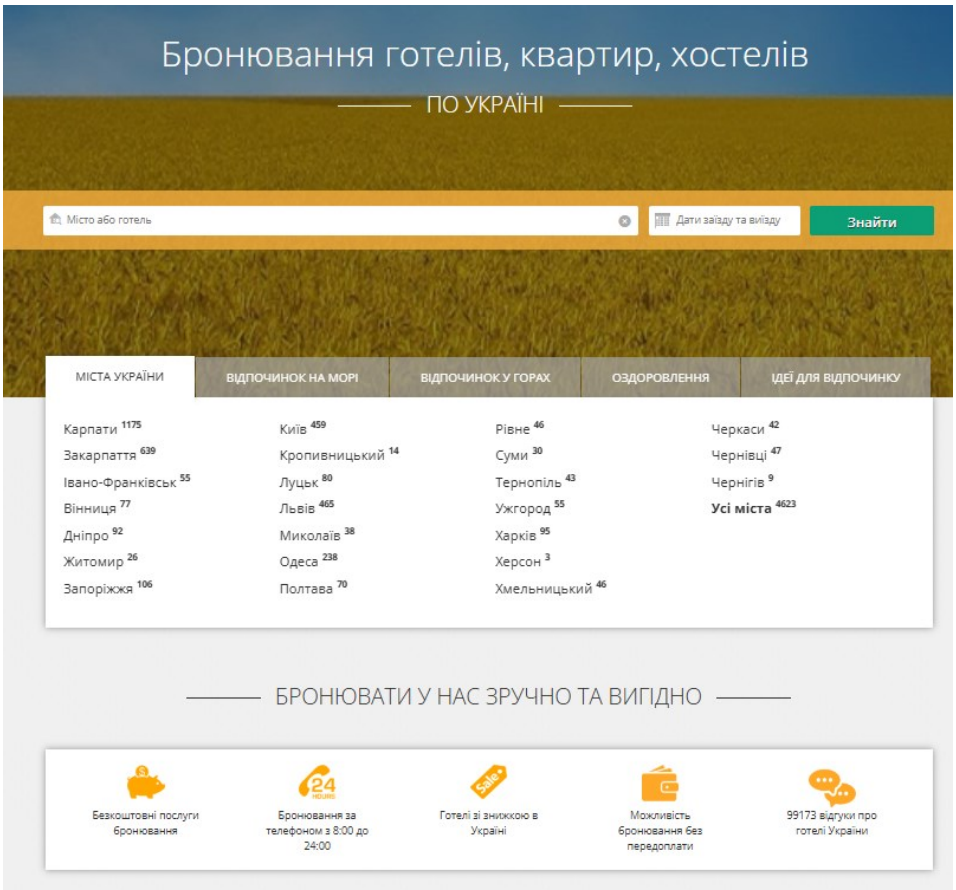


Рисунок 1.6 – Національний сервіс онлайн-бронювання Hotels24.ua

Функціональність сервісу включає пошук із фільтрами, детальні сторінки об’єктів, онлайн-бронювання та мобільний застосунок. Власникам надаються інструменти для ведення профілю, модерації інформації та управління ціновою політикою, що підтримує актуальність контенту й інтерактивність ринкової пропозиції. Партнерські та афіліатні механізми спрямовані на розширення каналів збуту й підвищення попиту.

З позицій інформаційних систем, сервіс стикається з викликами підтримання якості даних, регулярної модерації та конкуренції з міжнародними OTA. Ефективність його розвитку залежить від інвестицій у маркетинг і технічну інфраструктуру, що визначає можливості масштабування та вплив платформи на внутрішній туристичний ринок.

Doba.ua [36] є національною онлайн-платформою, що спеціалізується на сегменті короткострокової оренди житла. Сервіс функціонує як інформаційний посередник між орендодавцями та орендарями, концентруючи пропозиції приватних власників і забезпечуючи швидкий доступ користувачів до актуальних варіантів проживання у різних регіонах України (Рис. 1.7).

Каталог платформи включає квартири, апартаменти, приватні будинки та інші типи помешкань, що доступні для подобового розміщення. Географічна структура охоплює всю територію України – від великих міст до курортних зон і малих населених пунктів, що робить сервіс інструментом для внутрішнього туризму та короткострокових поїздок.

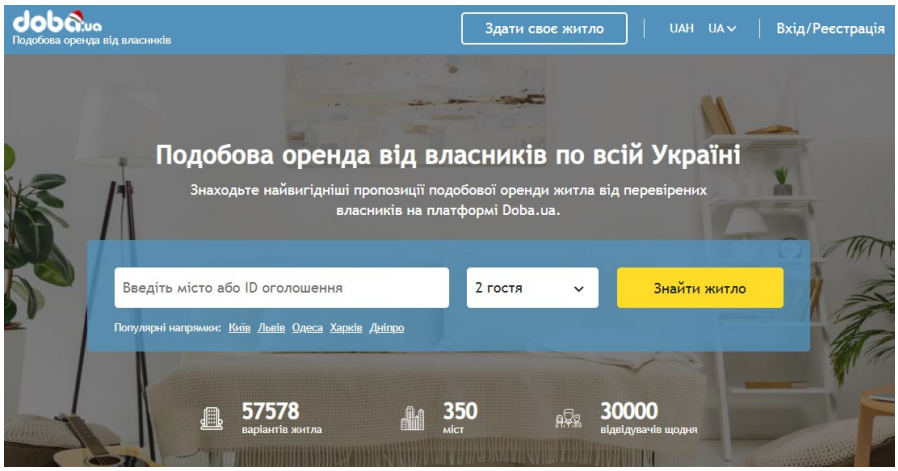


Рисунок 1.7 – Національна онлайн-платформа Doba.ua

Платформа забезпечує пошук із багатофакторними фільтрами, детальні сторінки об’єктів, перегляд фото, умови проживання, правила оренди та систему користувацьких відгуків. Характерною рисою є пряма комунікація між орендодавцем і орендарем, що замінює автоматизовані механізми бронювання. Власники самостійно керують оголошеннями, оновлюючи ціну, наявність і опис.

Doba.ua використовує інструменти модерації контенту, вебаналітики та адаптивний інтерфейс, орієнтований на мобільні пристрої. Технологічна архітектура спрямована на підтримання актуальності інформації та підвищення зручності пошуку.

GoHotels.com.ua [37] функціонує як національна онлайн-платформа, призначена для організації процесу бронювання готельних номерів без стягнення комісій із користувачів. Сервіс виконує роль посередника між готельними закладами та потенційними гостями, забезпечуючи прямий доступ до пропозицій розміщення та мінімізуючи транзакційні витрати, характерні для традиційних OTA-платформ (Рис. 1.8).

Рисунок 1.8 – Національна онлайн-платформа GoHotels.com.ua

Платформа забезпечує пошук за ключовими параметрами (регіон, тип закладу, категорія комфорту, ціновий діапазон), формує структуровані сторінки готелів із детальними характеристиками номерного фонду, умовами проживання

та мультимедійним контентом. Механізм безкоштовного бронювання ґрунтується на прямій взаємодії користувача з адміністрацією закладу, що зменшує час обробки запитів і забезпечує прозорість комунікацій.

GoHotels.com.ua використовує адаптивні вебтехнології, інструменти модерації та перевірки даних, а також механізми аналітики для оптимізації структури пропозицій. Технічна архітектура орієнтована на підвищення доступності сервісу та точності інформації, що відображається у здатності швидко оновлювати дані про наявність номерів і тарифи.

Сервіс сприяє зниженню бар'єрів для залучення нових клієнтів у малих і середніх готельних підприємств, оскільки забезпечує видимість їхніх пропозицій без додаткових витрат на комісію. Це підсилює конкурентоспроможність локальних закладів розміщення та сприяє розвитку внутрішнього туризму. Формат безкоштовної взаємодії стимулює готелі самостійно підтримувати точність і повноту інформації, що підвищує довіру користувачів.

До переваг належать прозорий механізм бронювання, зменшення фінансового навантаження на готельні заклади та доступність для широкої аудиторії. Основні обмеження пов'язані з необхідністю ручної обробки бронювань з боку готелів, відсутністю повноцінної автоматизованої системи управління номерним фондом і залежністю якості сервісу від сумлінності партнерів.

PMS-система *SERVIO HMS* позиціонується як інтегрований програмний модуль, призначений для комплексної автоматизації операційних процесів у закладах готельного типу. Система забезпечує централізоване управління підрозділами підприємства, формуючи єдиний інформаційний простір для фронт-офісу, адміністративних служб, служби прийому і розміщення, бухгалтерії та інших операційних блоків (Рис. 1.9).

Архітектура SERVIO HMS орієнтована на багатоплатформність і пропонує вебінтерфейс, що дає змогу розгортати систему як на локальному сервері готелю, так і на віддалених хмарних ресурсах. Така гнучкість інфраструктури забезпечує

масштабованість, оперативне оновлення даних і доступ до інтерфейсу з різних робочих станцій.

Функціональний спектр модуля охоплює управління номерним фондом, облік бронювань, ведення клієнтських профілів, контроль завантаженості, взаємодію між службами та формування аналітичних звітів. Застосування системи сприяє підвищенню ефективності бізнес-процесів, мінімізації помилок персоналу та підвищенню якості сервісу в готельних підприємствах різного типу.

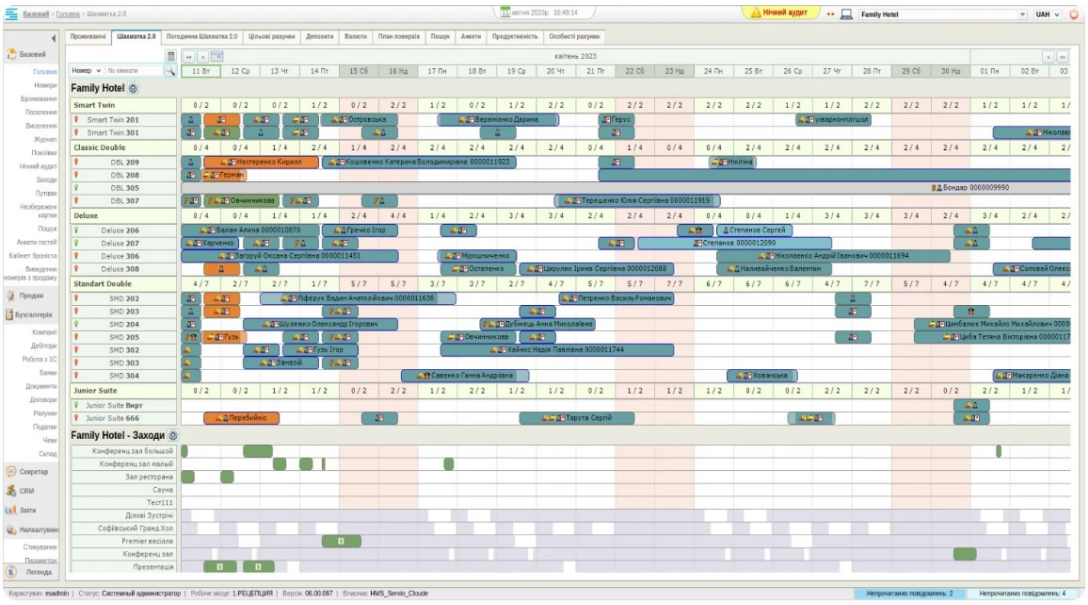


Рисунок 1.9 – PMS-система SERVIO HMS

Порівняльний аналіз розглянутих сервісів і PMS-систем показує їхнє призначення, цільову аудиторію та ключові переваги (Таблиця 1.2).

Таблиця 1.2

Порівняльна таблиця систем бронювання

Назва	Основна функція	Цільова аудиторія	Ключові переваги
Booking.com	Бронювання житла та туристичних послуг глобально	Міжнародні туристи, власники готелів	Глобальне охоплення, 28 млн+ об’єктів, багатомовна підтримка, аналітика та AI

Hotels24.ua	Національна платформа бронювання	Українські туристи, локальні готелі	Фокус на внутрішньому туризмі, регіональний пошук, партнерські кабінети
Doba.ua	Подобова оренда житла від приватних власників	Внутрішні туристи, приватні орендодавці	Peer-to-peer модель, гнучкі ціни, прямий контакт з власником
GoHotels.com.ua	Безкоштовне бронювання готелів	Українські туристи, малі та середні готелі	Відсутність комісії, прямий контакт із готелями, прозорість
SERVIO HMS	Комплексна PMS-система для готелів	Середній та великий готельний бізнес	Централізоване управління всіма підрозділами, веб-інтерфейс, локальна або хмарна інсталяція
EasyMS PMS	Легка хмарна PMS-система	Малі та середні готелі, хостели	Простота використання, онлайн-бронювання через сайт, соцмережі та месенджери, мобільний доступ

Міжнародна платформа Booking.com забезпечує глобальне охоплення та багатомовну підтримку, тоді як Hotels24.ua орієнтована на внутрішній ринок України з регіональним пошуком і партнерськими кабінетами для готелів. Doba.ua спеціалізується на подобовій оренді житла від приватних власників із прямим контактом та гнучкими цінами. GoHotels.com.ua пропонує безкоштовне бронювання готелів для користувачів і малих готельних підприємств. Системи управління готелями SERVIO HMS і EasyMS PMS забезпечують автоматизацію бізнес-процесів, причому перша орієнтована на середній та великий бізнес із комплексною інтеграцією підрозділів, а друга – на малі та середні заклади з онлайн-бронюванням через сайт, соціальні мережі та мобільні пристрої.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ АВТОМАТИЗАЦІЇ ПРОЦЕСУ БРОНЮВАННЯ НОМЕРІВ БАЗИ ВІДПОЧИНКУ.

2.1. Постановка задачі, призначення та вимоги до розробки

У сучасних умовах цифрової трансформації сфера гостинності потребує впровадження ефективних інструментів управління ресурсами. Бази відпочинку, зокрема база «Гарт», часто стикаються з низкою організаційних та технічних проблем, пов’язаних із використанням застарілих або ручних систем бронювання, що ґрунтуються на телефонних дзвінках чи електронних таблицях. Такий підхід спричиняє виникнення конфліктів у даних, ризик подвійного бронювання, труднощі з оперативним коригуванням цінової політики, значні витрати часу на підтвердження замовлень та інформування клієнтів, а також ускладнює аналіз завантаженості номерного фонду через відсутність централізованих аналітичних інструментів.

Проблематика дослідження полягає у необхідності створення системи, що забезпечить автоматизацію бізнес-процесів бази відпочинку, підвищення ефективності управління ресурсами та прибутковості. Розробка орієнтована на впровадження алгоритмів управління дохідністю, які дозволяють оптимізувати використання ресурсів із фіксованою пропозицією відповідно до попиту, сезонності та інших динамічних чинників.

Метою розроблюваного вебзастосунку є створення інтегрованої, надійної та інтелектуальної системи управління базою відпочинку «Гарт». Система повинна забезпечувати автоматизоване бронювання номерів у реальному часі, підтримку гнучкої цінової політики, централізовану аналітику та зручний інтерфейс взаємодії між клієнтами й адміністрацією.

Для клієнтів застосунок має гарантувати швидкий пошук і безпечне бронювання доступних номерів, а для адміністрації – ефективний контроль за ресурсами, візуалізацію стану завантаженості та можливість оперативного коригування тарифів відповідно до ринкових умов.

Виходячи з поставленої проблеми, визначено основні напрями роботи.

1. Розробити структуру реляційної бази даних, яка забезпечуватиме зберігання інформації про номери, користувачів і бронювання.

2. Створити відмовостійкий прикладний програмний інтерфейс REST API на базі FastAPI, що гарантуватиме швидкий доступ до даних і підтримуватиме атомарність транзакцій під час бронювання.

3. Реалізувати бізнес-логіку, яка забезпечує перевірку доступності номерів і запобігає випадкам подвійного бронювання в системі.

4. Розробити захищений адміністративний вебінтерфейс на платформі Next.js, який дозволятиме виконання CRUD-операцій і надаватиме засоби для візуалізації аналітичних даних.

5. Створити модуль динамічного ціноутворення, що автоматично змінюватиме базову вартість номерів залежно від сезонності та дня тижня.

6. Розробити сервіс автоматизованої комунікації з клієнтами, який надсилатиме електронні повідомлення після успішного бронювання.

7. Провести дослідження та впровадити механізми блокування транзакцій на рівні ORM, які забезпечуватимуть цілісність даних і стабільність системи за умов високого навантаження, запобігаючи виникненню конфліктів доступу.

Вимоги до розробки показані у таблицях 2.1 та 2.2.

Таблиця 2.1

Функціональні вимоги до розробки

Назва вимоги	Опис реалізації
Управління ресурсами	Система має забезпечувати повний життєвий цикл управління даними про номерний фонд. Адміністратор повинен мати можливість додавати, переглядати, редагувати та вилучати відомості про номери через захищений інтерфейс.
Атомарне бронювання	Вебзастосунок має приймати нові заявки на бронювання лише за умови відсутності перетину дат із раніше підтвердженими записами. Така логіка повинна

	забезпечувати цілісність даних і виключати можливість дублювання бронювань.
Фільтрація результатів пошуку	Інтерфейс користувача повинен надавати можливість фільтрації номерів за датами заїзду й виїзду, а також за мінімальною місткістю. Це має сприяти зручності пошуку та підвищенню ефективності взаємодії клієнта із системою.
Автентифікація користувачів	Для виконання адміністративних операцій має бути реалізований механізм автентифікації на основі JWT-токенів. Його впровадження повинно гарантувати безпечний доступ до службових функцій та захист від несанкціонованих дій.
Візуалізація зайнятості номерного фонду	Адміністративна панель повинна містити засоби для графічного відображення поточного стану завантаженості номерів. Така функція має забезпечувати швидку оцінку доступності ресурсів та оперативне управління бронюваннями.
Система знижок	У системі має бути реалізований механізм автоматичного надання фіксованої знижки (25%) користувачам, які вводять чинний код працівника. Цей функціонал повинен забезпечувати прозорість і автоматизацію програми лояльності.

Таблиця 2.2

Вимоги до інтерфейсу користувача

Назва вимоги	Опис реалізації
Адаптивність інтерфейсу	Інтерфейс вебзастосунку має бути реалізований із дотриманням принципу *Mobile First* та повною адаптивністю до різних типів пристроїв (комп’ютери, планшети, смартфони).
Зручна навігація	Структура інтерфейсу повинна передбачати наявність чіткої та інтуїтивно зрозумілої навігаційної панелі, яка забезпечує швидкий перехід до основних розділів системи, зокрема «Номери», «Бронювання» та «Вхід для адміністратора».
Зручність введення даних	У формі бронювання мають бути реалізовані елементи інтерфейсу, що спрощують процес заповнення даних, зокрема випадаючі списки для вибору номера та автоматична валідація введених полів

2.2. Вибір моделі розробки програмного засобу

Для розроблення вебзастосунку системи бронювання обрано гнучку модель програмної інженерії – Agile, що характеризується високою адаптивністю до змін вимог і спрямованістю на досягнення максимальної якості програмного продукту. На відміну від класичних каскадних підходів, дана методологія ґрунтується на ітеративно-інкрементному процесі, у межах якого проєкт поділяється на короткі цикли (спринти). Після завершення кожного спринту формується проміжна версія продукту, яка підлягає перевірці та оцінюванню з боку зацікавлених осіб (Рис. 2.1).

Рисунок 2.1 – Процес розробки за методологією Agile

Вибір моделі розробки програмного забезпечення базується на доцільності використання гнучкої методології Agile, яка забезпечує адаптивність процесу створення вебзастосунку та ефективну взаємодію між етапами його реалізації.

Застосування підходу Agile зумовлене низкою чинників, що підвищують ефективність розробки. По-перше, на початкових етапах проєктування неможливо повністю передбачити всі деталі реалізації, зокрема особливості механізму динамічного ціноутворення або специфіку адміністративного інтерфейсу. Гнучкість методології Agile дає змогу уточнювати вимоги поступово, у межах кожної ітерації.

По-друге, ітераційна структура процесу сприяє отриманню оперативного зворотного зв’язку. Демонстрація проміжних результатів – мінімально життєздатного продукту – після кожного спринту дозволяє науковому керівнику

або потенційному користувачу оцінювати прогрес і вчасно коригувати технічні рішення.

По-третє, поділ проєкту на невеликі логічно завершені завдання створює умови для ефективного управління ризиками. Такий підхід дає можливість своєчасно виявляти й усувати архітектурні недоліки, наприклад, циклічні залежності у програмному коді чи нераціональні структури даних.

Крім того, поетапна інтеграція компонентів сприяє підвищенню надійності системи. Незалежна розробка та тестування модулів – бекенду й фронтенду – забезпечує можливість їхньої подальшої безперервної інтеграції в єдину функціональну систему без порушення цілісності проєкту.

Розробка даного продукту була поділена на 5 основних спринтів (Табл. 2.3), кожен з яких мав чітку мету та завершувався робочим інкрементом продукту.

Таблиця 2.3

Етапи розробки за обраною моделлю

Спринт	Назва	Ключові задачі	Результат (Інкремент)
Спринт 1	Foundation (Інфраструктура)	Налаштування Docker для PostgreSQL. Ініціалізація FastAPI (Backend) та Next.js (Frontend). Базова структура проєкту.	Розгорнуте локальне середовище розробки.
Спринт 2	Backend Core (Ядро API)	Проєктування БД (SQLAlchemy). Реалізація CRUD для номерів. Впровадження JWT-автентифікації. Бізнес-логіка is_room_available.	Робочий REST API з документацією Swagger UI.
Спринт 3	Frontend UI (Інтерфейс)	Створення адаптивного макету. Реалізація каталогу номерів, форми бронювання та панелі адміністратора.	Веб-інтерфейс, інтегрований з API.
Спринт 4	Novelty & Automation (Новизна)	Розробка алгоритму динамічного ціноутворення. Інтеграція email-сповіщень	Інтелектуальна система з автоматизацією.

		(SMTP). Розробка інтерактивної сітки зайнятості.	
Спринт 5	Testing & Deployment (Якість)	Написання модульних тестів (pytest) та тестів інтерфейсу (Jest). Налаштування Docker-контейнерів для продакшну. Опис CI/CD.	Готовий до розгортання, протестований продукт.

Методологія Agile має низку переваг, що підтверджують її доцільність для даного проєкту. Вона забезпечує високу гнучкість у реалізації змін, наприклад, можливість додавання нової функції знижки для викладачів на пізньому етапі розробки. Процес відрізняється прозорістю, оскільки після кожного спринту розробник отримує чітке уявлення про поточний стан проєкту. Постійне тестування, як ручне, так і автоматизоване, сприяє зниженню кількості критичних помилок у фінальному релізі.

Водночас використання Agile має певні обмеження. Одним із ризиків є розширення обсягу проєкту через постійне додавання нових функцій, що потребує чіткого визначення пріоритетів. Крім того, у разі індивідуальної розробки методологія вимагає високого рівня самодисципліни, оскільки розробник поєднує ролі Scrum Master, Product Owner і члена команди розробки.

2.3. Опис проєкту програмного засобу

Розроблена система управління базою відпочинку «Гарт» надає такі основні функціональні можливості:

Для клієнта:

1. Перегляд каталогу доступних номерів з детальною інформацією (назва, опис, місткість, базова ціна).
2. Фільтрація номерів за бажаними датами заїзду/виїзду та кількістю осіб.
3. Перегляд актуальної ціни на вибрані дати (з урахуванням динамічного ціноутворення).
4. Оформлення бронювання обраного номера.

5. Отримання автоматичного підтвердження бронювання на електронну пошту.

Для адміністратора:

1. Автентифікація в системі за допомогою логіну та пароля.
2. Перегляд дашборду зі статистикою (загальна кількість бронювань, орієнтовний дохід).
3. Візуалізація зайнятості номерів у вигляді інтерактивної сітки (на поточний місяць).
4. Додавання нових номерів до бази даних.
5. Можливість ручного створення бронювання через сітку зайнятості.

Діаграма варіантів використання (Use Case Diagram)

Діаграма використання відображає функціональні вимоги до системи та демонструє взаємодію між користувачами (акторами) і основними процесами (прецедентами). Вона дозволяє візуалізувати, які функції виконує система у відповідь на дії користувачів різних типів, а також визначити межі її функціональних можливостей (Рис. 2.2.).

У моделі розглядаються два типи акторів – гість та адміністратор. Гість є неавторизованим користувачем вебсайту й потенційним клієнтом бази відпочинку. Адміністратор, своєю чергою, є авторизованим співробітником, який має повноваження щодо управління системою та виконання адміністративних операцій.

Для гостя передбачено кілька основних сценаріїв взаємодії з системою. Базовим є перегляд номерів, доступний усім відвідувачам сайту. Додатково користувач може здійснювати фільтрацію за датою, що дозволяє уточнити пошук вільних номерів. Цей процес розширює сценарій перегляду номерів. Основною цільовою дією гостя є бронювання номера, яке включає послідовність підпроцесів: перевірку доступності номерів як системну операцію та введення контактних даних для підтвердження бронювання.

Діяльність адміністратора охоплює ширший спектр операцій. Першим кроком є авторизація у системі, що забезпечує доступ до адміністративних

функцій. Після входу адміністратор отримує доступ до дашборду – головної сторінки, де відображається узагальнена статистична інформація про стан бази відпочинку. Крім того, передбачено можливість перегляду сітки зайнятості, яка візуалізує статус номерів у вигляді календаря. Адміністратор також може додавати нові номери до бази даних та створювати бронювання вручну для клієнтів через інтерфейс сітки зайнятості. Останній прецедент передбачає включення перевірки доступності номерів як обов'язкового підпроцесу.

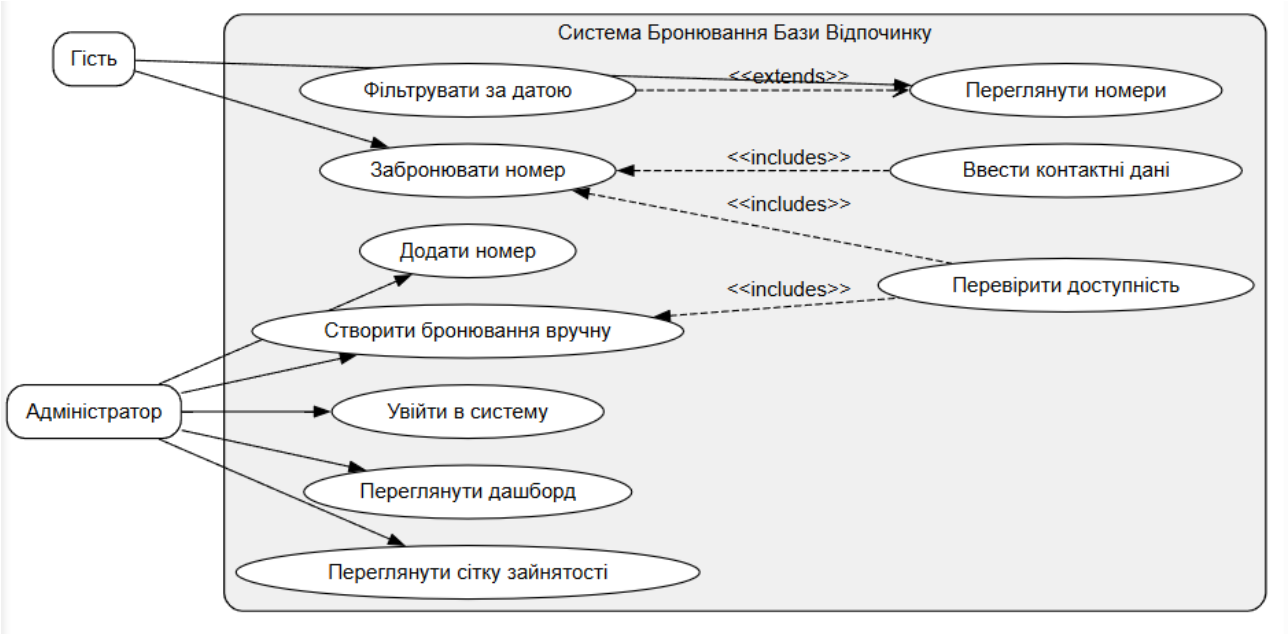


Рисунок 2.2 – Діаграма варіантів використання (Use Case Diagram)

Таким чином, діаграма використання відображає логічну структуру взаємодії між користувачами та системою, окреслює межі її функціональності та забезпечує розуміння основних сценаріїв роботи з вебресурсом як з позиції клієнта, так і адміністратора.

Діаграма класів (Class Diagram)

Діаграма класів відображає статичну структуру програмної системи та демонструє взаємозв'язки між її основними сутностями. У моделі визначено абстрактний клас Person, який узагальнює спільні характеристики користувачів системи, зокрема атрибут *email*. Від нього успадковуються класи Admin та Guest, що відображають різні ролі користувачів (Рис. 2.3).

Клас Admin представляє користувача з розширеними правами доступу. До його складу входять атрибути *hashed_password* та *is_admin*, що забезпечують автентифікацію та контроль прав доступу. Основні методи цього класу реалізують функції адміністрування, такі як додавання, редагування та видалення номерів, перегляд дашборду, аналіз списку бронювань та створення бронювань вручну.

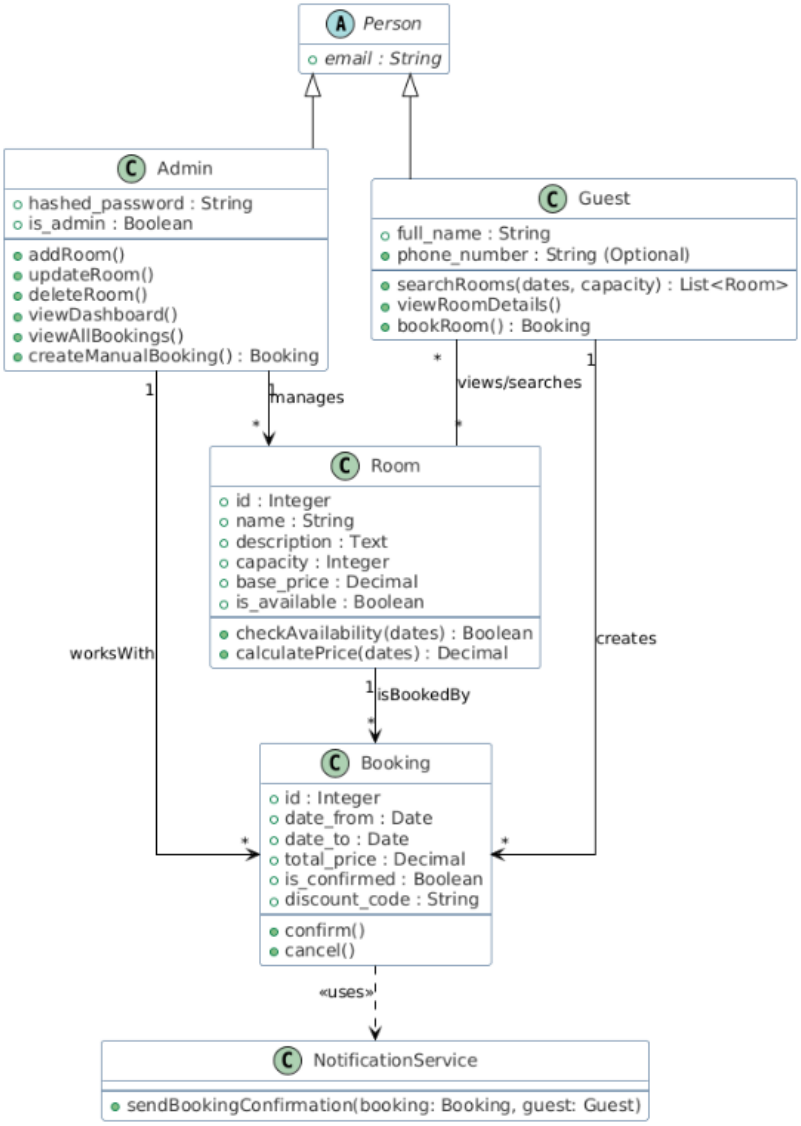


Рисунок 2.3 – Діаграма класів вебзастосунку

Клас Guest описує клієнта, який здійснює пошук і бронювання номерів. Він має атрибути *full_name* та *phone_number*. Методи класу забезпечують можливість пошуку номерів за заданими параметрами, перегляду детальної інформації про них та оформлення бронювання.

Клас Room моделює об’єкт проживання у базі відпочинку. Він містить атрибути *id*, *name*, *description*, *capacity*, *base_price* та *is_available*, які характеризують номер і його стан. Методи класу відповідають за перевірку доступності та розрахунок вартості проживання у визначений період.

Клас Booking відображає факт бронювання номера. До його складу входять атрибути *id*, *date_from*, *date_to*, *total_price*, *is_confirmed* та *discount_code*. Основні методи реалізують підтвердження або скасування бронювання.

Для автоматизації інформування користувачів використовується сервісний клас NotificationService, який відповідає за надсилання повідомлень клієнтам. Його метод забезпечує відправлення підтвердження про бронювання гостю після успішної операції.

У системі встановлено кілька типів зв’язків між класами. Зокрема, клас Admin успадковує властивості абстрактного класу Person та має асоціації з класами Room і Booking, що відображає його функції з управління номерним фондом і замовленнями. Клас Guest також успадковує Person і пов’язаний із класом Room через відношення перегляду та пошуку номерів, а з класом Booking – через зв’язок типу «один до багатьох», який фіксує історію його бронювань. Крім того, між класами Room і Booking існує зв’язок, що визначає можливість бронювання одного номера кілька разів у різні періоди. Клас Booking має залежність від NotificationService, оскільки використовує його функціонал для надсилання підтверджень клієнтам.

Запропонована структура забезпечує логічну узгодженість моделі, чітке розмежування обов’язків між компонентами системи та підтримку принципів повторного використання коду. Вона сприяє підвищенню гнучкості системи, спрощує її подальшу модифікацію та розширення функціональності.

Структура бази даних

База даних реалізована на СУБД PostgreSQL. Вона складається з трьох основних таблиць.

Таблиця `rooms` (Номери) зберігає інформацію про номерний фонд бази (Таблиця 2.4; Рис. 2.4.).

Таблиця 2.4.

Таблиця rooms (Номери)

Поле	Тип	Опис
id	SERIAL (PK)	Унікальний ідентифікатор номера.
name	VARCHAR	Назва номера (наприклад, "Люкс 1").
description	TEXT	Детальний опис зручностей.
capacity	INTEGER	Максимальна кількість гостей.
base_price	FLOAT	Базова вартість за добу (без урахування сезонності).
is_available	BOOLEAN	Прапорець доступності номера для бронювання.

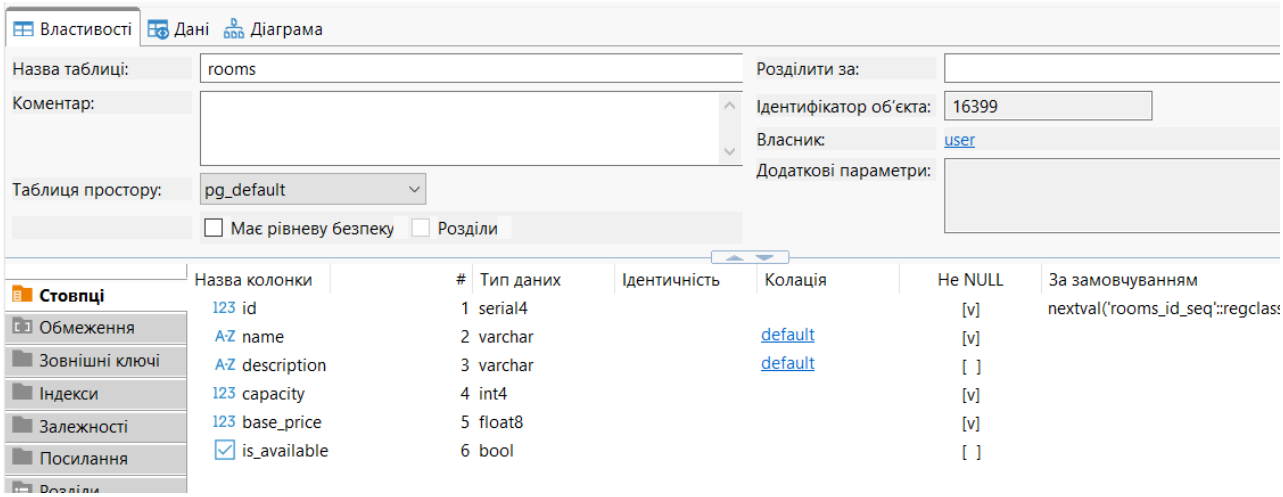


Рисунок 2.4 – Таблиця rooms (Номери)

Таблиця `bookings` (Бронювання) зберігає інформацію про всі замовлення (Таблиця 2.5; Рис. 2.5.).

Таблиця 2.5.

Таблиця `bookings` (Бронювання)

Поле	Тип	Опис
id	SERIAL (PK)	Унікальний ідентифікатор бронювання.
room_id	INTEGER (FK)	Посилання на `rooms.id`.
client_name	VARCHAR	ПІБ клієнта.
client_email	VARCHAR	Email для сповіщень.
date_from	DATE	Дата заїзду.
date_to	DATE	Дата виїзду.
total_price	FLOAT	Розрахована вартість за весь період.
discount_code	VARCHAR	Код використаної знижки.

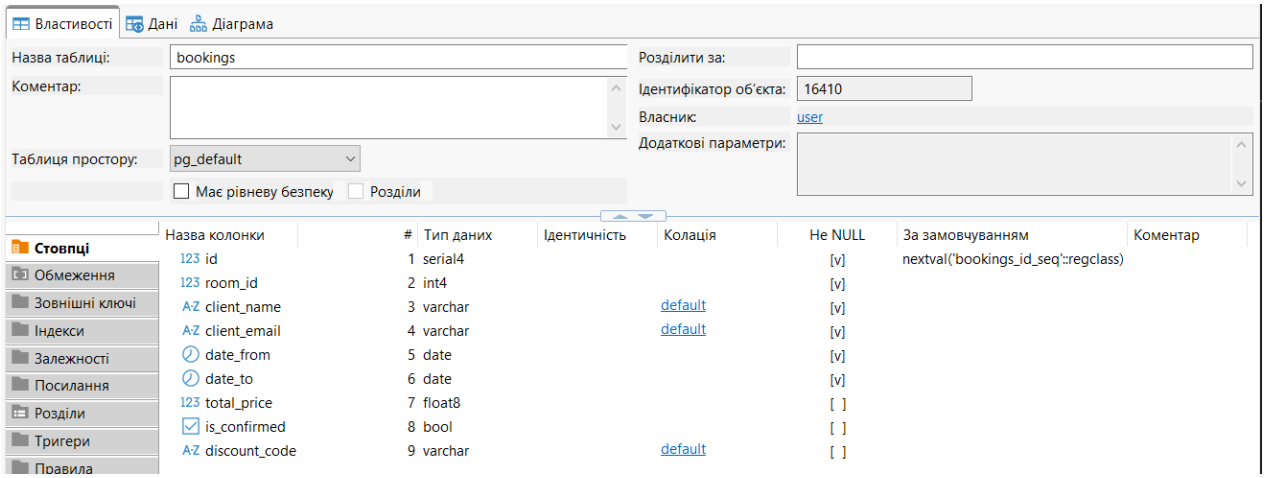


Рисунок 2.5 – Таблиця `bookings` (Бронювання)

Таблиця `users` (Користувачі) зберігає дані адміністраторів для доступу до панелі управління (Таблиця 2.6; Рис. 2.6.).

Таблиця 2.6.

Таблиця `users` (Користувачі)

Поле	Тип	Опис
id	SERIAL (PK)	Унікальний ідентифікатор користувача.
email	VARCHAR (UNIQUE)	Логін (email) адміністратора.
hashed_password	VARCHAR	Хеш пароля (bcrypt).
is_admin	BOOLEAN	Прапорець прав адміністратора.

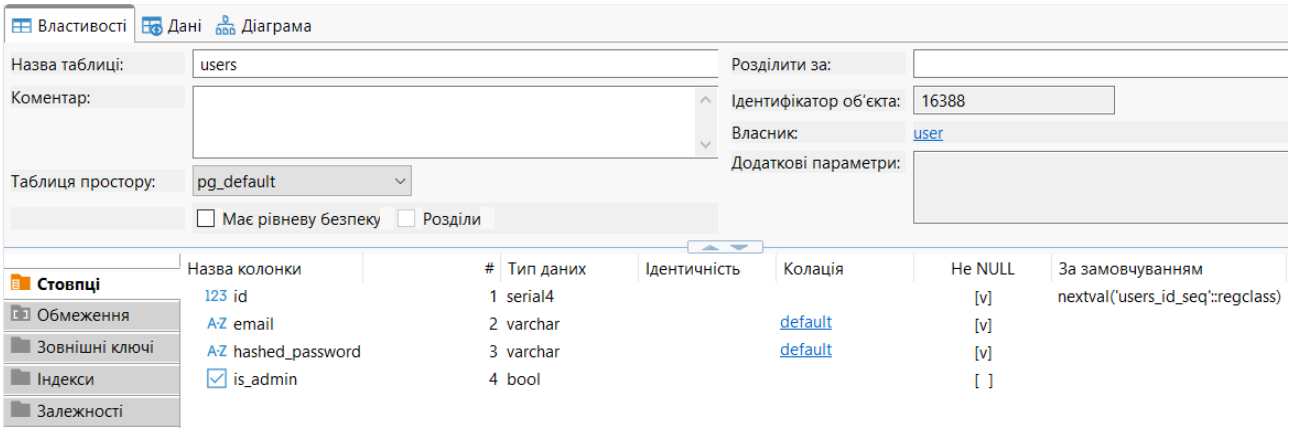


Рисунок 2.6 – Таблиця `users` (Користувачі)

Архітектурна модель (клієнт–сервер)

Розроблена система ґрунтується на класичній триланковій архітектурі, яка забезпечує модульність, масштабованість та незалежність компонентів. Кожен

рівень системи виконує власні функції та взаємодіє лише із суміжними рівнями, що сприяє підвищенню стабільності й гнучкості програмного рішення.

Перший рівень – презентаційний (Presentation Tier або Frontend) – відповідає за взаємодію користувача з системою та візуалізацію даних. Його реалізовано за допомогою фреймворку Next.js, що базується на технології React. Цей рівень включає клієнтську частину, яка функціонує безпосередньо у браузері користувача як односторінковий застосунок (SPA), забезпечуючи динамічність інтерфейсу без необхідності перезавантаження сторінки. Окрім цього, використовується серверний рендеринг (SSR) у середовищі Node.js, що забезпечує формування HTML-коду на сервері перед його передачею клієнту. Такий підхід підвищує швидкість початкового завантаження сторінок і покращує показники пошукової оптимізації. Презентаційний рівень забезпечує відображення інтерфейсу, збір і первинне опрацювання даних від користувача, надсилання запитів до прикладного рівня через AJAX або fetch API, а також оброблення повідомлень про помилки.

Другий рівень – рівень бізнес-логіки (Application Tier або Backend API) – є ядром системи, яке відповідає за опрацювання запитів, реалізацію бізнес-правил і управління даними. Його реалізовано з використанням фреймворку FastAPI мовою Python, який функціонує під управлінням ASGI-сервера Uvicorn. У межах цього рівня реалізовано RESTful API, що забезпечує комунікацію з клієнтською частиною. До основних функцій належать автентифікація та авторизація користувачів через перевірку JWT-токенів, виконання бізнес-правил, таких як перевірка доступності номерів і розрахунок динамічної вартості проживання, а також застосування знижок відповідно до заданих умов. Для перевірки вхідних даних використовуються Pydantic-схеми, що гарантують відповідність отриманих JSON-запитів вимогам структури даних.

Третій рівень – рівень даних (Data Tier або Database) – забезпечує надійне зберігання та підтримання цілісності інформації. Як основну систему керування базами даних застосовано PostgreSQL, яка забезпечує збереження реляційних даних у таблицях користувачів, номерів та бронювань. Цей рівень підтримує

ACID-транзакції, що гарантують атомарність операцій, зокрема під час створення бронювань, а також виконує складні SQL-запити для фільтрації та аналітики даних.

Взаємодія між рівнями здійснюється за допомогою протоколів HTTP або HTTPS у форматі даних JSON. Презентаційний рівень надсилає запити до прикладного рівня через RESTful API, використовуючи стандартні методи GET для отримання даних (наприклад, списку номерів або бронювань) та POST для створення нових ресурсів, таких як замовлення чи запити на авторизацію. Для захищеного доступу до ресурсів клієнт додає до кожного запиту заголовок авторизації виду *Authorization Bearer access_token*.

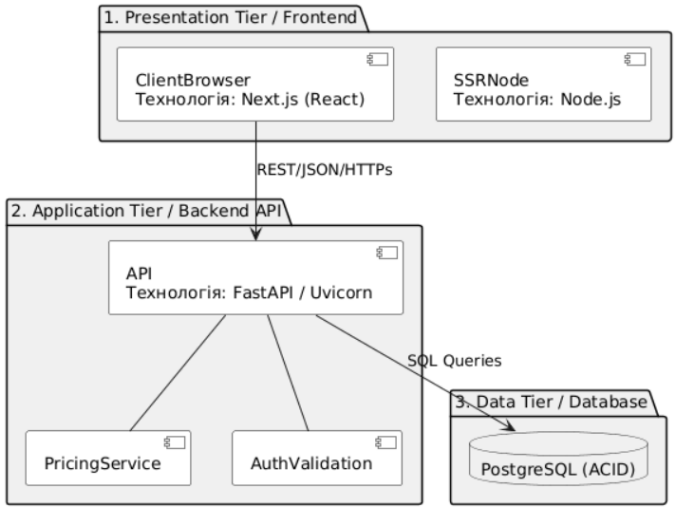


Рисунок 2.7 – Архітектурна модель (Клієнт-Сервер)

Триланкова архітектура забезпечує можливість незалежного масштабування кожного рівня системи. Зокрема, при зростанні навантаження може бути збільшено кількість екземплярів API-сервісу без впливу на інші компоненти. Такий підхід спрощує технічну підтримку, сприяє підвищенню продуктивності та забезпечує ефективне розмежування логічних частин системи відповідно до принципів сучасного програмного проєктування.

2.4. Обґрунтування вибору інструментальних засобів

Для розроблення серверної частини вебзастосунку було обрано мову програмування Python [38]. Такий вибір є обґрунтованим з огляду на низку ключових факторів, що мають вирішальне значення для досягнення цілей

дослідження, зокрема у частині реалізації елементів новизни, пов’язаних із динамічним ціноутворенням.

Python посідає провідні позиції у сфері аналізу даних і машинного навчання [39]. Наявність розвиненої екосистеми бібліотек створює сприятливе середовище для впровадження інтелектуальних функцій. Зокрема, Pandas забезпечує ефективне опрацювання та аналіз структурованих даних, NumPy використовується для високопродуктивних математичних обчислень, а Scikit-learn – для побудови моделей машинного навчання.

Завдяки цим можливостям Python дозволяє реалізувати не лише поточну спрощену модель динамічного ціноутворення, засновану на правилах, але й забезпечує перспективу інтеграції повноцінних моделей машинного навчання для прогнозування попиту без необхідності зміни технологічного стеку.

Python відомий зрозумілим та компактним синтаксисом, близьким до природної мови. Це дає змогу зосередитися на вирішенні прикладних завдань, таких як реалізація складної логіки перевірки доступності номерів. Такий підхід підвищує ефективність розробки. Крім того, зрозумілість та структурованість коду спрощують його тестування та підтримку.

Сучасні версії Python мають вбудовану підтримку асинхронного програмування через бібліотеку asyncio, що сприяє розробленню високопродуктивних вебзастосунків. Це стало підґрунтям для появи нового покоління фреймворків, серед яких FastAPI, використаний у цьому проєкті, вирізняється швидкодією та зручністю. Асинхронна архітектура дає змогу опрацьовувати велику кількість одночасних запитів без блокування основного потоку виконання, що особливо актуально під час пікових навантажень, наприклад, у період масового бронювання номерів.

Застосування мови Python є стратегічним рішенням, яке забезпечує високу ефективність розробки, гнучкість архітектури та потенціал для подальшого інтелектуального розвитку системи, зокрема шляхом впровадження алгоритмів машинного навчання та прогнозної аналітики.

Під час розроблення серверної частини вебзастосунку було проведено порівняльний аналіз сучасних Python-фреймворків, зокрема Django та Flask. За результатами аналізу для реалізації системи було обрано FastAPI – сучасний високопродуктивний вебфреймворк, призначений для створення прикладних програмних інтерфейсів (API) [40].

Вибір цього фреймворку зумовлений низкою ключових переваг, що є критично важливими для побудови системи бронювання.

FastAPI базується на стандарті ASGI (Asynchronous Server Gateway Interface) і використовує фреймворк Starlette для вебмаршрутизації та сервер Uvicorn для опрацювання запитів [41].

Завдяки цій архітектурі забезпечується повноцінне асинхронне опрацювання запитів із використанням механізмів *async/await*. Асинхронна модель дає змогу серверу опрацьовувати численні конкурентні запити без блокування ресурсів, що значно підвищує пропускну здатність.

Однією з важливих переваг FastAPI є автоматичне створення документації API відповідно до стандартів OpenAPI та JSON Schema. Завдяки цій функції розробник отримує інтерактивний вебінтерфейс Swagger UI одразу після визначення першого ендпоінта. Це забезпечує можливість тестування API безпосередньо у браузері без потреби використовувати сторонні інструменти. Подібна інтеграція прискорює процес розроблення, налагодження коду та інтеграції бекенд-частини з фронтендом.

FastAPI тісно інтегрований із бібліотекою Pydantic, яка застосовує стандартні анотації типів Python для автоматичної перевірки вхідних даних. У системі бронювання важливою вимогою є коректність інформації, зокрема перевірка логічної послідовності дат, недопущення від’ємних значень цін або помилкових форматів електронних адрес. Бібліотека Pydantic автоматично здійснює перевірку отриманих JSON-запитів згідно із заздалегідь визначеними схемами ще до передачі даних у бізнес-логіку. Це мінімізує кількість коду, необхідного для ручної перевірки, та забезпечує цілісність інформації на рівні API, що зменшує ризик виникнення логічних і структурних помилок.

Таким чином, використання FastAPI є обґрунтованим технічним рішенням, яке забезпечує високу продуктивність, надійну перевірку даних, гнучкість у масштабуванні та зручність у процесі розроблення і тестування програмного забезпечення.

У процесі розроблення вебзастосунку в якості основної системи управління базами даних було обрано об’єктно-реляційну систему PostgreSQL. Ця система є потужною відкритою (Open Source) об’єктно-реляційною СУБД, що вирізняється високою надійністю, стійкістю до відмов та суворим дотриманням міжнародних стандартів. PostgreSQL часто характеризують як одну з найрозвинутіших відкритих реляційних систем управління базами даних у світі [42].

Основним аргументом на користь вибору PostgreSQL є її повна сумісність із принципами ACID (Atomicity, Consistency, Isolation, Durability). Така сумісність забезпечує надійність опрацювання транзакцій, що є критично важливим для систем, у яких цілісність даних має пріоритетне значення, наприклад у фінансових або бронювальних застосунках. У подібних системах транзакція або повністю виконується, або скасовується, що виключає можливість часткових або некоректних змін. PostgreSQL підтримує високі рівні ізоляції транзакцій, такі як Repeatable Read та Serializable, що унеможливлює виникнення типових проблем конкурентного доступу, зокрема неконсистентного або непослідовного читання.

Суттєвою перевагою PostgreSQL є ефективна підтримка складних запитів, зокрема під час роботи з часовими інтервалами. Наявність спеціалізованих типів даних, таких як tsrange, а також розвинений набір функцій та операторів, дають змогу здійснювати швидку перевірку перетину часових періодів. Наприклад, у функції перевірки доступності кімнати під час бронювання використання операторів перетину на індексованих стовпцях типу range забезпечує високу продуктивність навіть за значної кількості записів [43].

Важливою характеристикою PostgreSQL є її розширюваність, зокрема підтримка формату JSONB (JSON Binary), який дозволяє зберігати, індексувати

та ефективно опрацьовувати напівструктуровані дані. Такий підхід створює гнучкі можливості для подальшої еволюції системи, адже зміни у структурі збережених даних не потребують модифікації схеми реляційних таблиць. Це поєднує переваги традиційних реляційних баз даних і NoSQL-рішень. Крім того, PostgreSQL підтримує геопросторові дані, що відкриває можливість реалізації додаткових функцій, наприклад, пошуку об’єктів за географічною близькістю.

Для взаємодії із системою управління базами даних обрано інструмент об’єктно-реляційного відображення SQLAlchemy. Ця бібліотека є потужним ORM-засобом для Python, який забезпечує безпечну, зручну та масштабовану роботу з даними. Однією з головних переваг SQLAlchemy є підвищений рівень безпеки, оскільки система автоматично екранує вхідні дані та застосовує параметризовані запити, що унеможливлює SQL-ін’єкції.

Застосування об’єктно-орієнтованого підходу в SQLAlchemy сприяє підвищенню зрозумілості та підтримуваності коду. Розробник може працювати з елементами бази даних як із повноцінними об’єктами Python, використовуючи атрибути та методи класів замість безпосереднього формування SQL-запитів. Такий підхід зменшує ризик помилок і сприяє структурованості коду. Додатковою перевагою є портативність ORM-рішень: SQLAlchemy підтримує широкий спектр СУБД (SQLite, MySQL, Oracle, MS SQL Server), що дозволяє без значних змін коду мігрувати на іншу платформу у разі потреби.

Поєднання PostgreSQL і SQLAlchemy формує надійну, безпечну та масштабовану основу для побудови вебзастосунків. PostgreSQL забезпечує стійкість, відповідність ACID-вимогам, ефективне опрацювання структурованих і неструктурованих даних, тоді як SQLAlchemy гарантує безпечну взаємодію з базою, гнучкість розроблення та підтримку принципів об’єктно-орієнтованого програмування. Разом вони створюють оптимальний технологічний стек для реалізації вебдодатків, орієнтованих на високу продуктивність і надійність.

Клієнтська частина вебзастосунку реалізована з використанням фреймворку Next.js, який є метафреймворком для бібліотеки React. Next.js розширює функціональні можливості React, забезпечуючи підтримку

рендерингу на стороні сервера, маршрутизації, оптимізації продуктивності та збірки застосунку. Це рішення дає змогу створювати швидкі, гнучкі та масштабовані вебінтерфейси, що відповідають сучасним вимогам до продуктивності та зручності користувацького досвіду [44].

Однією з ключових переваг Next.js є підтримка рендерингу на стороні сервера (Server-Side Rendering, SSR) та React Server Components (RSC). Використання SSR забезпечує швидке відображення контенту, оскільки HTML-сторінка генерується на сервері ще до її завантаження користувачем. На відміну від підходу Client-Side Rendering, де користувач спочатку отримує порожній HTML-файл, у цьому випадку зміст сторінки доступний миттєво, що істотно скорочує час до появи контенту та підвищує якість користувацької взаємодії.

Додатковою перевагою серверного рендерингу є покращена оптимізація для пошукових систем (SEO). Оскільки пошукові роботи отримують повністю сформований HTML-код, вони можуть швидко індексувати сторінки без необхідності виконання JavaScript-коду. Це особливо важливо для застосунків, де контент (наприклад, описи або ціни) повинен бути доступний для пошукової індексації [45].

Використання React Server Components є сучасним підходом до побудови архітектури застосунку, який дозволяє виконувати частину компонентів виключно на сервері. Такий підхід зменшує обсяг JavaScript-коду, що завантажується у браузер, і підвищує швидкодію застосунку. Крім того, це забезпечує можливість безпосереднього доступу до даних (наприклад, із бази даних чи API) без необхідності створення додаткових проміжних ендпоїнтів, що спрощує структуру проєкту.

Ще однією важливою особливістю Next.js є файлова маршрутизація (App Router), яка базується на структурі проєктних файлів. Кожен файл у каталозі app/ автоматично відповідає певному маршруту, що істотно спрощує налаштування та підтримку проєкту. Наприклад, сторінка, розташована у файлі app/rooms/page.js, автоматично відповідає маршруту /rooms. Такий підхід підвищує передбачуваність структури коду, зменшує ризик помилок і покращує

організацію проєкту. Крім того, Next.js забезпечує ефективну клієнтську навігацію між сторінками, що створює відчуття роботи з односторінковим застосунком (SPA), навіть якщо початкове завантаження здійснюється на сервері.

Next.js успадковує всі переваги екосистеми React, що забезпечує доступ до великої кількості готових інтерфейсних рішень і бібліотек. Використання React гарантує сумісність із численними UI-компонентами (зокрема Material UI, Ant Design), бібліотеками керування станом (Redux, Zustand) та інструментами для роботи з формами.

Крім того, фреймворк надає можливість створення серверних ендпоїнтів API безпосередньо у каталозі app/api. Це дає змогу реалізовувати прості серверні функції або проксі-сервіси без потреби у створенні окремого бекенд-сервера. Такий підхід є зручним для швидкого тестування або розширення функціональності застосунку.

Використання Next.js у поєднанні з React забезпечує оптимальний баланс між продуктивністю, структурованістю та масштабованістю клієнтської частини вебзастосунку. Підтримка SSR і RSC підвищує швидкодію та SEO-оптимізацію, файлова маршрутизація спрощує архітектуру, а потужна екосистема React гарантує гнучкість і стабільність розроблення сучасних вебрішень.

Для забезпечення відтворюваності середовища, спрощення процесів розгортання та підвищення стабільності роботи системи використовується технологія Docker. Ця платформа призначена для розроблення, доставки та виконання програмних застосунків у контейнерах. Контейнер являє собою легку, портативну та незалежну від операційної системи одиницю програмного забезпечення, яка містить усі необхідні компоненти для запуску застосунку, зокрема код, бібліотеки, системні інструменти та параметри конфігурації.

Використання Docker забезпечує ізоляцію компонентів системи, що відповідає принципам мікросервісної архітектури. Кожен сервіс застосунку функціонує у власному контейнері, наприклад, серверна частина (API на Python/FastAPI) розгортається окремо від клієнтської частини (Next.js) та бази

даних (PostgreSQL). Такий підхід дозволяє уникнути конфліктів залежностей, оскільки кожен контейнер має власне середовище виконання. Це усуває проблему, відому як "dependency hell", і забезпечує стабільність роботи програмного комплексу, незалежно від програмного забезпечення, встановленого на хост-системі. Ізоляція контейнерів також підвищує рівень безпеки, обмежуючи доступ між процесами та файловими системами різних сервісів.

Однією з найважливіших переваг Docker є портативність та відтворюваність середовища. Завдяки стандартизованим конфігураційним файлам (Dockerfile) забезпечується єдність середовища розроблення, тестування та експлуатації. Усі етапи життєвого циклу застосунку – від локальної розробки до розгортання на виробничому сервері – виконуються в ідентичних умовах. Це усуває типову проблему, коли програмне забезпечення працює на одній машині, але не запускається на іншій. Крім того, Docker значно скорочує час початкового налаштування середовища розробника: достатньо однієї команди для запуску всіх необхідних сервісів, що підвищує ефективність командної роботи [46].

Контейнеризація також сприяє спрощенню процесів масштабування та розгортання. Docker-контейнери легко інтегруються з оркестраторами, такими як Docker Swarm або Kubernetes, що забезпечує швидке масштабування застосунку відповідно до поточного навантаження. У разі збільшення кількості запитів система може автоматично запускати додаткові екземпляри контейнерів без змін у вихідному коді. Оновлення компонентів виконуються шляхом заміни контейнера новою версією, що зменшує час простою та дозволяє оперативно повертатися до попередньої конфігурації у разі виявлення помилок.

Застосування Docker у складі технологічного стеку забезпечує стабільність, гнучкість і масштабованість системи. Обрана архітектура включає такі основні компоненти: бекенд (Python/FastAPI), що характеризується високою продуктивністю та асинхронним опрацюванням запитів; фронтенд (Next.js), який поєднує переваги серверного рендерингу й потужної екосистеми React; базу даних PostgreSQL, відому своєю ACID-надійністю та підтримкою складних типів

даних; а також Docker, який гарантує ізоляцію середовища, портативність і спрощене масштабування [47].

Зазначена комбінація технологій утворює сучасний, надійний і масштабований технологічний стек, який забезпечує високу швидкість розроблення, стабільність функціонування та готовність до подальшого розширення системи бронювання.

2.5. Етапи програмної реалізації

2.5.1. Налаштування середовища розробки (foundation)

Першим етапом програмної реалізації було створення надійного, ізольованого та відтворюваного середовища розробки. Цей "фундамент" є критично важливим для забезпечення стабільності системи та її подальшого розгортання (DevOps). Етап було розділено на три паралельні завдання: контейнеризація бази даних, ініціалізація бекенду та ініціалізація фронтенду.

Контейнеризація бази даних за допомогою Docker

Для організації зберігання даних обрано систему керування базами даних PostgreSQL**.** Замість традиційного локального встановлення СУБД було застосовано технологію контейнеризації Docker, що забезпечує ізоляцію середовища виконання та підвищує портативність програмного забезпечення. Для автоматизованого розгортання та керування контейнером бази даних було створено конфігураційний файл `docker-compose.prod.yml`**,** який визначає параметри запуску відповідного сервісу (*db*).

Ключові кроки:

1. Встановлення Docker Desktop.

2. Створення файлу `docker-compose.prod.yml` у корені проєкту (Рис. 2.8).

3. Опис сервісу `db` з використанням офіційного образу `postgres:16-alpine`.

4. Налаштування змінних середовища (env_file) для визначення імені бази, користувача та пароля.

5. Підключення постійного сховища (Docker Volume) `postgres_data` для збереження даних БД навіть після зупинки контейнера.

6. Відкриття порту `5432` для доступу з локальної машини (DBeaver) та бекенду.

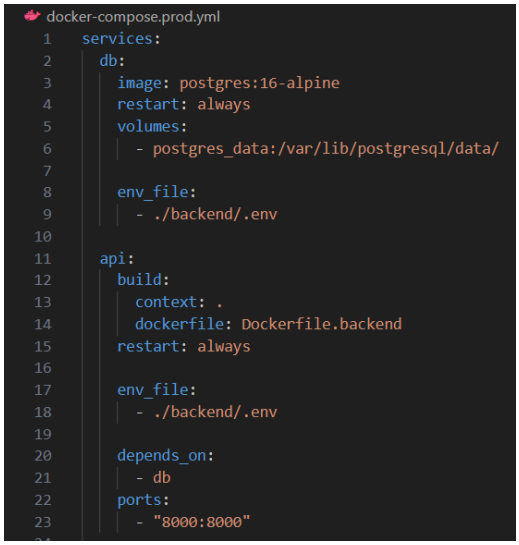


Рисунок 2.8 – Фрагмент коду (`docker-compose.prod.yml`)

Ініціалізація серверної частини (Backend) з використанням FastAPI

Серверна складова програмного забезпечення (API) реалізована мовою програмування Python із застосуванням фреймворку FastAPI. З метою ізоляції залежностей проєкту від глобального середовища системи було створено віртуальне середовище (venv), що забезпечує контрольоване керування бібліотеками та підвищує відтворюваність програмного середовища.

Ключові кроки:

1. Створення папки `backend/`.
2. Створення віртуального середовища командою: python -m venv venv
3. Активація середовища: .\venv\Scripts\Activate
4. Встановлення ключових бібліотек (FastAPI, Uvicorn, SQLAlchemy, Psycopg2, Pydantic та ін.) та їх фіксація у файлі `requirements.txt`.

Ініціалізація клієнтської частини (Frontend) з використанням Next.js

Для розроблення сучасного, високопродуктивного та адаптивного інтерфейсу користувача застосовано фреймворк Next.js, побудований на основі

React**,** із підтримкою TypeScript**.** Такий вибір забезпечує ефективну організацію компонентної архітектури, типобезпечність коду та підвищує масштабованість клієнтського застосунку.

Ключові кроки:

1. Встановлення Node.js та пакетного менеджера `npm`.

2. Створення папки `frontend/`.

3. Ініціалізація проєкту командою `npx create-next-app@latest`, обравши ключові опції:

- TypeScript: (Так) – для статичної типізації та надійності коду.
- Tailwind CSS: (Так) – для "utility-first" підходу до стилізації.
- App Router: (Так) – для використання нової архітектури Server Components.

4. Встановлення додаткових бібліотек (`lucide-react` для іконок, `dayjs` для роботи з датами, `jest` для тестування).

2.5.2. Реалізація серверної частини (Backend API)

Проєктування моделей даних

За допомогою ORM-бібліотеки SQLAlchemy було розроблено реляційні моделі, що описують основні сутності системи: Room (номер), Booking (бронювання) та User (користувач). Використання об’єктно-реляційного підходу забезпечило високий рівень абстракції від безпосередніх SQL-запитів, підвищило безпеку доступу до даних (зокрема, шляхом запобігання SQL-ін’єкціям) та оптимізувало процес взаємодії з базою даних.

```python
backend > app > models.py
44
45  class Booking(Base):
46      __tablename__ = "bookings"
47
48      id = Column(Integer, primary_key=True, index=True)
49
50      room_id = Column(Integer, ForeignKey("rooms.id"), nullable=False)
51
52      client_name = Column(String, nullable=False)
53      client_email = Column(String, nullable=False)
54
55      date_from = Column(Date, nullable=False)
56      date_to = Column(Date, nullable=False)
57
58      total_price = Column(Float)
59      is_confirmed = Column(Boolean, default=False)
60
61      discount_code = Column(String, nullable=True)
62
63      room = relationship("Room", back_populates="bookings")
64
```

Рисунок 2.9 – Фрагмент коду `models.py` з визначенням класу `Booking`

Бізнес-логіка та операції CRUD

Важливою складовою забезпечення надійності функціонування системи є реалізація функції is_room_available, призначеної для запобігання подвійному бронюванню одного номера на однакові дати. Зазначена функція здійснює перевірку наявності перетину часових інтервалів між новими запитами користувачів та вже підтвердженими записами про бронювання, що зберігаються в базі даних.

```python
14  def is_room_available(
15      db: Session, room_id: int, date_from: date, date_to: date
16  ) -> bool:
17
18      room = db.query(models.Room).filter(models.Room.id == room_id).first()
19      if not room or not room.is_available:
20          return False
21
22      if date_from >= date_to:
23          return False
24
25      overlapping_bookings = (
26          db.query(models.Booking)
27          .filter(
28              models.Booking.room_id == room_id,
29              models.Booking.is_confirmed == True,
30              models.Booking.date_from < date_to,
31              models.Booking.date_to > date_from
32          )
33          .first()
34      )
```

Рисунок 2.10 – Фрагмент коду запобігання подвійного бронювання

Автентифікація користувачів із використанням JWT.

Для забезпечення захисту адміністративної панелі та відповідних API-ендпоінтів (зокрема, операцій додавання номерів і перегляду всіх бронювань) було впроваджено механізм автентифікації на основі JSON Web Tokens (JWT).

Після успішного проходження процедури входу в систему (запит POST /auth/token) адміністратор отримує автентифікаційний токен, який необхідно передавати в заголовку запиту Authorization: Bearer <token> для здійснення подальших дій у межах захищених ресурсів системи.

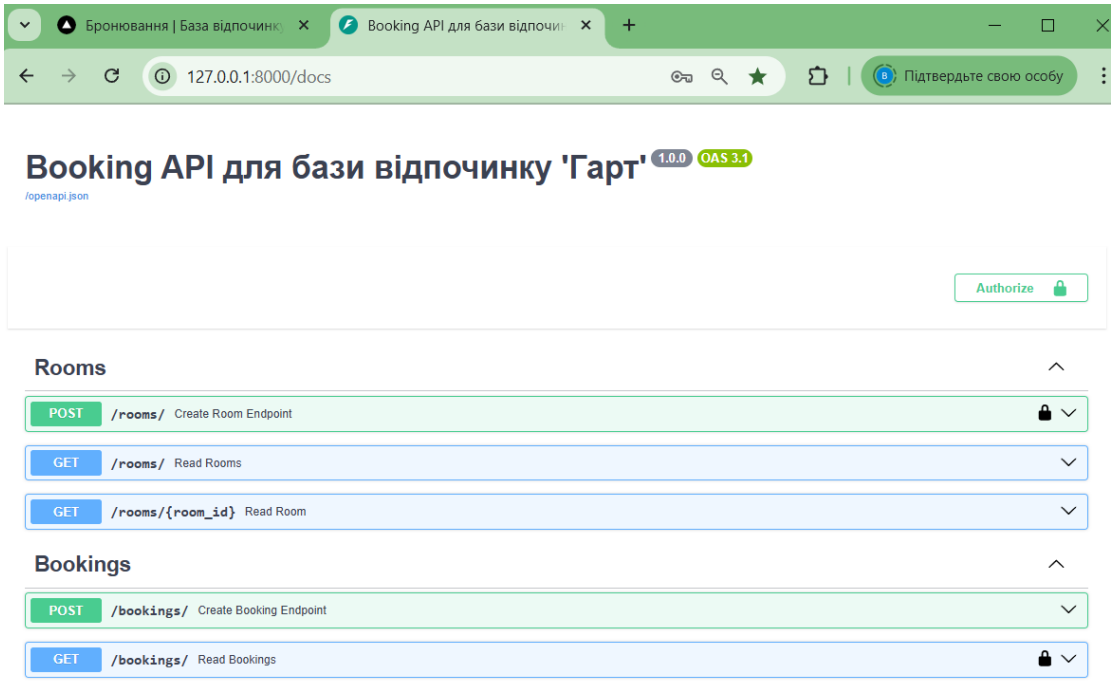


Рисунок 2.11 – Swagger UI з успішною авторизацією адміністратора

2.5.3. Реалізація клієнтської частини (Frontend UI)

Клієнтська частина вебзастосунку реалізована з використанням фреймворку Next.js 14+ (App Router), що дозволило поєднати переваги серверного рендерингу (SSR) для публічних сторінок та інтерактивності клієнтських компонентів (CSR) для форм. Дизайн інтерфейсу розроблено за принципом Mobile-First з використанням бібліотеки Tailwind CSS.

Публічна частина системи (Landing Page та каталог)

Основним призначенням публічної частини вебзастосунку є залучення користувачів і забезпечення зручного інструменту для пошуку доступних номерів.

Крок 1: Розроблення головної сторінки (Landing Page).

Головна сторінка (app/page.tsx) виконує функцію візитівки бази відпочинку та реалізована у вигляді статичного серверного компонента, що забезпечує швидке завантаження контенту (Рис. 2.12).

Для оптимізованого відображення головного зображення об’єкта відпочинку використано компонент next/image, який автоматично адаптує зображення до різних розмірів екранів.

Додано заклики до дії (Call-to-Action buttons), що спрямовують користувача до каталогу номерів або форми бронювання, сприяючи підвищенню конверсії відвідувачів у клієнтів (Рис. 2.13).

frontend > app > page.tsx > HomePage

```tsx
import Link from "next/link";
import Image from "next/image";
import { Mountain, Hotel, ArrowRight } from "lucide-react";

export default function HomePage() {
  return (
    <div className="flex flex-col items-center justify-center min-h-[70vh] text-center px-4">

      <Image
        src="/baza-hart-main.jpg"
        alt="База відпочинку Гарт"
        width={800}
        height={400}
        className="w-full max-w-4xl h-auto rounded-xl shadow-2xl mb-8 object-cover"
        priority
      />

      <h1 className="text-5xl md:text-6xl font-extrabold text-gray-900 mb-4 tracking-tight">
        Ласкаво просимо до бази практик «Гарт»
      </h1>
      <p className="text-xl text-gray-600 max-w-2xl mb-8">
        Автоматизована система бронювання для комфортного відпочинку на природі.
        Плануйте свій заїзд, перевіряйте доступність та отримуйте динамічні
        ціни.
      </p>
```

Рисунок 2.12 – Фрагмент коду (`frontend/app/page.tsx`)

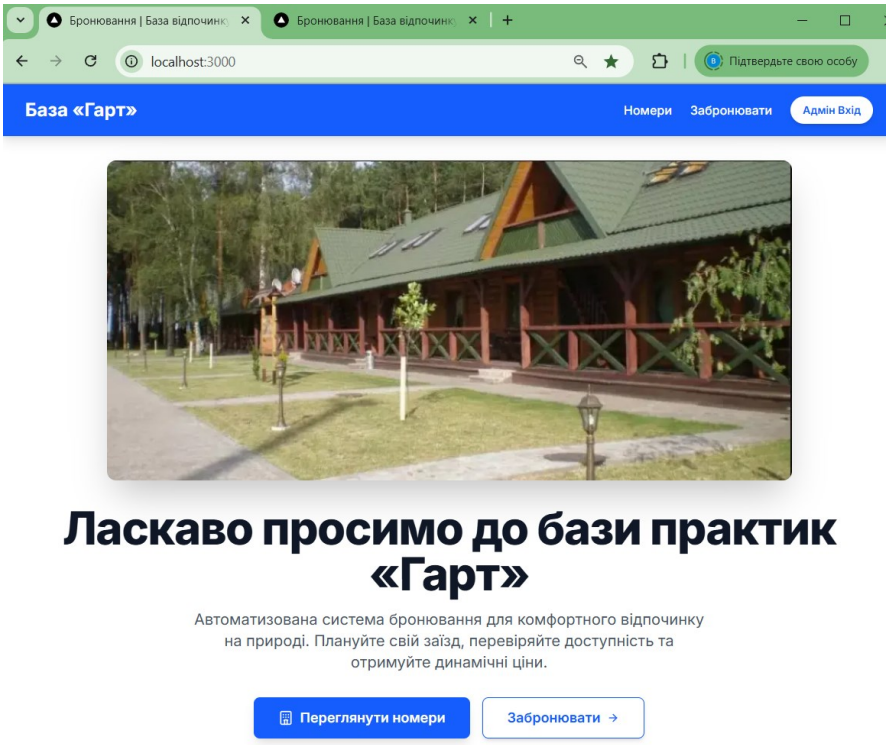


Рисунок 2.13 – Реалізація головної сторінки

Крок 2: Розроблення каталогу номерів із можливістю фільтрації

Сторінку каталогу (app/rooms/page.tsx) реалізовано у вигляді асинхронного серверного компонента, що дає змогу здійснювати запити до Backend API безпосередньо з сервера Next.js. Такий підхід знижує навантаження на клієнтський браузер і забезпечує приховування внутрішньої адреси API, підвищуючи безпеку застосунку (Рис. 2.14).

Для реалізації інтерфейсу фільтрації створено клієнтський компонент RoomFilter, який змінює URL-параметри (наприклад, `?date_from=...&min_capacity=...`) відповідно до обраних користувачем критеріїв пошуку. Серверна сторінка зчитує дані параметри та передає їх у GET-запит до FastAPI, забезпечуючи динамічне оновлення каталогу відповідно до заданих умов (Рис. 2.15).

```
async function getRooms(queryString: string): Promise<Room[]> {
  try {
    const response = await fetch(`${API_BASE_URL}/rooms/?${queryString}`, {
      cache: "no-store",
    });

    if (!response.ok) {
      console.error(`API Error: ${response.status} ${response.statusText}`);
      return [];
    }

    return response.json();
  } catch (error) {
    console.error("Помилка отримання номерів. Перевірте FastAPI:", error);
    return [];
  }
}
```

Рисунок 2.14 – Фрагмент коду організації асинхронної функції

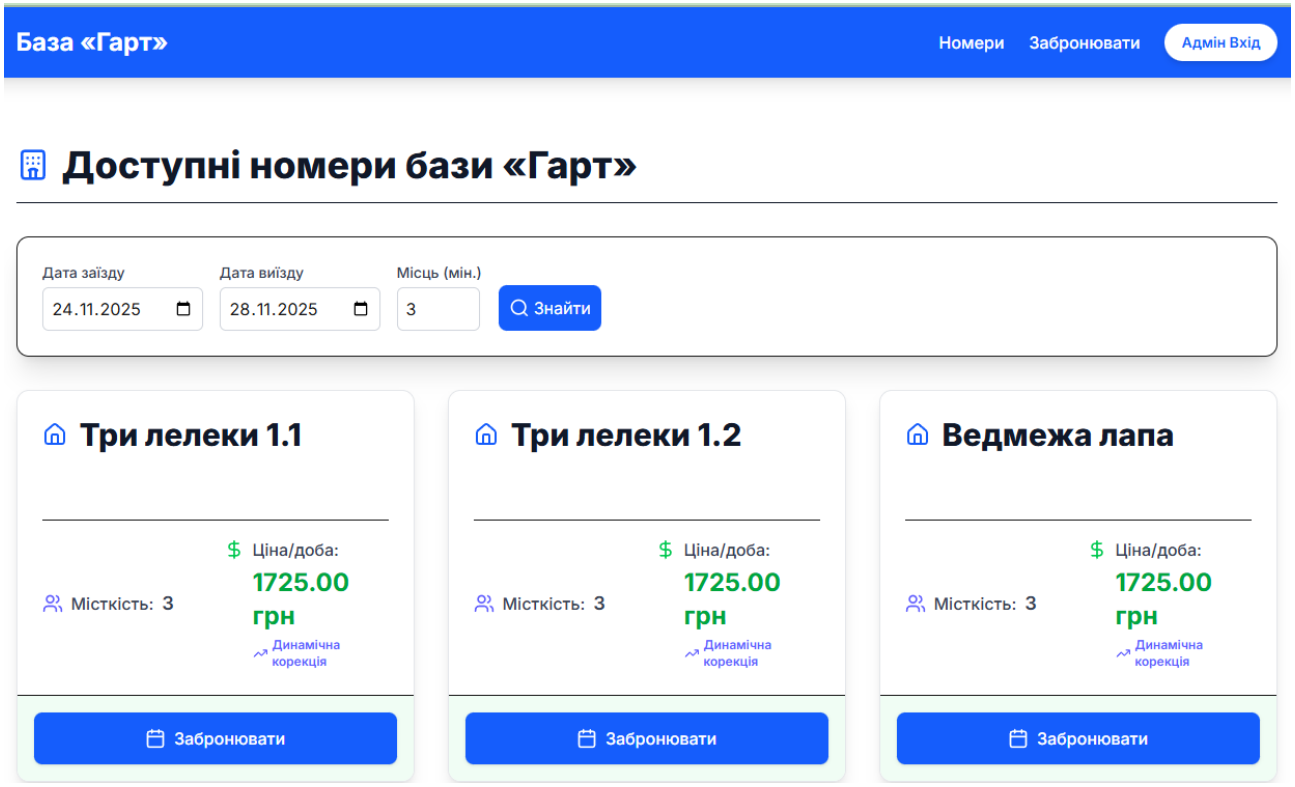


Рисунок 2.15 – Форма фільтрації номерів за датою та кількістю місць

Система бронювання

Система бронювання передбачає реалізацію складної взаємодії з користувачем, тому її було структуровано у вигляді двох основних компонентів

– серверного контейнера, який відповідає за завантаження даних, та клієнтської форми, що забезпечує інтерактивність користувацького інтерфейсу.

Сторінка app/booking/page.tsx виконує роль контролера, який попередньо завантажує список усіх або відфільтрованих номерів із бекенду та передає їх у клієнтський компонент форми як props. Такий підхід забезпечує швидке початкове завантаження сторінки без появи візуальних затримок або індикаторів очікування.

Компонент форми (BookingForm.tsx) реалізовано із використанням React-хуків (useState) для керування станом полів введення (Рис. 2.16). Форма містить динамічний випадаючий список (Рис. 2.17), який автоматично генерується на основі отриманих даних, що усуває потребу у ручному введенні ідентифікаторів номерів. Перед відправленням даних здійснюється клієнтська валідація, яка перевіряє заповненість полів та коректність зазначених дат. Взаємодія з API відбувається за допомогою методу fetch, який надсилає POST-запит для створення нового бронювання та опрацьовує відповіді сервера, включно з успішним створенням запису (код 201) або виявленням конфлікту (код 409).

```
176     <form onSubmit={handleSubmit} className="space-y-6">
177
178       <div>
179         <label
180           htmlFor="room_id"
181           className="block text-sm font-medium text-gray-700"
182         >
183           Оберіть номер
184         </label>
185         <select
186           name="room_id"
187           value={formData.room_id}
188           onChange={handleChange}
189           required
190           className="mt-1 block w-full border-gray-300 rounded-md shadow-sm p-3"
191           disabled={rooms.length === 0}
192         >
193           {rooms.length === 0 && (
194             <option value="0">--- Номерів не знайдено ---</option>
195           )}
196           {rooms.map((room) => (
197             <option key={room.id} value={room.id}>
198               {room.name} (Місць: {room.capacity} |{" "}
199               {room.base_price.toFixed(2)} грн)
200             </option>
201           ))}
202         </select>
```

Рисунок 2.16 – Фрагмент коду де організовано випадаючий список

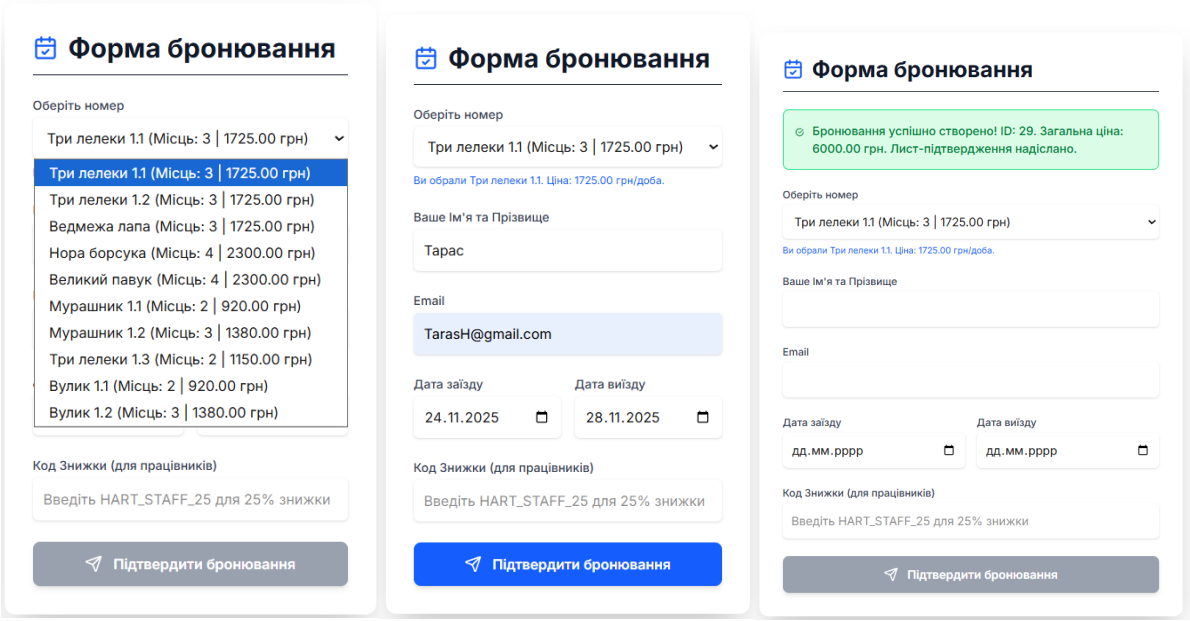


Рисунок 2.17 – Сторінка, де розкрито випадаючий список з номерами

2.5.4 Реалізація інтелектуальних функцій

Реалізовані інтелектуальні функції спрямовані на автоматизацію бізнес-процесів і підвищення прибутковості бази відпочинку шляхом використання алгоритмів динамічного ціноутворення та інтерактивної візуалізації даних.

Одним із ключових завдань проєкту стало створення гнучкої системи ціноутворення, здатної автоматично реагувати на часові фактори. З цією метою було розроблено окремий серверний модуль pricing_service.py, який відповідає за логіку формування вартості проживання. Застосування бібліотеки Pendulum забезпечило ефективну роботу з датами та визначення днів тижня.

Алгоритм динамічного ціноутворення враховує низку чинників. По-перше, сезонність, що передбачає застосування підвищувального коефіцієнта (наприклад, 1.3) у літні місяці. По-друге, дні тижня, для яких у п’ятницю, суботу та неділю встановлюється додатковий коефіцієнт (наприклад, 1.15). По-третє, персональні знижки, які надаються за умови введення дійсного промокоду, зокрема для співробітників університету передбачено знижку у розмірі 25% (Рис. 2.18-2.21).

Таким чином, система динамічного ціноутворення забезпечує адаптивність цінової політики та сприяє оптимізації прибутковості підприємства.

```
HIGH_SEASON_MONTHS = [6, 7, 8]
HIGH_SEASON_MULTIPLIER = 1.30
WEEKEND_MULTIPLIER = 1.15
MAX_OCCUPANCY_MULTIPLIER = 1.40 #
OCCUPANCY_THRESHOLD = 0.80
STAFF_DISCOUNT_CODE = "HART_STAFF_25"
STAFF_DISCOUNT_PERCENT = 0.75
```

Рисунок 2.18 – Параметри для динамічного ціноутворення

```
def calculate_dynamic_price(db: Session, room_id: int, base_price: float, date_from: date, date_to: date) -> float:

    total_price = 0.0
    current_date = date_from

    while current_date < date_to:
        multiplier = get_price_multiplier(db, room_id, current_date)
        daily_price = base_price * multiplier
        total_price += daily_price

        current_date += timedelta(days=1)

    return total_price
```

Рисунок 2.19 – Функція розрахунку загальної ціни

```
def apply_staff_discount(total_price: float, discount_code: str | None) -> float:

    if discount_code == STAFF_DISCOUNT_CODE:
        return total_price * STAFF_DISCOUNT_PERCENT
    return total_price
```

Рисунок 2.20 – Функція застосування знижки для працівників

```python
def get_price_multiplier(db: Session, room_id: int, check_date: date) -> float:

    multiplier = 1.0

    if check_date.month in HIGH_SEASON_MONTHS:
        multiplier *= HIGH_SEASON_MULTIPLIER

    day_of_week = pendulum.instance(check_date).day_of_week
    if day_of_week in [pendulum.FRIDAY, pendulum.SATURDAY, pendulum.SUNDAY]:
        multiplier *= WEEKEND_MULTIPLIER

    from . import crud

    occupancy_rate = crud.get_room_occupancy_rate(db, room_id, check_date)

    if occupancy_rate >= OCCUPANCY_THRESHOLD:
        demand_factor = 1 + (occupancy_rate * (MAX_OCCUPANCY_MULTIPLIER - 1))
        multiplier *= demand_factor

    return multiplier
```

Рисунок 2.21 – Функція визначення загального коефіцієнту ціни

Механізм розрахунку ціни було інтегровано у відповідні API-ендпоінти: GET /rooms, який забезпечує відображення актуальної вартості номерів на поточну дату, та POST /bookings, що використовується для визначення фінальної ціни замовлення під час оформлення бронювання. На клієнтській стороні додано візуальний індикатор, який інформує користувача про застосування динамічних цінових коефіцієнтів, що підвищує прозорість процесу формування вартості та довіру до системи (Рис. 2.22).

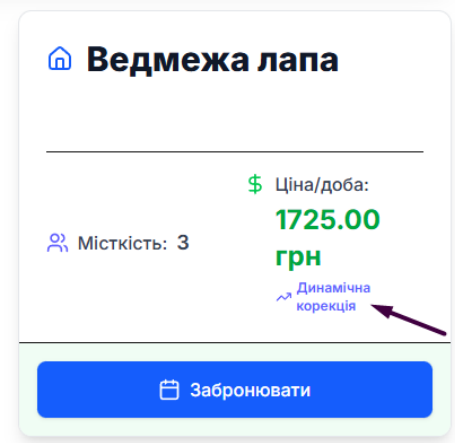


Рисунок 2.22 – Динамічна корекція

На рисунку 2.23 подано три приклади бронювання одного й того самого номера на дві доби, що відрізняються за вартістю. Бронювання з ID 30 охоплює період із понеділка по середу та має вартість 3000 грн**.** Бронювання з ID 31 здійснене на період із п’ятниці по неділю, що характеризується вищою ціною у зв’язку із застосуванням підвищувального коефіцієнта вихідних днів. Бронювання з ID 32 охоплює той самий період, але передбачає застосування промокоду, що забезпечує знижку для працівників від базової вартості.

Останні бронювання

ID	КЛІЄНТ (EMAIL)	НОМЕР (ID)	ДАТИ	ЦІНА
32	Ангеліна (angelinka@gmail.com)	5	2025-12-26 - 2025-12-28	2760.00 грн
31	Ангеліна (angelinka@gmail.com)	5	2025-12-19 - 2025-12-21	3450.00 грн
30	Ангеліна (angelinka@gmail.com)	5	2025-12-15 - 2025-12-17	3000.00 грн

Рисунок 2.23 – Таблиця останніх бронювань

Автоматичні сповіщення (Email)

З метою підвищення якості користувацького досвіду та зниження навантаження на адміністратора системи було реалізовано механізм автоматичних транзакційних сповіщень електронною поштою.

Для забезпечення надсилання повідомлень використано бібліотеку fastapi-mail, яка підтримує асинхронну відправку листів. Такий підхід дає змогу уникнути блокування основного потоку виконання програми під час встановлення з’єднання з поштовим сервером. Конфігураційні параметри, зокрема логін, пароль додатка та адреса SMTP-сервера, зберігаються у захищеному файлі .env, що підвищує безпеку системи (Рис. 2.24).

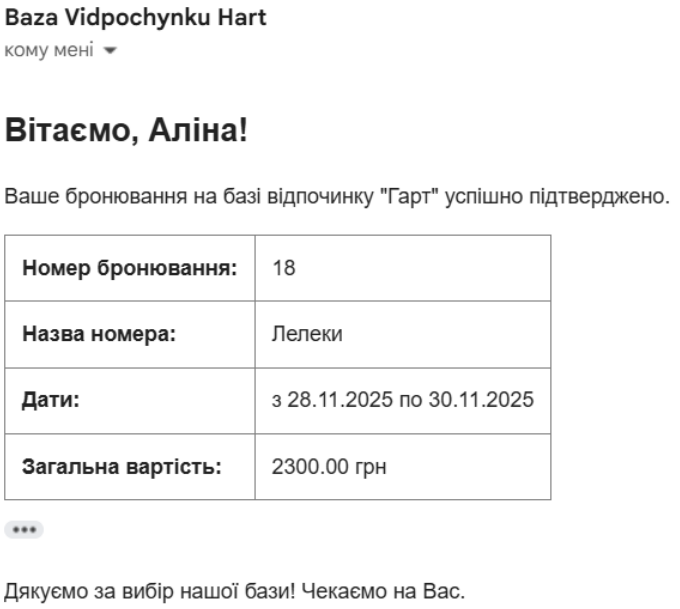
Після успішного збереження бронювання у базі даних система автоматично генерує HTML-шаблон листа, який містить деталі замовлення, і надсилає його на електронну адресу користувача, вказану під час оформлення бронювання (Рис. 2.25). Така автоматизація сприяє оперативному інформуванню клієнтів та оптимізує роботу адміністрації бази відпочинку.

```
    booking_details = {
        "id": db_booking.id,
        "client_name": db_booking.client_name,
        "client_email": db_booking.client_email,
        "room_name": room.name,
        "date_from": db_booking.date_from.strftime("%d.%m.%Y"),
        "date_to": db_booking.date_to.strftime("%d.%m.%Y"),
        "total_price": db_booking.total_price,
    }

    await email_service.send_booking_confirmation(booking_details)

    return db_booking
```

Рисунок 2.24 – Генерація та відправка листа



Baza Vidpochynku Hart

кому мені

Вітаємо, Аліна!

Ваше бронювання на базі відпочинку "Гарт" успішно підтверджено.

Номер бронювання:	18
Назва номера:	Лелеки
Дати:	з 28.11.2025 по 30.11.2025
Загальна вартість:	2300.00 грн

•••

Дякуємо за вибір нашої бази! Чекаємо на Вас.

Рисунок 2.25 – Лист підтвердження

Адміністративна панель (Admin Dashboard).

Адміністративна панель є центральним елементом управління інформаційною системою бази відпочинку «Гарт». Доступ до неї надається виключно авторизованим користувачам після успішної перевірки JWT-токена, що забезпечує належний рівень безпеки.

Інтерфейс панелі розроблено за принципом «єдиного вікна», що гарантує швидкий доступ до ключових показників ефективності (KPI) та інструментів управління контентом.

Основними складовими адміністративної панелі є інформаційні віджети, блок управління та таблиця останніх бронювань.

Інформаційні віджети (KPI Cards) відображають агреговані показники діяльності системи. Віджет «Всього бронювань» показує загальну кількість

успішних транзакцій, що дозволяє оперативно оцінити рівень активності клієнтів. Віджет «Орієнтовний дохід» автоматично обчислює сумарну вартість підтверджених бронювань, забезпечуючи можливість моніторингу фінансових показників у режимі реального часу (Рис. 2.26).

Блок управління містить основні інструменти для адміністрування контенту. Центральним елементом цього блоку є кнопка «Додати номер», яка ініціює процес створення нового запису в номерному фонді бази даних. Після натискання відбувається клієнтське перенаправлення на захищений маршрут /admin/rooms/add, де розміщено форму введення параметрів нового номера, таких як назва, опис, місткість і базова ціна.

Таблиця «Останні бронювання» надає зведений огляд останніх транзакцій у системі. Вона відображає унікальний ідентифікатор замовлення, контактні дані клієнта, обраний номер, період проживання та фінальну вартість. Завдяки цьому елементу адміністратор може оперативно відстежувати нові бронювання без необхідності безпосереднього звернення до бази даних, що підвищує ефективність управління.

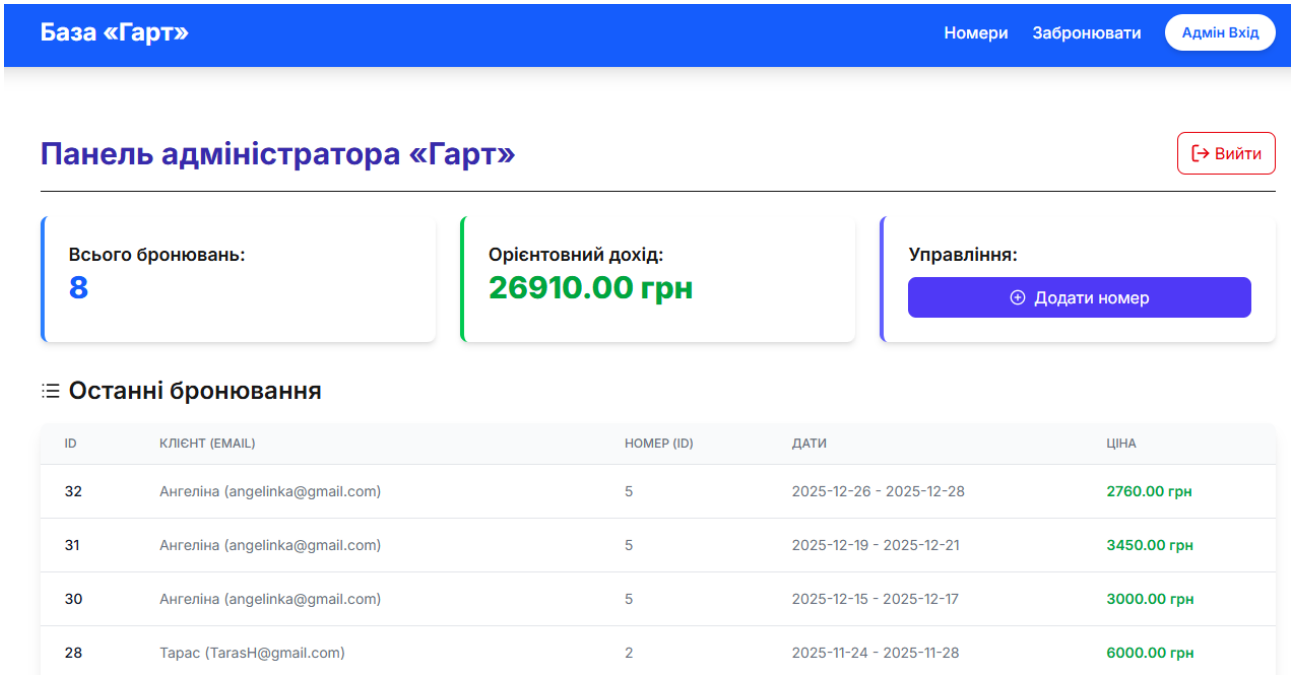


Рисунок 2.26 – Панель адміністратора

Така структура інтерфейсу суттєво підвищує ефективність роботи персоналу, скорочуючи час, необхідний для виконання рутинних операцій з моніторингу та управління контентом.

Кнопка «Додати номер» запускає процес створення нового об’єкта у системі. Після її натискання здійснюється клієнтське перенаправлення на захищений маршрут /admin/rooms/add за допомогою функції `router.push('/admin/rooms/add')`, що реалізовано у фреймворку Next.js (рис.2.27).

На відкритій сторінці відображається форма створення номера, у якій адміністратор вводить основні параметри – назву, опис, максимальну місткість, базову ціну та статус доступності. Після заповнення форми введені дані проходять клієнтську валідацію і надсилаються POST-запитом на захищений ендпоінт API /rooms/. У разі успішного виконання запиту новий номер автоматично відображається у публічному каталозі та стає доступним для бронювання користувачами.

Такий підхід забезпечує безперервність робочого процесу, знижує навантаження на адміністратора та сприяє оперативному оновленню інформації в системі.

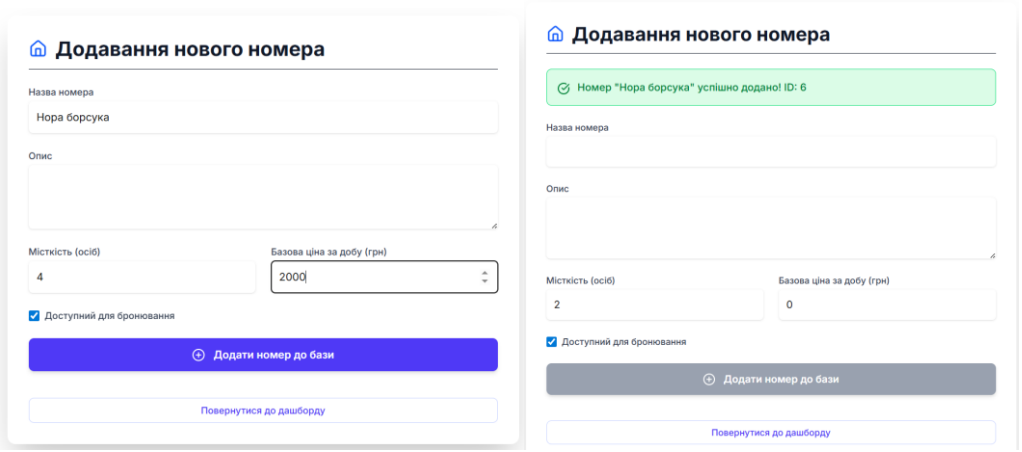


Рисунок 2.27 – Додавання нового номера

Інтерактивна сітка зайнятості (Admin Dashboard)

Для адміністратора було розроблено інтерактивний інструмент візуалізації завантаженості номерного фонду, який дає змогу оперативно оцінювати стан бронювань без необхідності переглядати табличні списки.

Логіка побудови сітки реалізована на клієнтській стороні у компоненті OccupancyGrid.tsx, який отримує список усіх номерів і пов'язаних із ними бронювань. За допомогою бібліотеки dayjs генерується масив дат, що охоплює наступні тридцять днів і використовується для побудови часової шкали відображення (Рис. 2.28).

Під час рендерингу формується двовимірна таблиця, у якій рядки відповідають окремим номерам, а стовпці – календарним датам. Для кожної клітинки здійснюється перевірка наявності бронювання, що перекриває відповідний день. Якщо номер зайнятий, клітинка позначається червоним кольором; якщо вільний – зеленим. Натискання на вільну клітинку відкриває модальне вікно, у якому адміністратор може виконати швидке ручне бронювання (Рис. 2.29).

```
141  const days: dayjs.Dayjs[] = [];
142  let currentDay = startDate;
143  while (currentDay.isBefore(endDate)) {
144    days.push(currentDay);
145    currentDay = currentDay.add(1, "day");
146  }
147
148  const getBookingForDay = (roomId: number, day: dayjs.Dayjs) => {
149    return bookings.find(
150      (booking) =>
151        booking.room_id === roomId &&
152        day.isBetween(
153          dayjs(booking.date_from),
154          dayjs(booking.date_to),
155          "day",
156          "[)"
157        )
158    );
159  };
```

Рисунок 2.28 – Логіка побудови сітки номерів

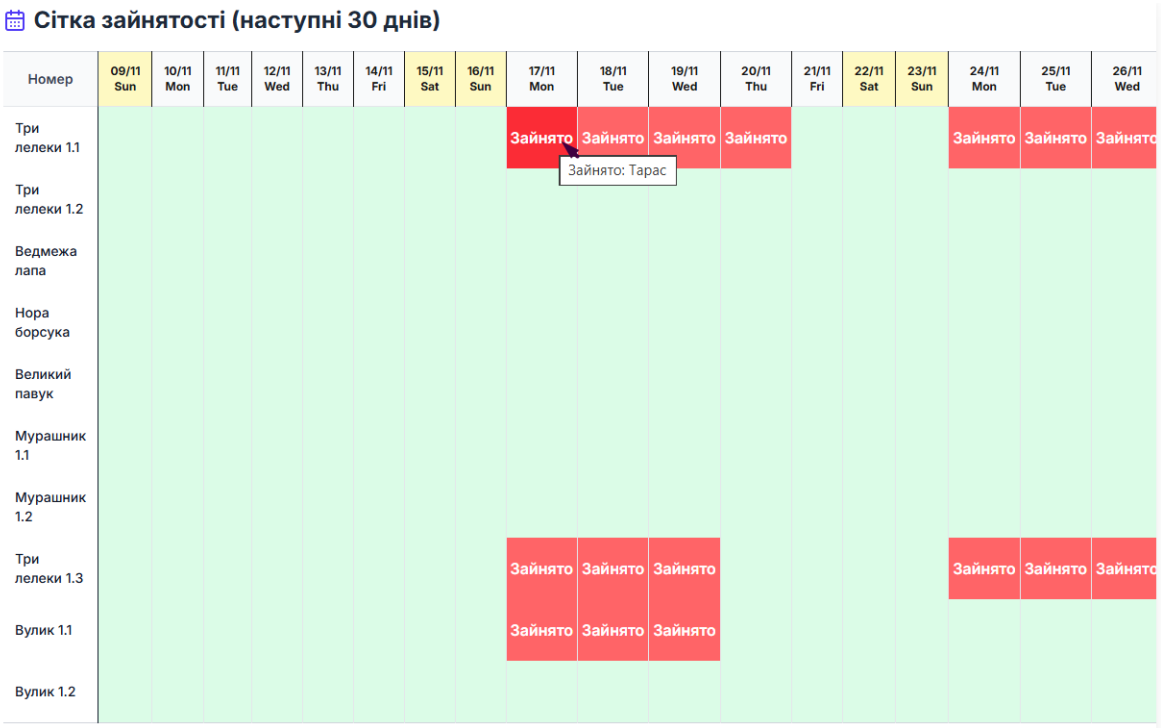


Рисунок 2.29 – Сітка зайнятості номерів

Такий підхід забезпечує наочне відображення завантаженості номерів, прискорює процес прийняття управлінських рішень і підвищує ефективність роботи адміністративного персоналу.

2.6. Організація тестування та налагодження програмного засобу

Тестування та налагодження є невід'ємними етапами життєвого циклу програмної розробки, оскільки забезпечують перевірку якості, надійності та відповідності створеного програмного засобу функціональним вимогам. У межах проведеного дослідження було реалізовано багаторівневу стратегію тестування, спрямовану на комплексну перевірку працездатності системи.

Модульне тестування (Unit Testing) бекенду

Основною метою модульного тестування є перевірка коректності роботи окремих, ізольованих функціональних компонентів бізнес-логіки, особливо тих, що забезпечують ключовий функціонал системи бронювання.

Для реалізації процесу тестування використано фреймворк Pytest, який відзначається гнучкістю, простотою синтаксису та широкими можливостями автоматизації перевірок. У ролі тимчасового середовища збереження даних

застосовано комбінацію SQLAlchemy та SQLite (in-memory), що дозволяє створювати ізольовану базу даних в оперативній пам’яті для кожного тестового запуску. Такий підхід забезпечує атомарність тестів, незалежність їх виконання та високу швидкість опрацювання.

Було розроблено набір тестів, зокрема файл tests/test_business_logic.py, який охоплює критичні аспекти бізнес-логіки. Одним із ключових напрямів перевірки стала функція is_room_available, що реалізує механізм запобігання подвійному бронюванню одного номера на ті самі дати (Рис. 2.30). Даний тест гарантує, що система правильно опрацьовує часові інтервали бронювань і не допускає конфліктів між записами.

```
47  def test_is_room_available_overlap_failure(db: Session):
48      room = Room(id=1, name="Тест", capacity=2, base_price=100, is_available=True)
49      existing_booking = Booking(
50          id=1,
51          room_id=1,
52          client_name="Іван",
53          client_email="ivan@example.com",
54          date_from=date(2025, 1, 5),
55          date_to=date(2025, 1, 10),
56          total_price=500,
57          is_confirmed=True
58      )
59      db.add_all([room, existing_booking])
60      db.commit()
61
62      is_available = is_room_available(db, room_id=1, date_from=date(2025, 1, 8), date_to=date(2025, 1, 12))
63      assert is_available is False
```

Рисунок 2.30 – Фрагмент коду тестування на перетин номерів

Тестування механізму динамічного ціноутворення (pricing_service)

Метою даного етапу тестування є перевірка правильності реалізації алгоритму розрахунку вартості проживання з урахуванням сезонних та календарних коефіцієнтів. Особлива увага приділяється автоматичному коригуванню базової ціни залежно від пори року та дня тижня.

Тестування спрямоване на підтвердження того, що система коректно застосовує підвищувальний коефіцієнт для літнього сезону, який становить 30% від базової вартості, а також додатковий коефіцієнт для вихідних днів (п’ятниця–неділя) у розмірі 15%. Перевірка виконується шляхом моделювання різних

комбінацій дат і базових тарифів, що дозволяє оцінити стабільність і точність роботи функції pricing_service у різних сценаріях використання (Рис. 2.31).

```python
@pytest.mark.parametrize("test_date, expected_price", [
    (date(2025, 7, 4), 149.5), # 4 Липня 2025 = П'ятниця
    (date(2025, 8, 4), 130.0), # 4 Серпня 2025 = Понеділок
    (date(2025, 11, 8), 115.0), # 8 Листопада 2025 = Субота
    (date(2025, 10, 1), 100.0) # 1 Жовтня 2025 = Середа
])
def test_calculate_dynamic_price_factors(db: Session, test_date: date, expected_price: float):
    room = Room(id=1, name="Тест", capacity=2, base_price=100, is_available=True)
    db.add(room)
    db.commit()

    price = calculate_dynamic_price(db, room_id=1, base_price=100.0, date_from=test_date, date_to=test_date + timedelta(days=1))

    assert price == pytest.approx(expected_price)
```

Рисунок 2.31 – Фрагмент коду тестування ціни

Тестування механізму застосування знижок.

Метою цього етапу є перевірка правильності реалізації логіки опрацювання знижкових кодів у процесі бронювання. Особливу увагу зосереджено на сценарії використання службових кодів для працівників, що передбачають зниження вартості проживання на 25% від базової ціни.

Тестування проводиться шляхом емуляції введення дійсного знижкового коду під час оформлення замовлення. Система має коректно розраховувати кінцеву вартість бронювання з урахуванням зазначеного відсоткового зменшення, забезпечуючи стабільність та точність роботи модуля розрахунку кінцевої суми.

```python
def test_apply_staff_discount(db: Session):

    room = Room(id=1, name="Тест", capacity=2, base_price=100, is_available=True)
    db.add(room)
    db.commit()

    base_dynamic_price = calculate_dynamic_price(db, room_id=1, base_price=100.0, date_from=date(2025, 10, 1), date_to=date(2025, 10, 2))
    assert base_dynamic_price == 100.0

    price_with_discount = apply_staff_discount(base_dynamic_price, STAFF_DISCOUNT_CODE)
    assert price_with_discount == pytest.approx(75.0)

    price_without_discount = apply_staff_discount(base_dynamic_price, "WRONG_CODE")
    assert price_without_discount == 100.0
```

Рисунок 2.32 – Фрагмент коду тестування знижки

Тестування було проведено шляхом запуску пакета з кореневого каталогу backend/ за допомогою команди pytest. Усі вісім тестових сценаріїв успішно пройшли перевірку, що засвідчує стабільність і надійність основної бізнес-логіки

програмного забезпечення. Отримані результати підтвердили коректність реалізації ключових функціональних модулів системи.

Тестування компонентів фронтенду за допомогою Jest

Метою тестування є перевірка коректності відображення та функціональної поведінки основних компонентів інтерфейсу користувача на клієнтській стороні. Для реалізації цього етапу було використано сучасний інструментарій, що забезпечує повноцінну емуляцію роботи браузера. Основним середовищем тестування обрано фреймворк Jest, який інтегровано у середовище *Next.js*. Для моделювання взаємодії користувача із компонентами застосовано бібліотеку React Testing Library (RTL), що дозволяє виконувати тести, орієнтовані на реальні сценарії використання, зокрема пошук елементів за текстом або роллю. Компонент jest-environment-jsdom забезпечує імітацію середовища браузера (DOM) у консольному режимі.

Налаштування системи тестування фронтенду охоплювало встановлення необхідних залежностей до проєкту *frontend* та створення конфігураційних файлів у його кореневому каталозі. Файл *jest.config.mjs* містить параметри адаптації Jest до роботи з *Next.js*, *TypeScript* та аліасами шляхів виду *@/components/*. Файл *jest.setup.js* імпортує розширення *@testing-library/jest-dom*, яке надає доступ до додаткових методів перевірки, таких як *toBeInTheDocument()*.

Наведена конфігурація забезпечує можливість ізольованого тестування React-компонентів, що сприяє підвищенню стабільності та прогнозованості функціонування клієнтської частини системи.

```
import nextJest from "next/jest.js";

const createJestConfig = nextJest({
  dir: "./",
});

const config = {
  coverageProvider: "v8",
  testEnvironment: "jsdom",
  setupFilesAfterEnv: ["<rootDir>/jest.setup.js"],
  moduleNameMapper: {
    "^@/(.*)$": "<rootDir>/$1",
  },
};

export default createJestConfig(config);
```

Рисунок 2.33 – Фрагмент коду `frontend/jest.config.mjs`

Третім етапом було розроблення тестового сценарію (*Test Case*) для перевірки коректності роботи окремих компонентів інтерфейсу. Зокрема, створено тест *Header.test.tsx*, який відповідає за оцінювання функціональності головного навігаційного компонента *Header*.

У межах тестування перевірено два основні аспекти. Перший полягає у визначенні правильності рендерингу елемента бренду, тобто відображення назви «База «Гарт»» у структурі компонента. Другий аспект спрямований на перевірку наявності всіх основних навігаційних посилань, зокрема пунктів меню «Номери», «Забронювати» та «Адмін Вхід».

Отримані результати засвідчили коректне функціонування навігаційного компонента, що підтверджує відповідність інтерфейсу вимогам до зручності користування та повноти навігації (Рис. 2.34).

```tsx
import React from "react";
import { render, screen } from "@testing-library/react";
import Header from "@/components/Header";
import { describe, it, expect } from "@jest/globals";

describe("Header Component", () => {
  it('should render the brand name "База «Гарт»"', () => {
    render(<Header />);

    const brandElement = screen.getByRole("link", { name: /База «Гарт»/i });

    expect(brandElement).toBeInTheDocument();
  });

  it("should have navigation links for Rooms, Booking, and Admin", () => {
    render(<Header />);

    const roomsLink = screen.getByRole("link", { name: /Номери/i });
    const bookingLink = screen.getByRole("link", { name: /Забронювати/i });
    const adminLink = screen.getByRole("link", { name: /Адмін Вхід/i });

    expect(roomsLink).toBeInTheDocument();
    expect(bookingLink).toBeInTheDocument();
    expect(adminLink).toBeInTheDocument();
  });
});
```

Рисунок 2.34 – Фрагмент коду `frontend/tests/Header.test.tsx`

Виконання тестів здійснювалося за допомогою команди `npm run test`, яка визначена у файлі *package.json* та викликає інструмент *Jest* у режимі безперервного спостереження (*--watchAll*). Результати тестування засвідчили успішне проходження двох перевірок, що підтвердило коректність рендерингу базових компонентів інтерфейсу користувача та відповідність їхньої поведінки очікуваним вимогам.

2.7. Аналіз отриманих результатів дослідження, рекомендації щодо використання та впровадження

У процесі виконання кваліфікаційної роботи було розроблено та впроваджено повнофункціональний вебзастосунок для автоматизації процесів бронювання, який повністю відповідає завданням.

У результаті проведеного дослідження та реалізації програмного рішення досягнуто низку важливих технічних результатів.

Одним із ключових досягнень є усунення проблеми атомарності операцій та запобігання ситуаціям подвійного бронювання. У традиційних або спрощених системах управління даними часто виникає явище «гонки даних», коли кілька користувачів одночасно намагаються здійснити бронювання одного й того самого номера. У межах створеного вебзастосунку цю проблему розв'язано за допомогою спеціально розробленої бізнес-логіки, реалізованої у функції `is_room_available` модуля `crud.py`. Алгоритм здійснює перевірку перетину дат (`models.Booking.date_from < date_to` та `models.Booking.date_to > date_from`) перед створенням нового запису в базі даних. Успішне проходження модульних тестів підтвердило надійність і коректність роботи даного механізму.

Другим важливим результатом є впровадження інтелектуального механізму динамічного ціноутворення. Статичні ціни, що не враховують сезонні чи попитові коливання, знижують прибутковість системи. Для вирішення цієї проблеми створено окремий модуль `pricing_service.py`, який реалізує евристичну модель управління дохідністю. Алгоритм автоматично коригує вартість проживання залежно від сезонності (підвищення на 30% у літній період) та днів тижня (підвищення на 15% у вихідні дні – п'ятницю, суботу та неділю). Крім того, система підтримує застосування персоналізованих знижок, зокрема для працівників університету. Ці результати підтверджують доцільність використання стеку технологій Python/FastAPI для інтеграції інтелектуальних бізнес-алгоритмів.

Третій результат полягає у підвищенні ефективності адміністративного управління. Запропонована адміністративна панель із візуальною сіткою зайнятості (`OccupancyGrid.tsx`) дає змогу оперативно оцінювати стан номерного фонду та виконувати створення бронювань безпосередньо з інтерфейсу системи. Такий підхід суттєво скорочує час виконання рутинних операцій і підвищує продуктивність працівників.

Ще одним значним досягненням є повна автоматизація комунікацій із клієнтами. Реалізований модуль електронної пошти (`email_service.py`) забезпечує автоматичне формування та відправлення підтверджень бронювання після успішного завершення транзакції. Клієнт отримує лист із деталізованою інформацією про замовлення та фінальну суму до оплати. Це рішення дозволило зменшити навантаження на адміністратора і підвищити рівень користувацького досвіду.

Для адміністратора вебсистеми процес роботи починається з первинного налаштування, яке передбачає створення облікового запису адміністратора за допомогою інтерфейсу Swagger UI (/docs) через запит POST /auth/register/admin. Після цього здійснюється наповнення бази даних, що виконується через сторінку «Додати номер» у панелі адміністратора, де вводиться актуальна інформація про номерний фонд бази відпочинку «Гарт». У подальшій операційній діяльності адміністратор може використовувати «Сітку зайнятості» для швидкого моніторингу завантаженості номерів і створення бронювань вручну, наприклад, у разі телефонних звернень клієнтів. Для проведення аналітики рекомендується застосовувати віджети «Всього бронювань» та «Орієнтовний дохід» на дашборді, які забезпечують оперативну оцінку ключових показників ефективності.

З боку клієнта взаємодія із системою починається з етапу пошуку, який здійснюється на сторінках «Номери» або «Бронювання» із використанням фільтрів за датою та місткістю. Після цього користувач обирає потрібний номер із динамічно оновлюваного випадаючого списку та переходить до оформлення бронювання. Для отримання персональної знижки клієнт має можливість ввести промокод HART_STAFF_25 у спеціальне поле форми, що забезпечує автоматичне застосування знижки у розмірі 25%.

Для ефективного впровадження розробленої системи бронювання рекомендовано здійснювати розгортання у середовищі віртуального приватного сервера (VPS), що функціонує під керуванням операційної системи Linux, наприклад Ubuntu 22.04. На сервері мають бути встановлені програмні засоби

Docker та Docker Compose, які забезпечують контейнеризацію компонентів і спрощують процес деплойменту. Перед запуском системи необхідно заповнити конфігураційний файл *backend/.env*, у якому слід вказати актуальні параметри середовища, зокрема змінну *DATABASE_URL* із безпечним паролем, а також облікові дані *MAIL_USERNAME* та *MAIL_PASSWORD*, отримані через створення «Пароля додатка» (App Password) у Google для безпечної SMTP-автентифікації. Після завершення конфігурації виконання команди `docker compose -f docker-compose.prod.yml up --build -d` у кореневій директорії проєкту забезпечує автоматичну збірку та запуск усіх контейнерів системи, зокрема бази даних, серверної частини API та вебінтерфейсу.

Подальший розвиток системи передбачає її масштабування та підвищення рівня інтелектуальності. Зокрема, можливо розширити наявну евристичну модель динамічного ціноутворення, реалізовану у модулі *pricing_service.py*, шляхом інтеграції бібліотеки *Scikit-learn* для побудови моделі лінійної регресії. Такий підхід дозволить прогнозувати попит не лише на основі сезонних чинників, а й на основі історичних даних щодо завантаженості номерного фонду, що зробить систему самонавчальною. Крім того, доцільним напрямом розвитку є інтеграція платіжних сервісів, таких як *LiqPay* або *Fondy*, що дасть змогу реалізувати автоматичну зміну статусу бронювання з *is_confirmed=False* на *is_confirmed=True* лише після підтвердження успішної онлайн-оплати.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було встановлено, що поставлена мета – спроєктувати та розробити комплексний вебзастосунок для автоматизації процесу бронювання номерів на прикладі бази відпочинку «Гарт» – досягнута повністю.

Проведений аналіз предметної області дозволив виявити основні недоліки існуючих ручних систем бронювання, зокрема високий ризик помилок через людський фактор, зокрема подвійне бронювання, відсутність механізмів гнучкого ціноутворення та значні витрати часу на адміністративні операції. На основі цього було спроєктовано сучасну трьохланкову архітектуру вебзастосунку, яка характеризується відмовостійкістю, масштабованістю та модульністю. Система складається з незалежних компонентів: серверної частини, реалізованої на Python із застосуванням асинхронного фреймворку FastAPI, що забезпечує високу продуктивність; клієнтського інтерфейсу, розробленого на базі Next.js (React) із TypeScript, який гарантує швидкий і адаптивний користувацький досвід; та системи зберігання даних на основі PostgreSQL, ізольованої у Docker-контейнері для забезпечення портативності й стабільності роботи.

У межах дослідження вирішено ключове практичне завдання – запобігання подвійному бронюванню. Розроблена бізнес-логіка здійснює перевірку перетину часових інтервалів перед створенням нового запису в базі даних. Коректність роботи алгоритму підтверджена результатами модульного тестування із застосуванням pytest, що засвідчує надійність запропонованого рішення.

Встановлено також ефективність рішення щодо динамічного ціноутворення. Модуль pricing_service.py автоматично коригує вартість номерів на основі евристичних правил, що враховують сезонність, дні тижня та рівень завантаженості бази. Інтеграція системи персоналізованих знижок для працівників університету (25%) підтверджує можливість адаптації тарифної політики до різних категорій користувачів.

Дослідження засвідчило, що запропонована система забезпечує повну автоматизацію ключових бізнес-процесів. Клієнтська частина реалізує інтуїтивний інтерфейс для пошуку та бронювання номерів, а адміністративна панель, захищена на основі JWT, надає засоби управління контентом, інструменти CRUD та інтерактивну сітку зайнятості для візуального моніторингу та оперативного бронювання. Додатково інтеграція SMTP-сервісу забезпечує автоматичне надсилання клієнтам підтверджень бронювання.

Якість і надійність системи підтверджено комплексним тестуванням, яке включало модульне тестування бекенду (pytest) та компонентне тестування фронтенду (Jest). Розроблений програмний продукт готовий до практичного впровадження завдяки повній контейнеризації всіх сервісів (PostgreSQL, FastAPI, Next.js) із використанням Dockerfile і docker-compose.prod.yml.

Таким чином, усі завдання кваліфікаційної кваліфікаційної роботи виконано в повному обсязі, а розроблений вебзастосунок має практичну цінність і може бути використаний для оптимізації роботи бази відпочинку та підвищення її прибутковості.

Подальший розвиток системи може включати інтеграцію платіжних онлайн-шлюзів (наприклад, LiqPay) для здійснення миттєвої оплати бронювань, а також розширення модулю динамічного ціноутворення до повноцінної моделі машинного навчання, що дозволить прогнозувати попит на основі історичних даних і адаптивно коригувати стратегію ціноутворення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Vrontis D., Massoud M., Dennaoui H., Nemar S. E. The impact of e-service on hotels' booking: adjusted TAM framework for customers' intentions to book hotels online. *GBER*. 2022. Vol. 26, No. 3. P. 285. DOI: 10.1504/GBER.2022.122385.
2. Atkinson S., Lee K. Design and Implementation of a Study Room Reservation System: Lessons from a Pilot Program Using Google Calendar. *College & Research Libraries*. 2018. Vol. 79, No. 7. Pp. 916–930. DOI: 10.5860/crl.79.7.916.
3. Xuan W. Implementation of a secure room booking system at the University of Manitoba Libraries. *International Journal of Librarianship*. 2021. Vol. 6, No. 2. Pp. 63–72. DOI: 10.23974/ijol.2021.vol6.2.194.
4. Kolesárová S., Šenková A., Kormaníková E., Šambronská K. Customer Reviews of Accommodation as an Important Factor in Choosing and Booking Accommodation: Analysis of Conditions in V4 Countries. *Administrative Sciences*. 2024. Vol. 14, No. 12. P. 308. DOI: 10.3390/admsci14120308.
5. Srivastava N., Mishra A., Malhotra G. Exploring flow experience for hotel's branded booking apps. *International Journal of Retail & Distribution Management*. 2025. Vol. 53, No. 10–11. Pp. 1093–1106. DOI: 10.1108/IJRDM-01-2025-0061.
6. Molina-Collado A., Gómez-Rico M., Sigala M., Molina M. V., Aranda E., Salinero Y. Correction to: Mapping tourism and hospitality research on information and communication technology: a bibliometric and scientific approach. *Information Technology & Tourism*. 2022. Vol. 24, No. 2. Pp. 341–342. DOI: 10.1007/s40558-022-00230-z.
7. Majid G. M., Tussyadiah I., Kim Y. R., Pal A. Intelligent automation for sustainable tourism: a systematic review. *Journal of Sustainable Tourism*. 2023. Vol. 31, No. 11. Pp. 2421–2440. DOI: 10.1080/09669582.2023.2246681.
8. Ivanov S., Webster C. Conceptual Framework of the Use of Robots, Artificial Intelligence and Service Automation in Travel, Tourism, and Hospitality Companies. In: Ivanov S., Webster C. (eds.). *Robots, Artificial Intelligence, and*

Service Automation in Travel, Tourism and Hospitality. Emerald Publishing Limited, 2019. Pp. 7–37. DOI: 10.1108/978-1-78756-687-320191001.

9. OtelMS. Modern Hotel PMS and Channel Management Integration [Електронний ресурс]. URL: https://otelms.com/ (дата звернення: 14.05.2025).
10. Hotelminder. Hotel Booking Systems Overview and Trends [Електронний ресурс]. URL: https://hotelminder.com (дата звернення: 03.09.2025).
11. Borse S. Cloud-based Hotel Management System. *International Research Journal of Engineering and Technology (IRJET)*. 2022. Vol. 9, No. 12. Pp. 1187–1191.
12. Никига О., Мороз С., Журило М. Особливості систем резервування в туризмі. *Економіка та суспільство*. 2025. No. 78. DOI: 10.32782/2524-0072/2025-78-2.
13. AltexSoft. What is a hotel property management system (PMS): Products and features [Електронний ресурс]. URL: https://www.altexsoft.com/blog/hotel-property-management-systems-products-and-features/ (дата звернення: 22.03.2025).
14. Central Reservation System (CRS): Guide for hotels. 2025 [Електронний ресурс]. URL: https://www.siteminder.com/r/central-reservation-system/ (дата звернення: 11.07.2025).
15. Data Model for a Hotel Management System. 2025 [Електронний ресурс]. URL: https://www.red-gate.com/blog/data-model-for-hotel-management-system (дата звернення: 05.09.2025).
16. Bin Mahathir A. A., Li Hang P., Zhun Hei C., Tong Hua L., Ul Amin N. Design and Implementation of a Relational Database System for Efficient Facility and Booking Management. 03 March 2025. DOI: 10.20944/preprints202503.0070.v1.
17. Győrödi C. A., Dumşe-Burescu D. V., Zmaranda D. R., Győrödi R. Ş., Gabor G. A., Pecherle G. D. Performance Analysis of NoSQL and Relational Databases with CouchDB and MySQL for Application's Data Storage. *Applied Sciences*. 2020. Vol. 10, No. 23. P. 8524. DOI: 10.3390/app10238524.
18. A guide to ACID properties in database management systems [Електронний ресурс]. URL: https://www.mongodb.com/resources/basics/databases/acid-transactions?utm_source (дата звернення: 29.04.2025).

19. Kliukovkin G. How to Solve Race Conditions in a Booking System. 2023 [Електронний ресурс]. URL: https://hackernoon.com/how-to-solve-race-conditions-in-a-booking-system?utm_source (дата звернення: 16.08.2025).
20. Lakshmi Nivas, Mallik Reddy V. Data Privacy and Security in E-commerce: Modern Database Solutions. *International Journal of Advanced Engineering Technologies and Innovations*. 2023. Vol. 1, No. 03. Pp. 248–263.
21. Ye C., Hwang W.-C., Chen K., Yu X. Polaris: Enabling Transaction Priority in Optimistic Concurrency Control. *Proceedings of the ACM on Management of Data*. 2023. Vol. 1, No. 1. Pp. 1–24. DOI: 10.1145/3588724.
22. Solat S. Security of Electronic Payment Systems: A Comprehensive Survey. 2017. *arXiv*. DOI: 10.48550/ARXIV.1701.04556.
23. Chippagiri S., Ramesh A. PCI DSS: A Critical Analysis of Implementation, Effectiveness, and Legislative Impact in Payment Card Security. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*. 2025. Vol. 11, No. 1. Pp. 1258–1266. DOI: 10.32628/CSEIT251112115.
24. Pandey V., Kushwaha G. S., Srivastava S. Factors Influencing Adoption of Digital Payments: A PLS SEM and ANN Approach. *Purushartha*. 2024. Vol. 16, No. 2. Pp. 91–104. DOI: 10.21844/16202116207.
25. van der Boor M., Borst S. C., van Leeuwaarden J. S. H., Mukherjee D. Scalable load balancing in networked systems: A survey of recent advances. 2018. DOI: 10.48550/ARXIV.1806.05444.
26. Filho M., Pimentel E., Pereira W., Maia P. H. M., Cortés M. I. Self-Adaptive Microservice-based Systems – Landscape and Research Opportunities. 2021. *arXiv*. DOI: 10.48550/ARXIV.2103.08688.
27. Kavitha D., Singh A. K., Chauhan S. Real Time Hotel Booking Demand Optimization. *International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)*. 2024. Pp. 519–527. DOI: 10.48175/IJARSCT-17978.
28. Lima Santos L., Gomes C., Malheiros C., Crespo C., Bento C. Factors Influencing Hotel Revenue Management in Times of Crisis: Towards Financial Sustainability.

International Journal of Financial Studies. 2024. Vol. 12, No. 4. P. 112. DOI: 10.3390/ijfs12040112.

29. Saitta S., D'Amico V., Farinella G. Dynamic Price Prediction for Revenue Management System in Hospitality Sector. In: *Proceedings of the 13th International Conference on Data Science, Technology and Applications*. Dijon, France: SCITEPRESS, 2024. Pp. 218–228. DOI: 10.5220/0012707700003756.
30. PCI Security Standards Overview [Електронний ресурс]. URL: https://www.pcisecuritystandards.org/standards/?utm_source (дата звернення: 19.03.2025).
31. Sun S., Law R., Hyun S. S. Exploration of Hotel Reservation Through Mobile Online Travel Agencies. *Journal of Tourism Research*. 2024. Vol. 26, No. 4. P. e2734. DOI: 10.1002/jtr.2734.
32. 70+ online travel booking statistics & trends [Updated for 2025] [Електронний ресурс]. URL: https://www.perk.com/blog/online-travel-booking-statistics/?utm_source (дата звернення: 07.08.2025).
33. Myrhodskyy A. V., Romanyuk O. V., Romanyuk O. N., Titova N. V. Development of a high availability method for configuration management software. *Optoelectronic Information-Power Technologies*. 2023. Vol. 46, No. 2. Pp. 64–75. DOI: 10.31649/1681-7893-2023-46-2-64-75.
34. Система онлайн бронювання Booking.com [Електронний ресурс]. URL: https://www.booking.com/ (дата звернення: 25.05.2025).
35. Бронювання готелів, квартир, хостелів Hotels24.ua [Електронний ресурс]. URL: https://hotels24.ua/uk/?utm_source (дата звернення: 30.09.2025).
36. Подобова оренда від власників по всій Україні [Електронний ресурс]. URL: https://doba.ua/ (дата звернення: 18.06.2025).
37. Система безкоштовного бронювання готелів GoHotels.com.ua [Електронний ресурс]. URL: https://gohotels.com.ua/ (дата звернення: 09.04.2025).
38. Patkar U., Singh P., Panse H., Bhavsar S., Pandey C. Python for web development. *IJCSMC*. 2022. Vol. 11, No. 4. Pp. 36–48. DOI: 10.47760/ijcsmc.2022.v11i04.006.

39. Kaur J., Attri V. K. The versatility of Python: Applications in diverse fields. *World Journal of Advanced Engineering Technology and Sciences*. 2025. Vol. 14, No. 2. Pp. 095–098. DOI: 10.30574/wjaets.2025.14.2.0059.
40. Jyoti S. N., Islam M. R., Kudapa S. P. та ін. The role of test automation frameworks in enhancing software reliability: A review of Selenium, Python, and API testing tools. *IJBEI*. 2024. Vol. 04, No. 04. Pp. 01–34. DOI: 10.63125/bvv8r252.
41. FastAPI History, Design and Future [Електронний ресурс]. URL: https://fastapi.tiangolo.com/history-design-future/ (дата звернення: 12.03.2025).
42. Khetani S. Data integrity and security: Blockchain vs. traditional databases. 2025 URL: https://www.theseus.fi/bitstream/handle/10024/888887/Khetani_Shreya.pdf (дата звернення: 21.04.2025).
43. Makris A., Tserpes K., Spiliopoulos G., Zissis D., Anagnostopoulos D. MongoDB Vs PostgreSQL: A comparative study on performance aspects. *Geoinformatica*. 2021. Vol. 25, No. 2. Pp. 243–268. DOI: 10.1007/s10707-020-00407-w.
44. Lazuardy M. F. S., Anggraini D. Modern front end web architectures with React.js and Next.js. *Research Journal of Advanced Engineering and Science*. 2022. Vol. 7, No. 1. Pp. 132–141.
45. Analyzing the Impact of Next.JS on Site Performance and SEO. *IJCATR*. Oct. 2023. DOI: 10.7753/IJCATR1210.1004.
46. Moreau D., Wiebels K., Boettiger C. Containers for computational reproducibility. *Nature Reviews Methods Primers*. 2023. Vol. 3, No. 1. P. 50. DOI: 10.1038/s43586-023-00236-9.
47. Fischer T., Hirn D., Kul G. A Reproducible Tutorial on Reproducibility in Database Systems Research. *Proceedings of the VLDB Endowment*. 2024. Vol. 17, No. 12. Pp. 4221–4224. DOI: 10.14778/3685800.3685840.

ДОДАТКИ

Додаток А

Технічне завдання
на розробку вебзастосунку для автоматизації процесу бронювання номерів бази відпочинку (на прикладі бази «Гарт»)

1. Вступ

1.1. Найменування програмного засобу

Вебзастосунок для автоматизації процесу бронювання номерів бази відпочинку «Гарт».

1.2. Призначення і особливість застосування

Програмний засіб має забезпечувати повну автоматизацію життєвого циклу процесів бронювання номерів. У межах системи необхідно реалізувати два основні інтерфейси:

1. Публічний інтерфейс має надавати користувачам доступ до актуальної інформації щодо номерного фонду, тарифів та доступності, а також забезпечувати можливість самостійного онлайн-бронювання з миттєвим формуванням підтвердження.
2. Адміністративний інтерфейс має надавати захищені інструменти для управління номерним фондом, перегляду та моніторингу завантаженості, а також адміністрування фінансових параметрів, зокрема перевірки доходів та застосування знижок.

2. Підстави для розробки

Робота з проєктування та розробки вебзастосунку для автоматизації процесу бронювання номерів бази відпочинку здійснюється на підставі написання кваліфікаційної роботи на ступінь магістра у Волинському національному університеті імені Лесі Українки.

3. Призначення розробки

Розробка має бути спрямована на створення єдиного, надійного та автоматизованого інформаційного середовища для підтримки процесів управління номерним фондом та організації бронювання бази «Гарт».

Система повинна забезпечувати вирішення таких функціональних задач:

1. Автоматизація процесів бронювання. Програмний засіб має забезпечувати клієнтам можливість самостійно перевіряти актуальну доступність номерів у реальному часі, здійснювати їх фільтрацію за визначеними критеріями та оформлювати бронювання з миттєвим підтвердженням.
2. Централізоване управління. Необхідно реалізувати єдину адміністративну панель для персоналу, яка повинна надавати інструменти

повного контролю за статусами номерів, історією бронювань та ключовими фінансовими показниками.

3. Оптимізація прибутковості. Система має включати модуль динамічного ціноутворення, що автоматично коригуватиме вартість номерів залежно від релевантних ринкових факторів (сезонність, попит, день тижня) з метою підвищення економічної ефективності діяльності.

4. Підвищення якості обслуговування. У межах системи має бути реалізовано механізм автоматизованої комунікації з клієнтами, зокрема функція миттєвого надсилання електронних підтверджень щодо здійснених бронювань.

5. Запобігання помилкам та конфліктним станам. Програмний засіб повинен гарантувати атомарність транзакцій бронювання, що технічно унеможливить виникнення ситуацій подвійного резервування одного й того ж номера на однакові дати.

4. Вимоги до програмного засобу

4.1 Вимоги до функціональних характеристик

1. Система повинна відображати каталог номерного фонду з інформацією про назву, опис, місткість та актуальну вартість проживання.

2. Користувач повинен мати можливість фільтрувати номери за датами заїзду та виїзду, причому система має показувати лише доступні варіанти.

3. Система повинна забезпечувати можливість оформлення бронювання шляхом вибору номера, введення контактних даних і визначення дат проживання.

4. Розрахунок вартості бронювання має виконуватися автоматично з урахуванням динамічного ціноутворення та передбачених знижок.

5. Після успішного створення бронювання система має автоматично надсилати клієнту електронне підтвердження.

7. Адміністратор після входу має отримувати доступ до зведеної інформації щодо кількості бронювань і орієнтовного доходу.

8. Адміністратор повинен мати доступ до інтерактивної календарної сітки, де відображається зайнятість номерів у різні дати.

9. Система повинна надавати можливість створення ручних бронювань безпосередньо через інтерфейс календарної сітки.

13. Система повинна унеможливлювати створення подвійного бронювання на один і той же період того самого номера.

4.2. Експлуатаційні вимоги

1. Система повинна забезпечувати безперервне функціонування.

2. Під час опрацювання операцій бронювання має бути гарантована атомарність транзакцій, що передбачає їх успішне завершення або повний відкат у разі виникнення помилки.

3. Система повинна коректно реагувати на виняткові ситуації, включно з поверненням коду 409 у разі конфлікту бронювання та коду 401 у випадку несанкціонованих запитів.

4. Backend має функціонувати в асинхронному середовищі ASGI, що забезпечує обробку великої кількості паралельних запитів.

5. Frontend повинен використовувати серверний рендеринг для забезпечення швидкого завантаження публічних сторінок.

6. Усі секретні параметри, зокрема паролі до бази даних, параметри сервісів поштової відправки та секрет JWT, повинні зберігатися у змінних середовища і не повинні включатися до кодової бази.

7. Архітектура проєкту повинна підтримувати горизонтальне масштабування компонентів API та вебінтерфейсу, що забезпечує можливість їх незалежного збільшення відповідно до навантаження.

4.3. Умови експлуатації

Програмний засіб має враховувати специфіку взаємодії різних категорій користувачів та особливості програмно-апаратного середовища.

Для гостей бази відпочинку система доступна через стандартний веббраузер, що забезпечує простоту взаємодії без необхідності встановлення додаткового програмного забезпечення.

Для адміністративного персоналу передбачено використання спеціалізованої панелі управління**,** яка надає розширені можливості з адміністрування номерного фонду та управління користувачами.

З погляду серверного середовища, вебзастосунок має бути сумісним із поширеними операційними системами Windows та Linux**,** а також підтримувати розгортання на сучасних PaaS-хостингах (Platform as a Service)**,** зокрема Vercel та Railway. Це дозволить забезпечити масштабованість, гнучкість у розгортанні та оптимальне використання обчислювальних ресурсів.

4.4. Вимоги до складу і параметрів технічних засобів

Клієнтська сторона повинна функціонувати на будь-якому пристрої, здатному запускати сучасний веббраузер, включно з комп’ютерами, ноутбуками, планшетами та смартфонами.

Сервер для промислового розгортання повинен працювати під управлінням Linux та відповідати мінімальним апаратним вимогам: 2 vCPU, 4 ГБ оперативної пам’яті та SSD-накопичувач обсягом 50 ГБ.

4.5. Вимоги до інформаційної та програмної сумісності

Серверна частина повинна працювати в середовищі Docker Engine версії не нижче 20.10 та Docker Compose версії не нижче 2.0. Взаємодія між фронтендом і бекендом має здійснюватися через RESTful API з використанням протоколів HTTP або HTTPS. Формат обміну даними повинен відповідати

стандарту JSON. API має бути сумісним із будь-яким клієнтом, що підтримує стандарт HTTP, включно з можливими мобільними застосунками у майбутньому. Система повинна підтримувати інтеграцію зі стандартними SMTP-серверами, що забезпечують можливість надсилання електронних повідомлень.

5. Етапи розробки

Розроблення програмного засобу організовано за гнучкою методологією Agile. Процес поділено на п’ять послідовних спринтів, кожен з яких охоплює проєктування, реалізацію, тестування та підготовку до подальших етапів.

1. Аналіз і проєктування, що включає уточнення вимог, формування технічного завдання, вибір технологічного стеку, розроблення UML-діаграм та налаштування базової інфраструктури.
2. Реалізація ядра Backend, що передбачає створення моделей даних, реалізацію CRUD-операцій, упровадження механізму запобігання подвійному бронюванню та додавання системи автентифікації на основі JWT.
3. Розроблення Frontend, що охоплює створення публічних сторінок, інтерактивних форм і захищеної адміністративної панелі з основними засобами керування.
4. Інтеграція нових функціональних модулів, що передбачає впровадження модуля динамічного ціноутворення, системи знижок та сервісу автоматизованих email-сповіщень.
5. Тестування та розгортання, що включає проведення модульного й інтерфейсного тестування, підготовку фінальних Docker-конфігурацій та виконання завершального приймального тестування.

6. Порядок контролю та приймання

Контроль якості та приймання програмного засобу здійснюється для перевірки відповідності створеної системи вимогам технічного завдання, а також забезпечення її готовності до практичної експлуатації. Процедури контролю охоплюють комплекс заходів, спрямованих на оцінку функціональних характеристик, експлуатаційних властивостей, сумісності та надійності системи.

Основними видами контролю є:

1. Проведення контролю якості.
2. Тестування компонентів інтерфейсу.
3. Виконання інтеграційного та ручного тестування.
4. Проведення приймання програмного засобу.
5. Підтвердження готовності до введення в експлуатацію.

Додаток Б

Інструкція користувачу

1. Загальні відомості

Вебзастосунок для автоматизації процесу бронювання номерів бази відпочинку (на прикладі бази «Гарт»).

2. Функціональне призначення

Програмний засіб призначений для автоматизації процесів пошуку, перегляду та бронювання номерів клієнтами, а також для управління номерним фондом, моніторингу завантаженості та обробки замовлень адміністратором бази відпочинку.

3. Умови застосування програми

Персональний комп’ютер або ноутбук повинен мати операційну систему Windows 10 або 11 з підтримкою WSL2, macOS або дистрибутив Linux, зокрема Ubuntu чи Debian. Для зберігання образів Docker та бази даних необхідний накопичувач HDD або SSD з вільним місцем не менше десяти гігабайт. Обсяг оперативної пам’яті має становити щонайменше чотири гігабайти, при цьому рекомендується вісім гігабайт. Необхідним є встановлення програмного забезпечення Docker Desktop та веб-браузера, наприклад Google Chrome, Mozilla Firefox, Safari або Microsoft Edge. Обов’язковою умовою є наявність стабільного доступу до мережі Інтернет для завантаження образів та відправки email-сповіщень.

4. Повідомлення оператору

Система передбачає інформування користувача про результати дій та помилки:

- Якщо обраний номер вже зайнятий на вказані дати – система повертає повідомлення про помилку конфлікту даних (409 Conflict).
- Якщо користувач намагається отримати доступ до адміністративної панелі без авторизації – система перенаправляє на сторінку входу або повертає помилку 401 Unauthorized.
- У разі успішного бронювання виводиться зелене спливаюче повідомлення з ID замовлення.

5. Опис роботи програми

У розробленому веб-додатку є можливість увійти як Гість (Клієнт) або Адміністратор.

Адміністратор може:

– проходити автентифікацію за допомогою логіна та пароля для доступу до захищеної панелі управління;

– переглядати зведену статистику на дашборді (загальна кількість бронювань, сумарний дохід);

– додавати нові номери до бази даних, вказуючи їх назву, опис, місткість та базову вартість;

– переглядати інтерактивну сітку зайнятості для візуального моніторингу вільних та заброньованих дат;

– здійснювати ручне бронювання номерів через інтерфейс сітки зайнятості для клієнтів, що звернулися телефоном;

– переглядати детальний список усіх транзакцій.

Можливості гостя включають:

– перегляд каталогу доступних номерів з фотографіями та детальним описом;

– пошук та фільтрацію номерів за датами заїзду/виїзду та необхідною кількістю місць;

– перегляд актуальної вартості проживання, яка автоматично розраховується з урахуванням динамічних коефіцієнтів (сезонність, попит);

– оформлення бронювання обраного номера із зазначенням контактних даних;

– використання промокоду для отримання знижки (наприклад, для співробітників);

– отримання автоматичного листа-підтвердження на вказану електронну пошту після успішного завершення операції.

АНОТАЦІЯ

Лащук М. Т. – Проєктування та розробка вебзастосунку для автоматизації процесу бронювання номерів бази відпочинку (на прикладі бази «Гарт») – Рукопис.

Кваліфікаційна робота на здобуття освітнього ступеня «магістр» за спеціальністю 122 Комп'ютерні науки. Волинський національний університет імені Лесі Українки, Луцьк, 2025 р.

Кваліфікаційна робота присвячена проєктуванню та розробці програмного засобу для автоматизації процесів бронювання та управління номерним фондом бази відпочинку. У роботі проаналізовано сучасні підходи до цифровізації сфери гостинності, розглянуто існуючі системи бронювання та виявлено їхні недоліки.

Здійснено успішне проєктування та реалізацію, тестування та налагодження програмного продукту, призначеного для комплексного управління бронюваннями, що відзначається високим рівнем функціональності та надійності. Система дозволяє клієнтам самостійно переглядати доступність номерів у реальному часі, здійснювати бронювання та отримувати миттєві підтвердження. Однією з ключових особливостей розробленого програмного продукту є впроваджений модуль динамічного ціноутворення, який автоматично коригує вартість послуг, а також система запобігання конфліктам даних, що гарантує атомарність транзакцій. Реалізовано захищену адміністративну панель з інтерактивною сіткою зайнятості, яка дозволяє персоналу ефективно моніторити стан номерного фонду та керувати замовленнями.

Результатом реалізації цього програмного продукту є зручність, швидкість та прозорість процесу бронювання як для клієнтів, так і для адміністрації, що створює передумови для підвищення прибутковості та покращення якості обслуговування.

Ключові слова: вебзастосунок, бронювання, автоматизація, динамічне ціноутворення, FastAPI, Next.js, PostgreSQL, Docker.

ABSTRACT

Lashchuk M. T. – Design and Development of a Web Application for Automating the Reservation Process of a Recreation Facility's Rooms (Based on the "Hart" Facility) – Manuscript.

Qualification work for obtaining the educational degree of "Master" in the specialty 122 Computer Science. Lesya Ukrainka Volyn National University, Lutsk, 2025.

The qualifying paper is dedicated to the design and development of a software tool for automating booking processes and managing the room stock of a recreation center. The paper analyzes modern approaches to digitalization in the hospitality sector, reviews existing booking systems, and identifies their shortcomings.

The successful design, implementation, testing, and debugging of a software product intended for comprehensive booking management, characterized by a high level of functionality and reliability, have been carried out. The system allows clients to independently view room availability in real-time, make bookings, and receive instant confirmations. One of the key features of the developed software product is the implemented dynamic pricing module, which automatically adjusts the cost of services, as well as a data conflict prevention system that ensures the atomicity of transactions. A secure administrative panel with an interactive occupancy grid has been implemented, allowing staff to effectively monitor the status of the room stock and manage orders.

The result of the implementation of this software product is the convenience, speed, and transparency of the booking process for both clients and administration, creating prerequisites for increasing profitability and improving the quality of service. The following technical tools were used: Python, FastAPI, Next.js, React, PostgreSQL, SQLAlchemy, Docker.

Keywords: web application, booking, automation, dynamic pricing, FastAPI, Next.js, PostgreSQL, Docker.