Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп'ютерних наук

**Допущено до захисту**

Завідувач кафедри комп'ютерних наук

доктор технічних наук, професор

Андрій БОНДАРЧУК

«01» червня 2026 р.

# КВАЛІФІКАЦІЙНА РОБОТА

**на здобуття освітнього ступеня «бакалавр»**
**Спеціальність 122 Комп'ютерні науки**
**Освітня програма 122.00.01 Інформатика**

**Тема:** КЛІЄНТ-СЕРВЕРНИЙ ДОДАТОК БРОНЮВАННЯ ТА ОБЛІКУ НОМЕРІВ У ГОТЕЛІ

Виконав
студент групи ІНб-1-22-4.0д
Грунківський Денис Васильович
_____
(підпис)
Науковий керівник
кандидат технічних наук,
доцент
Мельник І.Ю
_____
(підпис)

Київ - 2026

Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп'ютерних наук

**«Затверджую»**

Завідувач кафедри комп'ютерних наук

доктор технічних наук, професор

Андрій БОНДАРЧУК

«31» жовтня 2025 р.

# ЗАВДАННЯ НА КВАЛІФІКАЦІНУ РОБОТУ БАКАЛАВРА

«Клієнт-серверний додаток бронювання та обліку номерів у готелі»

Виконавець - студент групи ІНб-1-22-4.0д
Спеціальності 122 Комп'ютерні науки
Освітньої програми 122.00.01 Інформатика
Грунківський Денис Васильович

1. Вихідні данні: дослідження є нормативно-правова база України, наукові й технічні джерела у сферах веб-розробки, автоматизації процесів бронювання, а також методів валідації та захисту даних.
2. Основні завдання: Метою бакалаврської роботи є аналіз наукової та технічної бази зі створення клієнт-серверних вебзастосунків для автоматизації готельного бізнесу, обґрунтування методології та розробка структури дослідження. Практична частина передбачає створення системи бронювання та обліку номерного фонду з використанням технологічного стеку React, TypeScript, Node.js, Express та PostgreSQL, що включатиме модулі управління замовленнями, відгуками, технічними блокуваннями, адміністрування та аналітику. Робота завершується тестуванням функціоналу на коректність, безпеку й продуктивність, оформленням пояснювальної записки та підготовкою презентації результатів.
3. Пояснювальна записка: Обсяг – до 72 стор. формату А4 комп'ютерного набору з дотриманням вимог стандарту і методичних рекомендацій кафедри.
4. Графічні матеріали: Презентація.
5. Додатки: Структура бази даних
6. Строк подання роботи на кафедру: «01» червня 2026 р.

Науковий керівник:
кандидат технічних наук
доцент _____ Мельник І.Ю. дата

Виконавець:
_____ дата

# КАЛЕНДАРНИЙ ПЛАН

| № | Назва етапу дипломної роботи | Строк виконання етапу роботи | Примітка |
|---|---|---|---|
| 1 | Формування теми дипломної роботи бакалавра | 31.10.25 | Виконано |
| 2 | Ознайомлення з методичними рекомендаціями | 01.11.25 - 20.11.25 | Виконано |
| 3 | Складання змісту та календарного плану роботи | 21.11.25 - 20.12.25 | Виконано |
| 4 | Вибір технологій | 21.12.25 - 20.01.26 | Виконано |
| 5 | Написання реферату та вступу | 21.01.26 - 15.02.26 | Виконано |
| 6 | Написання першого розділу | 01.03.26 - 10.03.26 | Виконано |
| 7 | Написання другого розділу | 11.03.26 - 15.03.26 | Виконано |
| 8 | Реалізація основного функціоналу | 11.03.26 - 15.03.26 | Виконано |
| 9 | Написання третього розділу та висновків | 16.03.26 - 15.04.26 | Виконано |
| 10 | Виправлення зауважень | 29.04.26 - 15.05.26 | Виконано |
| 11 | Створення презентації | 16.05.26 - 12.06.26 | Виконано |
| 12 | Захист матеріалів дипломної роботи на засіданні кафедри | 12.05.26 | Виконано |
| 13 | Офіційний захист матеріалів дипломної роботи на засіданні екзаменаційної комісії | 15.06.26 | |

Науковий керівник
к.т.н, доцент
_____Мельник І.Ю
_____ дата

Виконавець

_____ Грунківський Д.В
_____ дата

# РЕФЕРАТ

Кваліфікаційна робота: 62с., 19 рис., 30 посилань.

**Актуальність**: В умовах глобальної цифровізації та інтенсивного розвитку інформаційних технологій автоматизація процесів бронювання та адміністрування стає ключовим фактором забезпечення конкурентоспроможності сучасних готельних підприємств. Значна частина вітчизняних закладів розміщення досі використовує застарілі методи ручного обліку або перебуває у повній фінансовій та операційній залежності від дорогих сторонніх платформ-агрегаторів. Це суттєво ускладнює динамічне управління номерами, обмежує пряму взаємодію з клієнтами та унеможливлює ефективний контроль поточної зайнятості. Проєктування та створення власного автономного програмного рішення дозволяє мінімізувати вплив людського фактора, оптимізувати операційні витрати підприємства, забезпечити прозорий централізований контроль замовлень та підвищити рівень інформаційної безпеки, що безпосередньо зумовлює високу актуальність розробки клієнт-серверної веб-системи з інтегрованою адміністративною панеллю.

**Об’єкт дослідження:** діяльність готельних підприємств, механізми управління даними та архітектурні рішення для створення систем підтримки прийняття рішень у готельному бізнесі.

**Предмет дослідження:** програмні методи, архітектурні патерни та технологічний стек для розробки клієнт-серверних веб-застосунків, що автоматизують процеси бронювання та адміністрування готельного фонду.

**Мета дослідження:** проєктування та практична реалізація веб-платформи для бронювання номерів, що включає інструменти для управління замовленнями, модерації контенту та візуальної аналітики діяльності готелю.

**Завдання дослідження:** Для реалізації поставленої мети передбачено виконання системного аналізу наявних комерційних аналогів, формалізацію

повного переліку функціональних і нефункціональних вимог до системи, архітектурне проєктування загальної структури застосунку та концептуальної моделі реляційної бази даних, програмну реалізацію інструментів безпечної автентифікації та авторизації з диференціацією прав доступу, розробку інтерактивного каталогу номерного фонду та спеціалізованого алгоритму валідації із технічним блокуванням дат для унеможливлення подвійного резервування, а також проведення комплексного тестування створеного програмного продукту.

**Результати роботи:** Для досягнення поставленої мети було здійснено системний аналіз існуючих комерційних аналогів, визначено повний перелік функціональних і нефункціональних вимог до системи, спроєктовано концептуальну модель реляційної бази даних та загальну архітектуру застосунку. У процесі безпосередньої розробки програмно реалізовано модулі захищеної автентифікації та авторизації користувачів із розмежуванням прав доступу, інтерактивний каталог кімнат, а також алгоритм валідації бронювань із механізмом технічного блокування дат для запобігання виникненню помилок подвійного резервування, після чого проведено комплексне тестування системи.

# ЗМІСТ

# ВСТУП

У сучасних умовах стрімкої цифровізації сфери гостинності автоматизація процесів бронювання стає критично важливим фактором конкурентоспроможності готельного бізнесу. Провідні світові та вітчизняні платформи забезпечують широкий функціонал, проте їх впровадження супроводжується високою вартістю ліцензування, складністю інтеграції та надмірною універсальністю, що ускладнює роботу малого та середнього бізнесу. Існуючі прогалини знань та ринкових пропозицій стосуються створення легковагових, модульних та адаптивних веб-рішень, які дозволяють гнучко керувати номерним фондом, уникати конфліктів дат у реальному часі, інтегрувати механізми модерації відгуків безпосередньо в життєвий цикл бронювання та візуалізувати аналітичні дані для оперативного прийняття управлінських рішень. Це зумовлює актуальність розробки власного програмного продукту, що поєднує сучасні клієнт-серверні технології, інтуїтивно зрозумілий інтерфейс та орієнтацію на реальні потреби локального готельного сегменту, дозволяючи мінімізувати залежність від дорогих сторонніх сервісів.

Об’єкт дослідження - діяльність готельних підприємств, механізми управління даними та архітектурні рішення для створення систем підтримки прийняття рішень у готельному бізнесі.

Предмет дослідження - програмні методи, архітектурні патерни та технологічний стек для розробки клієнт-серверних веб-застосунків, що автоматизують процеси бронювання та адміністрування готельного фонду.

Мета дослідження - проєктування та практична реалізація веб-платформи для бронювання номерів, що включає інструменти для управління замовленнями, модерації контенту та візуальної аналітики діяльності готелю.

Для досягнення поставленої мети необхідно розв’язати такі завдання:

1. Провести аналіз існуючих рішень у сфері автоматизації готельного бізнесу, виявити їхні переваги, недоліки та сформулювати функціональні й технічні вимоги до системи.

2. Обґрунтувати вибір технологічного стеку, спроєктувати архітектуру застосунку та структуру реляційної бази даних з урахуванням цілісності, нормалізації та ефективності запитів.

3. Реалізувати серверну частину (REST API) з модулями автентифікації, управління номерами, обробки бронювань, валідації дат та перевірки технічних блокувань.

4. Розробити клієнтську частину з інтерактивним каталогом, особистим кабінетом користувача та адміністративною панеллю, що містить інструменти дашборду, журналу подій та модерації відгуків.

Методи дослідження. У роботі використано комплекс методів: системний аналіз предметної області та програмних аналогів; об’єктно-орієнтоване проєктування (діаграми прецедентів, компонентів, ER-діаграми); методи формального моделювання бізнес-процесів бронювання; ітеративна розробка програмного забезпечення; методи тестування веб-застосунків; статистичні методи обробки даних для побудови аналітичних звітів та візуалізації показників завантаженості й доходу.

Практичне значення одержаних результатів полягає у створенні готового до впровадження програмного продукту, який може бути адаптований для використання в реальних готелях, хостелах або апартаментах. Система дозволяє мінімізувати людський фактор при прийомі замовлень, зменшити операційні витрати, підвищити швидкість обслуговування гостей та отримувати об'єктивну аналітику для оптимізації ціноутворення й управління номерним фондом.

Галузь застосування результатів роботи охоплює сферу гостинності, туристичний бізнес, а також IT-консалтинг у напрямку розробки спеціалізованих корпоративних веб-рішень.

# Розділ 1

# ТЕОРЕТИЧНІ ОСНОВИ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

## 1.1 Актуальність автоматизації процесів бронювання та обліку в готельному бізнесі

Сучасний стан сфери гостинності характеризується стрімкою цифровою трансформацією, що зумовлена зростанням попиту на онлайн-сервіси, підвищенням вимог клієнтів до швидкості та зручності бронювання, а також необхідністю оптимізації внутрішніх операційних процесів закладів розміщення. Аналіз ринкових трендів свідчить про активний перехід готелів від паперового обліку та фрагментованих інструментів до єдиних інформаційних систем управління. Попри наявність на ринку численних комерційних рішень, їхнє впровадження супроводжується низкою обмежень для малого та середнього бізнесу: висока вартість ліцензій, надмірна функціональність, складність адаптації під локальні бізнес-процеси та залежність від зовнішніх хмарних сервісів.

У вітчизняній практиці багато закладів продовжують використовувати гібридні методи обліку, поєднуючи електронні таблиці, месенджери для комунікації з гостями та окремі онлайн-форми для прийому замовлень. Такий підхід призводить до системних проблем: виникнення конфліктів дат через відсутність синхронізації в реальному часі, високий рівень помилок при ручному введенні даних, ускладнення контролю за статусами бронювань та неможливість оперативного отримання аналітичних звітів для прийняття управлінських рішень. Окрім того, відсутність централізованого механізму модерації відгуків та візуалізації ключових показників ефективності знижує якість сервісу та конкурентоспроможність закладу на цифровому ринку.

Актуальність розробки власного програмного продукту обумовлена потребою у створенні легковаго, модульного та масштабованого веб-застосунку, який поєднує сучасні клієнт-серверні технології, інтуїтивно зрозумілий інтерфейс та функціонал, орієнтований на реальні потреби готельного бізнесу. Реалізована система автоматизує ключові етапи життєвого циклу бронювання: від перегляду каталогу номерів із фільтрацією та перевірки доступності дат до формування замовлення з урахуванням технічних блокувань та подальшої модерації відгуків користувачів. Адміністративна панель забезпечує прозорий контроль за номерним фондом, оперативне

управління статусами замовлень, візуалізацію аналітики завантаженості та доходу, а також ведення детального журналу подій. Такий підхід дозволяє мінімізувати людський фактор, зменшити операційні витрати та створити єдине інформаційне середовище для взаємодії між персоналом закладу та гостями.

Таким чином, автоматизація процесів бронювання та обліку в готельному бізнесі є не лише технологічним трендом, а й стратегічною необхідністю для підвищення ефективності управління, забезпечення прозорості сервісу та адаптації до вимог цифрової економіки. Розробка власного веб-застосунку дозволяє усунути виявлені недоліки існуючих рішень, забезпечити гнучкість налаштувань під специфіку конкретного закладу та створити надійну основу для подальшого масштабування функціоналу.

## 1.2 Аналіз сучасних клієнт-серверних архітектур та підходів до розробки веб/десктоп систем

Сучасна розробка програмного забезпечення базується на принципах розподілених систем, де клієнтська та серверна частини взаємодіють через стандартизовані протоколи обміну даними. Клієнт-серверна архітектура залишається домінуючою моделлю для корпоративних та комерційних веб-застосунків завдяки масштабованості, централізованому управлінню даними та можливості незалежного оновлення компонентів.

Історично веб-системи будувалися за моделлю багатосторінкових додатків (MPA), де кожен запит користувача генерував нову HTML-сторінку на сервері. Це забезпечувало простоту індексації та сумісність, але призводило до повільної взаємодії, надмірного споживання трафіку та високого навантаження на серверну інфраструктуру. З поширенням технологій асинхронного обміну даними, а згодом і фреймворків для створення односторінкових додатків (SPA), акцент змістився на клієнтський рендеринг. SPA завантажує необхідні ресурси один раз, а подальша взаємодія відбувається через фонові запити до API, що забезпечує швидкий відгук інтерфейсу, зменшує навантаження на мережу та дозволяє реалізувати складні інтерактивні сценарії. Однак такий підхід вимагає додаткових заходів щодо управління станом застосунку, клієнтської маршрутизації та оптимізації початкового завантаження.

У контексті розробки систем бронювання доцільність вибору між веб- та десктоп-рішеннями визначається вимогами до доступності, кросплатформності та простоти підтримки. Десктоп-застосунки забезпечують високу продуктивність та глибоку інтеграцію з операційною системою, але вимагають встановлення на кожен пристрій, ускладнюють централізоване оновлення, обмежують мобільний доступ та збільшують витрати на технічну підтримку. Веб-застосунки, натомість, доступні з будь-якого пристрою з браузером, не потребують інсталяції, легко масштабуються на стороні сервера та дозволяють миттєво розгортати оновлення для всіх категорій користувачів. Для готельної системи, де клієнти взаємодіють з сервісом епізодично з різних пристроїв, а адміністратори потребують віддаленого доступу до панелі керування, веб-архітектура є найбільш оптимальним та економічно виправданим вибором.

Сучасні підходи до розробки клієнтської частини орієнтовані на компонентну архітектуру, декларативний опис інтерфейсу та строгу типізацію [11]. Компонентний підхід дозволяє інкапсулювати логіку, стилі та локальний стан у незалежні, перевикористовувані модулі, що спрощує тестування, підтримку коду та паралельну розробку. Використання мови TypeScript поверх JavaScript забезпечує статичну перевірку типів на етапі компіляції, що суттєво знижує кількість runtime-помилок, покращує автодоповнення в середовищах розробки, спрощує рефакторинг у великих проєктах та робить код самодокументованим [12]. Фреймворки на кшталт React реалізують модель віртуального DOM, що дозволяє ефективно порівнювати попередній та актуальний стан інтерфейсу і оновлювати лише змінені вузли, мінімізуючи прямі маніпуляції з реальним DOM та підвищуючи загальну продуктивність рендерингу.

Серверна частина сучасних веб-систем будується за принципом stateless-архітектури, де кожен запит містить всю необхідну інформацію для його обробки, а стан зберігається у зовнішніх сховищах (реляційні бази даних, кеш, сесійні сховища). REST API залишається найпоширенішим стандартом проєктування інтерфейсів взаємодії завдяки простоті, передбачуваності, підтримці стандартних HTTP-методів, зручності кешування та широкій інтеграції з інструментами моніторингу, логування та безпеки [2]. Альтернативні підходи, такі як GraphQL або gRPC, пропонують гнучкішу вибірку даних або високу продуктивність у розподілених мікросервісних середовищах, проте вимагають складнішої інфраструктури [4].

Платформа Node.js стала стандартом де-факто для серверної логіки веб-застосунків завдяки подієво-орієнтованій архітектурі та неблокуючій моделі вводу-виводу, що забезпечує високу пропускну здатність при обробці численних одночасних з'єднань без створення окремих потоків для кожного запиту. У поєднанні з мінімалістичним фреймворком Express, Node.js дозволяє швидко створювати маршрутизатори, middleware-компоненти для автентифікації, валідації вхідних даних, обробки файлів та централізованого керування помилками. Інтеграція з реляційними базами даних, зокрема PostgreSQL, забезпечує підтримку ACID-транзакцій, складних запитів із JOIN-ами, індексацію, розширену роботу з JSON-структурами та надійне резервне копіювання, що критично важливо для систем із суворою вимогою до цілісності фінансових та операційних даних.

Безпека та надійність архітектури забезпечуються через багаторівневу валідацію вхідних даних на клієнті та сервері, використання JWT-токенів для stateless-автентифікації, захист від типових веб-вразливостей (XSS, CSRF, SQL-ін'єкції) через параметризовані запити, санітизацію вводу та налаштування CORS-політик. Логування подій, моніторинг стану сервера та автоматизоване тестування API-ендпоінтів дозволяють підтримувати стабільність системи в умовах реального навантаження та оперативно виявляти аномалії.

Аналіз сучасних підходів свідчить, що для реалізації системи бронювання готелю оптимальною є трирівнева клієнт-серверна архітектура з чітким розділенням відповідальності між клієнтським інтерфейсом, API-шаром та сховищем даних. Використання SPA на базі React з TypeScript, серверної частини на Node.js/Express та реляційної СУБД PostgreSQL забезпечує баланс між продуктивністю, підтримуваністю, безпекою та швидкістю розробки, що повністю відповідає вимогам до сучасних веб-систем управління послугами та обґрунтовує вибір технологічного стеку в даній кваліфікаційній роботі.

### 1.3 Огляд існуючих систем бронювання готелів

Сучасний ринок програмного забезпечення для готельної індустрії умовно поділяється на кілька ключових категорій: Property Management Systems (PMS) для комплексного управління закладом, Booking Engine для прямого онлайн-бронювання, Channel Manager для синхронізації зовнішніх

каналів продажів та CRM-модулі для роботи з гостьовою базою. Провідні світові рішення, такі як Oracle Opera, Cloudbeds, Mews та SiteMinder, пропонують високий рівень автоматизації, інтеграцію з платіжними шлюзами, аналітичні дашборди та підтримку мультиканальних продажів. На вітчизняному ринку поширені платформи Bnovo, Travelline, Shelter та LocalPMS, які адаптовані до локальних вимог обліку та фіскалізації.

Таблиця 1.1

Характеристика систем бронювання та розробленого рішення

| Функціональний модуль | Комерційні PMS (Opera, Cloudbeds, Mews) | Вітчизняні системи (Bnovo, Travelline) | Розроблена веб-система |
|---|---|---|---|
| Онлайн-бронювання з валідацією дат | Так | Так | Так |
| Перевірка технічних блокувань (ремонт, резерв) | Так (окремий модуль) | Обмежено | Так (інтегровано в ядро) |
| Адміністративна панель управління номерами та статусами | Так | Так | Так |
| Модерація відгуків користувачів | Так (часто окремий сервіс) | Ні / Обмежено | Так (вбудований цикл) |
| Аналітика та візуалізація KPI у реальному часі | Так (преміум-тарифи) | Базові звіти | Так (вбудований дашборд) |
| Детальний журнал подій (логи дій адміністраторів) | Так (enterprise-рівень) | Ні | Так |

Попри функціональну насиченість комерційних продуктів, їхнє впровадження супроводжується низкою обмежень, особливо для закладів малого та середнього бізнесу. Більшість платформ працюють за моделлю SaaS із щомісячною абонентською платою, комісією за бронювання або додатковими витратами на підключення модулів аналітики, модерації відгуків та технічного блокування номерів. Крім того, закритий архітектурний підхід унеможливлює гнучку адаптацію інтерфейсу під специфічні бізнес-процеси, а залежність від зовнішньої хмарної інфраструктури створює ризики простою та обмежує контроль над даними.

Для системного порівняння існуючих рішень із розробленою веб-системою доцільно звернутися до порівняльної характеристики за ключовими функціональними критеріями..

Аналіз наведених даних свідчить, що світові та вітчизняні платформи зосереджені на масштабуванні функціоналу та інтеграції з глобальними каналами дистрибуції, що робить їх економічно недоцільними для невеликих готелів, хостелів або апартаментів. Окремі модулі, такі як перевірка технічних блокувань номерів, модерація відгуків безпосередньо в життєвому циклі бронювання або детальний журнал дій адміністраторів, часто доступні лише в корпоративних тарифах або реалізуються через сторонні інструменти, що ускладнює єдину точку управління.

До ключових обмежень існуючих рішень, які впливають на ефективність їхнього використання в умовах локального ринку, належать:

- висока сукупна вартість володіння через щомісячні підписки, комісії за транзакції та платне підключення додаткових модулів;
- відсутність вбудованого механізму перевірки технічних блокувань у базових версіях, що призводить до конфліктів дат під час ремонтних робіт або резервування номерів;
- обмежена аналітика та візуалізація ключових показників у реальному часі, що ускладнює оперативне коригування тарифів та управління завантаженістю;
- закритість архітектури, яка не дозволяє адаптувати логіку валідації, інтерфейс або інтегрувати власні бізнес-правила без залучення вендора;
- залежність від зовнішньої інфраструктури та відсутність повного контролю над безпекою та зберіганням даних.

У контексті виконання даної кваліфікаційної роботи виявлені прогалини обґрунтовують доцільність розробки власного веб-застосунку, який поєднує ключові переваги сучасних PMS-систем із мінімалістичним підходом до реалізації. Розроблена система орієнтована на забезпечення інтуїтивно зрозумілого інтерфейсу бронювання з автоматичною перевіркою доступності дат та технічних блокувань, реалізацію адміністративної панелі з візуалізацією KPI, журналом подій та інструментами модерації відгуків, а також використання сучасного стеку технологій (React, TypeScript, Node.js, PostgreSQL) для гарантії продуктивності, безпеки та масштабованості.

Таким чином, огляд існуючих систем бронювання готелів підтверджує наявність ринкового запиту на легковагове, модульне та економічно ефективне рішення, адаптоване до специфіки вітчизняного готельного бізнесу. Розробка власного веб-застосунку дозволяє усунути виявлені недоліки

комерційних продуктів, забезпечити повний контроль над даними та логікою роботи системи, а також створити технологічну основу для подальшого розширення функціоналу

### 1.4 Порівняльний аналіз аналогів: функціонал, технологічний стек, вартість, недоліки

Для обґрунтування вибору архітектурних рішень та технологічного стеку розроблюваної системи було проведено порівняльний аналіз трьох категорій існуючих програмних продуктів у сфері автоматизації готельного бізнесу:

1. Корпоративні хмарні PMS-системи світового рівня (на прикладі Cloudbeds, Mews).
2. Вітчизняні SaaS-рішення для малого та середнього бізнесу (на прикладі Bnovo, LocalPMS).
3. Розроблена власна веб-система (об'єкт дослідження даної роботи).

Аналіз проводився за чотирма ключовими критеріями: функціональна повнота, технологічна база, економічна доцільність (вартість володіння) та виявлені недоліки в контексті потреб локальних закладів розміщення.

Світові лідери ринку (Cloudbeds, Mews) пропонують екосистемний підхід, що охоплює не лише управління номерами, а й інтеграцію з сотнями каналів дистрибуції (OTA), глобальними платіжними шлюзами, системами контролю доступу (Smart Locks) та CRM-модулями. Їхній функціонал надлишковий для невеликих готелів чи апартаментів, оскільки включає складні інструменти динамічного ціноутворення на основі штучного інтелекту, управління великими мережами філій та багатомовну підтримку на рівні підприємства.

Вітчизняні аналоги (Bnovo, LocalPMS) зосереджені на базових операційних процесах: реєстрація гостей, ведення шахівниці зайнятості, формування звітів для податкової та фіскалізація. Однак вони часто мають обмежений функціонал у частині клієнтського сервісу: відсутні зручні модулі прямого онлайн-бронювання з миттєвою валідацією дат на стороні клієнта, механізми технічного блокування номерів (ремонт/санітарія) часто реалізовані через ручне введення коментарів без жорсткої програмної заборони бронювання, а модулі модерації відгуків або відсутні, або винесені в окремі платні інтеграції.

Таблиця 1.2

Порівняння технологічних стеків

| Критерій | Світові PMS (Cloudbeds, Mews) | Вітчизняні SaaS (Bnovo, LocalPMS) | Розроблена система |
|---|---|---|---|
| Архітектура | Мікросервісна, хмарна (AWS/Azure) | Монолітна або модульна, хмарна | Клієнт-серверна (SPA + REST API) |
| Frontend | Angular, React (пропрієтарні збірки) | Vue.js, jQuery (залежить від версії) | React, TypeScript |
| Backend | Java, Go, .NET Core | PHP (Laravel), Python, Node.js | Node.js, Express |
| База даних | PostgreSQL, MongoDB (шардування) | MySQL, PostgreSQL | PostgreSQL |
| Доступ до коду | Закритий (Closed Source) | Закритий (SaaS) | Відкритий (для власника) |
| Інтеграції | Широкий спектр готових API | Обмежений набір партнерських API | Гнучкий REST API для розширення |

Розроблена система проектується з акцентом на цільовий функціонал, необхідний для ефективного управління окремим закладом:

- Інтерактивне бронювання: валідація дат у реальному часі з урахуванням активних бронювань та технічних блокувань.
- Управління статусами: гнучкий цикл життя бронювання (pending → confirmed → checked_in → cancelled).
- Модерація відгуків: вбудований цикл залишення та схвалення відгуків без сторонніх сервісів.
- Аналітика: візуалізація KPI (завантаженість, дохід) у режимі реального часу.

Технологічний вибір безпосередньо впливає на продуктивність, безпеку та можливість масштабування системи.

Світові та вітчизняні рішення використовують потужні, але складні в підтримці стеки, оптимізовані для тисяч одночасних користувачів. Розроблена система обирає сучасний, але легковагий стек: React забезпечує швидкий інтерфейс завдяки віртуальному DOM та компонентному підходу; TypeScript гарантує типову безпеку коду, знижуючи кількість runtime-помилок; Node.js

дозволяє ефективно обробляти велику кількість одночасних запитів до API завдяки неблокуючій моделі введення-виведення; PostgreSQL надає надійне зберігання структурованих даних з підтримкою складних транзакцій та JSON-операцій, що є критичним для фінансових операцій бронювання.

Економічна складова є визначальним фактором для малого та середнього бізнесу.

● Світові PMS: Вимагають значних початкових інвестицій та щомісячної абонентської плати, яка часто залежить від кількості номерів або підключених каналів продажів. Додаткові модулі (онлайн-бронювання, аналітика) часто оплачуються окремо. Загальна вартість володіння (TCO) може сягати тисяч доларів на рік.

● Вітчизняні SaaS: Пропонують більш доступні тарифи, проте також працюють за моделлю підписки. Часто присутні приховані витрати: комісія за кожне бронювання, плата за технічну підтримку, додаткова оплата за навчання персоналу.

● Розроблена система: Реалізована як власна розробка, що усуває необхідність щомісячних ліцензійних платежів. Основні витрати зводяться до одноразових витрат на розробку та мінімальних експлуатаційних витрат на хостинг сервера та доменне ім'я. Це забезпечує значну економію коштів у довгостроковій перспективі та повну незалежність від політики ціноутворення сторонніх вендорів.

Під час порівняльного аналізу було виявлено ряд системних недоліків існуючих аналогів, які обґрунтовують необхідність розробки власного продукту:

1. Надлишковість функціоналу: Великі PMS-системи перевантажені інструментами, непотрібними для невеликих закладів, що ускладнює інтерфейс та збільшує час навчання персоналу.

2. Залежність від зовнішньої інфраструктури: Хмарні рішення повністю залежать від стабільності інтернет-з'єднання та доступності серверів провайдера. У разі аварії на стороні постачальника послуг робота закладу паралізується.

3. Обмежена гнучкість: Неможливість адаптації логіки валідації бронювань під специфічні бізнес-правила конкретного готелю без звернення до технічної підтримки вендора та оплати додаткових робіт.

4. Відсутність комплексної аналітики в базових тарифах: Детальні звіти щодо завантаженості, джерел трафіку та поведінки гостей часто доступні лише у преміум-пакетах.

5. Проблеми з локалізацією та підтримкою: Світові рішення можуть мати неповний переклад або специфіку, не адаптовану до українського законодавства та менталітету користувачів.

### 1.5 Формування вимог до розроблюваного додатку

На основі аналізу існуючих рішень та специфіки цільової аудиторії сформовано комплекс функціональних вимог, що чітко розмежовують можливості звичайного користувача та адміністратора. Для гостей системи передбачено реєстрацію та авторизацію через електронну пошту з отриманням JWT-токена, перегляд каталогу номерів із фільтрацією за типом, ціною та назвою, а також детальну інформацію про кожен номер, включаючи фотографії, зручності та правила проживання. Ключовим елементом є інтерактивна форма бронювання з автоматичним розрахунком вартості та перевіркою доступності дат з урахуванням активних замовлень і технічних блокувань. Особистий кабінет дозволяє відстежувати статуси бронювань, скасовувати активні резервації до дати заїзду, редагувати профіль та залишати відгуки з оцінкою після підтвердження статусу заселення, які автоматично надходять на модерацію.

Функціонал адміністративної панелі орієнтований на повний контроль над операційними процесами закладу та включає авторизацію з підвищеними правами доступу до захищеного інтерфейсу керування. Адміністратору доступний аналітичний дашборд із візуалізацією ключових показників ефективності, таких як рівень завантаженості, дохід за обраний період, кількість активних та очікувальних бронювань, середній рейтинг і загальна кількість відгуків, а також графічне відображення динаміки за місяцями. Система забезпечує управління номерним фондом через створення, редагування та м’яку архівацію записів із підтримкою завантаження фотографій, а також інструменти для зміни статусів бронювань, перегляду деталей замовлень і коментарів. Окремим модулем реалізовано механізм технічного блокування номерів із вказанням причини та журнал подій, що фіксує останні дії в системі з детальним логуванням часу, сум та імен

користувачів, а також цикл модерації відгуків із можливістю їх схвалення або видалення.

Нефункціональні вимоги визначають архітектурні обмеження та експлуатаційні характеристики розроблюваного продукту. Клієнтська частина реалізована як односторінковий додаток, що забезпечує миттєве перемикання між маршрутами без перезавантаження сторінки, а час відповіді серверного інтерфейсу на стандартні запити не перевищує трьохсот мілісекунд у локальному середовищі. Надійність та цілісність даних гарантується застосуванням типів діапазонів дат на рівні бази даних для унеможливлення конфліктів інтервалів, а також використанням параметризованих запитів для виключення порушення зв’язків між таблицями. Архітектура розділена на типізовану клієнтську та серверну частини з чіткою структурою маршрутизаторів, контролерів та сервісів, що спрощує рефакторинг і масштабування. Поточна версія системи функціонує в режимі резервування місць без інтеграції платіжних шлюзів, сповіщень та мобільного клієнта, що визначає межі її застосування на даному етапі розробки.

Забезпечення безпеки та зручності взаємодії реалізовано через багаторівневу модель контролю доступу та адаптивний інтерфейс. Аутентифікація базується на JWT-токенах, що передаються в заголовку авторизації, а розмежування прав виконується на рівні серверного проміжного програмного забезпечення з обмеженням доступу до адміністративних маршрутів виключно для відповідної ролі. Валідація всіх вхідних даних здійснюється за допомогою декларативних схем на стороні сервера, а захист від веб-вразливостей включає параметризовані запити для запобігання ін’єкціям, налаштування політики міждоменних запитів та перевірку типів завантажуваних файлів. Інтерфейс коректно адаптується під екрани різної роздільної здатності, використовує вкладкову навігацію та супроводжує кожну дію модальними вікнами і спливаючими сповіщеннями. Форми містять інтерактивну перевірку коректності введення, блокування кнопок заповнення та обмеження вибору дат, а статуси замовлень і активність номерів візуалізуються за допомогою кольорових індикаторів для миттєвого сприйняття інформації.

## Висновки до розділу 1

У першому розділі кваліфікаційної роботи проведено комплексне дослідження теоретичних основ та аналіз існуючих рішень у сфері автоматизації процесів бронювання готельних послуг. На основі викладеного матеріалу сформовано такі висновки:

1. Цифрова трансформація сфери гостинності об'єктивно зумовлює необхідність впровадження автоматизованих систем управління бронюванням. Аналіз ринкових тенденцій підтверджує, що заклади розміщення, які використовують ручні методи обліку або фрагментовані інструменти, зазнають системних проблем: конфлікти дат, високий рівень помилок при введенні даних, ускладнення контролю за статусами замовлень та відсутність оперативної аналітики для прийняття управлінських рішень.

2. Сучасна розробка веб-систем базується на клієнт-серверній архітектурі з чітким розділенням відповідальності між інтерфейсом, API-шаром та сховищем даних [3]. Для реалізації системи бронювання оптимальним є підхід односторінкового додатка (SPA) у поєднанні з REST API, що забезпечує високу швидкість відгуку інтерфейсу, ефективне кешування та незалежне масштабування компонентів. Використання компонентної архітектури, статичної типізації та неблокуючої моделі обробки запитів дозволяє створювати надійні, підтримувані та продуктивні рішення.

3. Ринок систем бронювання готелів сегментований на корпоративні PMS-платформи світового рівня та вітчизняні SaaS-рішення для малого бізнесу. Проведений порівняльний аналіз виявив, що світові лідери пропонують надмірно широкий функціонал із високою вартістю володіння, тоді як локальні аналоги часто обмежуються базовими операціями без глибокої аналітики, вбудованих інструментів модерації відгуків та детального журналу подій. Обидві категорії мають спільні обмеження: залежність від зовнішньої інфраструктури, закритість коду та обмежену гнучкість адаптації.

4. На основі критичного аналізу існуючих рішень сформовано комплекс функціональних і нефункціональних вимог до розроблюваного веб-застосунку. Функціональні вимоги чітко розмежовують можливості користувача (перегляд каталогу, бронювання, особистий кабінет, залишення відгуків) та адміністратора (управління номерами, статусами бронювань, модерація відгуків, аналітика, журнал подій). Нефункціональні вимоги

орієнтовані на продуктивність, безпеку даних, цілісність транзакцій та зручність подальшої підтримки кодової бази.

5. Обґрунтовано вибір технологічного стеку для реалізації системи: React та TypeScript для клієнтської частини забезпечують компонентну архітектуру, типову безпеку та ефективний рендеринг; Node.js та Express для серверної частини дозволяють ефективно обробляти численні одночасні запити; PostgreSQL як реляційна СУБД гарантує цілісність даних, підтримку складних запитів та ACID-транзакцій, що є критичним для фінансових операцій бронювання.

6. Виявлені прогалини існуючих рішень та сформовані вимоги безпосередньо вплинули на архітектурні рішення розроблюваної системи: інтеграція перевірки технічних блокувань у ядро валідації бронювань, вбудований цикл модерації відгуків, візуалізація KPI у реальному часі та детальний журнал подій для адміністраторів. Такий підхід дозволяє усунути недоліки комерційних аналогів та забезпечити економічно ефективне рішення для малого та середнього готельного бізнесу.

Отримані результати першого розділу створюють теоретичне та методологічне підґрунтя для переходу до практичної реалізації системи. У наступному розділі буде обґрунтовано вибір конкретних методів та інструментів розробки, спроектовано структуру бази даних та реалізовано ключові модулі веб-застосунку відповідно до сформованих вимог.

# Розділ 2

# ПРОЄКТУВАННЯ ТА МЕТОДОЛОГІЯ РОЗРОБКИ СИСТЕМИ

## 2.1 Обґрунтування вибору технологічного стеку

На основі функціональних і нефункціональних вимог, сформованих у першому розділі, було проведено системний вибір технологічного стеку для реалізації клієнт-серверного додатку бронювання та обліку номерів у готелі. Кожна технологія обиралася за критеріями продуктивності, безпеки, підтримуваності кодової бази, відповідності сучасним стандартам веб-розробки та здатності забезпечити цілісність транзакційних даних.

Клієнтська частина реалізована на базі бібліотеки React у поєднанні з мовою TypeScript [13]. React забезпечує компонентну архітектуру, віртуальний DOM та декларативний підхід до побудови інтерфейсу, що є критично важливим для створення інтерактивних форм бронювання, динамічної фільтрації каталогу та миттєвого оновлення стану без перезавантаження сторінки. Використання TypeScript дозволяє впровадити статичну типізацію на етапі компіляції, що суттєво знижує кількість помилок під час виконання, покращує автодоповнення в середовищах розробки, спрощує рефакторинг великих модулів та робить код самодокументованим. Для клієнтської маршрутизації застосовано React Router, який забезпечує навігацію між публічним каталогом, особистим кабінетом та адміністративною панеллю без повних оновлень сторінки [14]. Стилізація інтерфейсу реалізована за допомогою Tailwind CSS, що дозволяє швидко будувати адаптивну верстку з мінімальним обсягом кастомного CSS та забезпечує узгоджений дизайн-системний підхід [15].

Серверна частина побудована на платформі Node.js із використанням фреймворку Express [16]. Node.js обрано через його подієво-орієнтовану архітектуру та неблокуючу модель вводу-виводу, яка забезпечує високу пропускну здатність при обробці численних одночасних HTTP-запитів без створення окремих потоків. Express, як мінімалістичний фреймворк, дозволяє швидко створювати REST API-ендпоінти, реалізувати middleware-компоненти для автентифікації, валідації вхідних даних, обробки файлів та централізованого керування помилками [17]. Для управління сесіями та захищеним доступом до ресурсів використано механізм JWT (JSON Web Tokens), який забезпечує stateless-автентифікацію, не вимагає зберігання стану

на сервері та легко масштабується у розподілених системах. Валідація всіх вхідних даних реалізована за допомогою бібліотеки Zod, що дозволяє декларативно описувати схеми даних, автоматично генерувати повідомлення про помилки та унеможливлює передачу некоректних типів на рівень бізнес-логіки [19]. Завантаження фотографій номерів забезпечує бібліотека Multer, яка обробляє multipart/form-data дані, валідує типи файлів та зберігає їх у ізольованій директорії з унікальними іменами [18].

Система керування базами даних обрана PostgreSQL [6]. Це відкрита об'єктно-реляційна СУБД, яка підтримує стандарти ACID, складні транзакції, індексацію та тип daterange, критично важливий для перевірки перекриття інтервалів дат при бронюванні. На відміну від документо-орієнтованих баз даних, PostgreSQL гарантує строгу цілісність зв'язків між таблицями users, rooms, bookings, reviews та blocked_dates за допомогою зовнішніх ключів, що є обов'язковою вимогою для операційних систем бронювання. Порівняно з MySQL, PostgreSQL пропонує більш розширені можливості для аналітичних запитів, підтримку користувальницьких типів даних та кращу оптимізацію складних JOIN-операцій, необхідних для формування дашбордів та журналів подій.

Інструменти розробки включають збірку на базі Vite, що забезпечує швидку гарячу заміну модулів (HMR) та оптимізовану компіляцію для продакшену. Лінтинг та форматування коду налаштовані через ESLint та Prettier для підтримки єдиного стилю та виявлення потенційних помилок на ранніх етапах. Керування залежностями здійснюється через npm, що забезпечує детерміновану установку пакетів.

Порівняльний аналіз альтернатив підтверджує доцільність обраного стеку. Фреймворки на кшталт Angular або Vue.js також підходять для SPA, проте Angular має вищий поріг входження та надмірну складність для проєкту середнього масштабу, тоді як Vue.js не обрано через пріоритет екосистеми React для інтеграції з бібліотеками візуалізації та уніфікації підходу з серверною частиною на JavaScript/TypeScript. Серверні фреймворки Django (Python) або Laravel (PHP) пропонують вбудовані ORM та адміністративні панелі, проте вимагають синхронної або багатопотокової моделі обробки запитів, що менш ефективно для високонавантажених API, а також ускладнюють уніфікацію мови програмування між клієнтом і сервером. Обрання єдиної мови (TypeScript/JavaScript) для повного стеку дозволяє

переиспользувати типи, зменшити когнітивне навантаження та прискорити цикл розробки.

Таким чином, технологічний стек React + TypeScript, Node.js + Express, PostgreSQL та супутні інструменти повністю відповідають вимогам до продуктивності, безпеки, масштабованості та підтримуваності системи. Архітектурний підхід забезпечує чітке розділення відповідальності, типову безпеку коду, надійне зберігання транзакційних даних та гнучкість для подальшого розширення функціоналу без необхідності рефакторингу ядра системи.

## 2.2 Архітектурне проєктування системи

Архітектура розробленого веб-застосунку побудована за принципом трирівневої клієнт-серверної моделі з чітким розділенням відповідальності між шарами. Такий підхід забезпечує модульність, спрощує тестування та підтримку коду, дозволяє незалежно масштабувати компоненти та мінімізує вплив змін у одному шарі на інші. Проєктування здійснювалося з урахуванням вимог до продуктивності, безпеки даних, зручності супроводу та можливості подальшого розширення функціоналу.

Система складається з трьох логічних рівнів:

- Презентаційний рівень (Frontend) – відповідає за відображення інтерфейсу, обробку дій користувача, клієнтську валідацію та ініціювання HTTP-запитів.
- Рівень бізнес-логіки (Backend API) – виконує автентифікацію, валідацію вхідних даних, застосування бізнес-правил, управління транзакціями та формування структурованих відповідей.
- Рівень даних (Database) – забезпечує надійне зберігання структурованої інформації, підтримку цілісності зв'язків, виконання складних агрегаційних запитів та резервне копіювання.

Клієнтська частина реалізована як односторінковий додаток (SPA) на базі React із використанням TypeScript. Архітектурні особливості:

- Компонентна модель – кожен елемент інтерфейсу (картки номерів, форми бронювання, адміністративні таблиці, модальні вікна, сповіщення) є ізольованим, перевикористовуваним модулем із власним станом та логікою рендерингу.

● Маршрутизація – реалізована через React Router, що забезпечує навігацію між публічним каталогом, особистим кабінетом та адміністративною панеллю без повного перезавантаження сторінки.

● Управління станом – локальний стан компонентів обробляється через React Hooks (useState, useEffect). Глобальний стан мінімізовано завдяки централізованій обробці відповідей API та миттєвому оновленню інтерфейсу після успішних мутацій.

● Типізація – усі відповіді сервера, пропси компонентів та форми даних описані через TypeScript-інтерфейси (User, Booking, Room, Review, ApiResponse), що унеможливлює помилки типізації на етапі виконання.

● Обробка помилок – глобальні перехоплювачі помилок мережі, індикатори завантаження (Loader2) та спливаючі сповіщення (Toast) забезпечують зворотний зв'язок під час асинхронних операцій.

● Серверна логіка побудована на Node.js та Express з використанням архітектурного патерну «Маршрутизатор → Контролер → Сервіс → Рівень доступу до даних» [9].

● Маршрутизатори (routes/) – визначають URI-ендпоінти, HTTP-методи та ланцюжок middleware для кожного шляху.

● Контролери (controllers/) – обробляють HTTP-запити, викликають відповідні сервіси, формують уніфіковані JSON-відповіді та передають помилки в глобальний обробник.

● Сервіси (services/) – містять бізнес-логіку: перевірку доступності номерів, розрахунок вартості бронювання, валідацію перекриття дат, зміну статусів, агрегацію даних для дашборду.

● Middleware – забезпечують крос-функціональну логіку: перевірку CORS, парсинг тіла запиту, валідацію через Zod, автентифікацію через JWT, перевірку ролей (user/admin), обробку файлів (Multer) та централізоване управління помилками.

● Stateless-підхід – кожен запит містить усю необхідну контекстну інформацію. Стан не зберігається на сервері, що спрощує масштабування та балансування навантаження.

Сховище даних реалізовано на PostgreSQL. Логічна модель нормалізована до третьої нормальної форми (3NF) та включає такі сутності:

- users – облікові записи (id, full_name, email, password_hash, role, phone, created_at)

- rooms – каталог номерів (id, room_number, name, type, price_per_night, description, photo_url, amenities, rules, is_active)
- bookings – бронювання (id, user_id, room_id, check_in, check_out, total_price, status, comment, created_at, updated_at)
- reviews – відгуки (id, user_id, room_id, rating, title, comment, is_approved, created_at)
- blocked_dates – технічні блокування (id, room_id, start_date, end_date, reason, comment)

Зв'язки та обмеження:

- users 1:N bookings, rooms 1:N bookings, rooms 1:N reviews, rooms 1:N blocked_dates
- Зовнішні ключі гарантують посилальну цілісність. Видалення номерів реалізовано м'яко (is_active = false), що зберігає історію бронювань.
- Для перевірки перекриття дат використовується тип daterange та оператор &&, що забезпечує атомарність та уникнення станів race condition при одночасних запитах на бронювання [7].

Архітектура безпеки та контролю доступу

- Автентифікація: JWT-токени з обмеженим терміном дії. Токен генерується після успішного входу, підписується секретним ключем на сервері та передається в заголовку Authorization.
- Авторизація: Middleware перевіряє роль користувача. Маршрути /api/admin/* доступні лише для role: 'admin'.
- Захист даних: Параметризовані запити виключають SQL-ін'єкції. Валідація Zod блокує передачу некоректних типів. CORS обмежує доступ до API тільки з дозволених джерел. Файли зберігаються в ізольованій директорії з унікальними іменами та перевіркою MIME-типів.
- Обробка помилок: Глобальний errorHandler перехоплює синхронні та асинхронні помилки, логує їх у консоль сервера та повертає клієнту безпечне повідомлення без розкриття внутрішньої структури системи.

Потік даних та взаємодія компонентів

1. Користувач виконує дію в інтерфейсі → React формує запит із заголовком авторизації.
2. Запит надходить на Express-сервер → проходить ланцюжок middleware (CORS → body parser → Zod validation → JWT verify → role check).

3. Контролер делегує запит до сервісу → сервіс виконує бізнес-логіку та звертається до PostgreSQL через connection pool.

4. Результат повертається через сервіс → контролер → формується JSON-відповідь.

5. Клієнт оновлює локальний стан, показує toast-сповіщення та перенаправляє користувача або оновлює відображення.

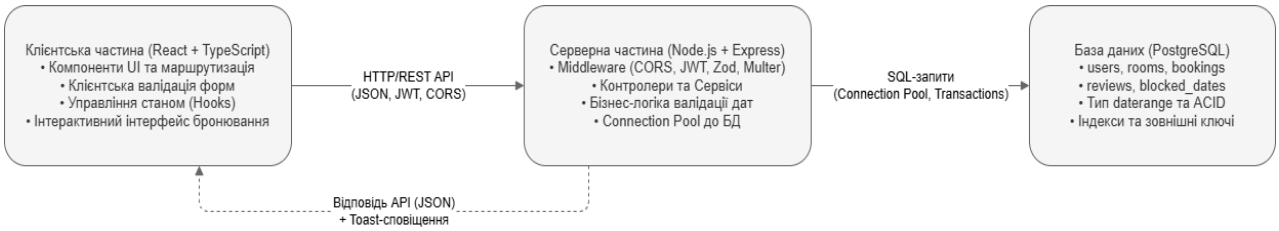


Рисунок 2.1. Архітектура клієнт-серверної системи

Обрана архітектура забезпечує високу ступінь модульності, безпеки та підтримуваності. Чітке розділення шарів дозволяє паралельну розробку клієнтської та серверної частин, спрощує інтеграційне тестування та закладає основу для майбутнього розширення функціоналу (підключення платіжних шлюзів, email/SMS-сповіщення, мобільний клієнт) без необхідності рефакторингу ядра системи. Використання TypeScript на обох кінцях стеку, патерну маршрутизатор-контролер-сервіс та реляційної моделі з daterange гарантує типову безпеку, передбачуваність поведінки та цілісність транзакційних даних у реальному часі.

## 2.3 Інформаційне проєктування: концептуальна та логічна моделі бази даних

Проєктування інформаційної моделі виконувалося послідовно: від виділення ключових сутностей предметної області до побудови нормалізованої реляційної схеми, оптимізованої під специфіку PostgreSQL. Метою даного етапу було забезпечення структурної цілісності даних, підтримка складних бізнес-правил валідації бронювань та створення передумов для ефективної аналітики і масштабування системи.

Концептуальна модель відображає предметну область на рівні бізнес-сутностей та зв'язків між ними, без урахування деталей реалізації в конкретній

системі керування базами даних. На основі функціональних вимог виділено п'ять основних сутностей:

- User (Користувач) – облікові записи клієнтів та адміністраторів із рольовою моделлю доступу.
- Room (Номер) – каталог номерного фонду з технічними характеристиками, зручностями, правилами проживання та прапорцем активності.
- Booking (Бронювання) – фіксує замовлення, зв'язуючи користувача з номером, містить дати заїзду/виїзду, розрахункову вартість, статус та коментар.
- Review (Відгук) – містить оцінку, текст та статус модерації, прив'язаний до конкретного номера та користувача.
- BlockedDate (Технічне блокування) – реєструє періоди недоступності номерів з вказанням причини, використовується для запобігання бронюванням під час ремонту чи резервування.

Зв'язки між сутностями реалізовано як 1:N (один-до-багатьох), що виключає необхідність у проміжних таблицях та спрощує логіку агрегації даних. Концептуальна модель враховує ключові бізнес-правила: неперетинність інтервалів дат для активних бронювань та блокувань, обов'язкова модерація відгуків перед публікацією, м'яке видалення номерів (is_active = false) для збереження історичних даних.

Графічне подання концептуальної моделі у вигляді ER-діаграми наведено на рисунку 2.2.

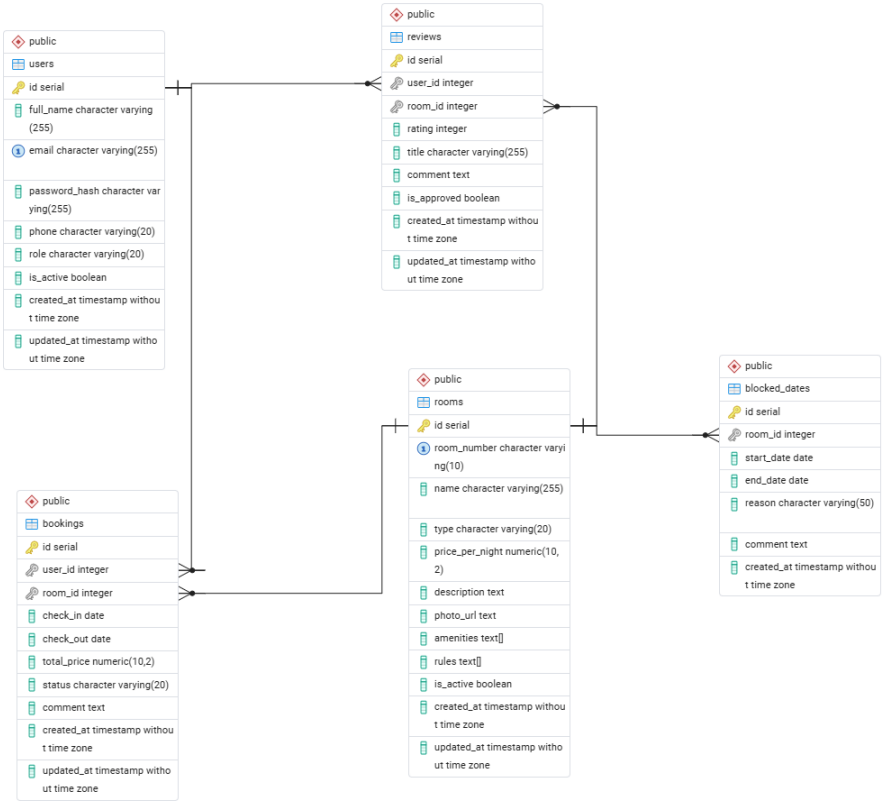


Рисунок 2.2. Логічна моделі бази даних

На етапі логічного проєктування концептуальна модель трансформована у реляційну схему з дотриманням третьої нормальної форми (3NF) [5]. Кожна таблиця містить строго типізовані поля, первинні та зовнішні ключі, обмеження цілісності та індекси для прискорення вибірок. Логічна схема повністю відповідає фізично реалізованій структурі в PostgreSQL та узгоджена з TypeScript-інтерфейсами клієнтської частини.

Детальну структуру таблиць, типи даних та зв'язки між ними у логічній моделі представлено на рисунку 2.3.

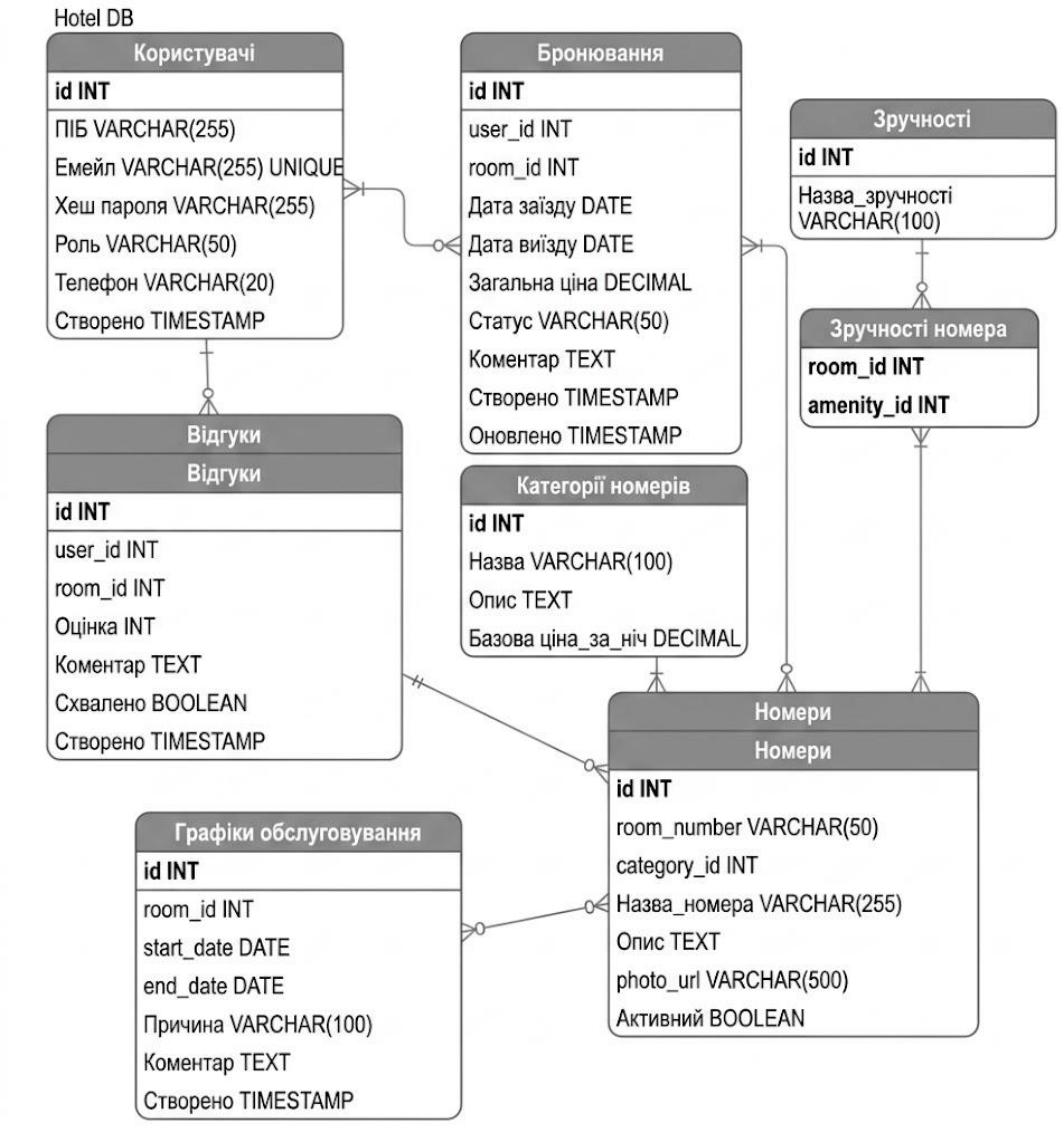


Рисунок 2.3. Логічна модель реляційної бази даних

Логічна модель спроектована з дотриманням третьої нормальної форми (3NF), що виключає транзитивні залежності, повторювані групи та аномалії оновлення чи видалення даних. Для збереження складних атрибутів, таких як перелік зручностей та правила проживання, використано нативний тип TEXT[] PostgreSQL, що дозволяє уникнути зайвих таблиць-довідників без порушення нормалізації. Цілісність інформації гарантується зовнішніми ключами: правило ON DELETE RESTRICT захищає записи користувачів та номерів від випадкового видалення за наявності пов'язаних бронювань або відгуків, тоді як ON DELETE CASCADE застосовано для технічних блокувань, оскільки їхня історія втрачає актуальність після видалення номера. Додатково реалізовано механізм м'якого видалення через поле is_active у таблиці rooms,

що дозволяє тимчасово виводити номери з публічного каталогу без втрати історичних даних про замовлення та аналітики.

Для забезпечення коректності бізнес-процесів та високої продуктивності системи передбачено атомарну перевірку неперетинності дат за допомогою типу daterange та оператора перетину &&, що повністю виключає ризик конфлікту інтервалів або стану гонки даних при одночасних запитах на бронювання. Оптимізація швидкості вибірок досягається за рахунок створення індексів на ключових полях user_id, room_id, status, check_in, check_out та is_approved, що прискорює формування звітів дашборду, пошук в особистому кабінеті та фільтрацію публічного каталогу. Загалом наведена логічна модель точно відповідає фізично реалізованій структурі бази даних PostgreSQL, забезпечує надійне зберігання транзакційних даних, підтримує складні аналітичні агрегації та створює стабільну технологічну основу для майбутнього розширення функціоналу без необхідності рефакторингу ядра системи.

### 2.4 Моделювання бізнес-процесів та алгоритмів

Для забезпечення коректної роботи системи бронювання необхідно чітко визначити основні бізнес-процеси та алгоритми їх виконання. Моделювання здійснювалося з використанням нотації BPMN 2.0 (Business Process Model and Notation) для візуалізації бізнес-процесів та блок-схем алгоритмів згідно з ДСТУ ISO 5807:2016 [24]. У розробленій системі виділено три ключові бізнес-процеси: процес бронювання номеру користувачем, що включає пошук доступних номерів, перевірку дат, розрахунок вартості та створення запису про бронювання; процес обробки бронювання адміністратором, який охоплює перегляд заявок, підтвердження або скасування бронювань та зміну статусів; а також процес модерації відгуків, що включає подання відгуку користувачем, перевірку адміністратором та публікацію.

Одним із найважливіших алгоритмів системи є перевірка доступності номеру на обрані дати, яка повинна враховувати наявні активні бронювання, технічні блокування (ремонт, резерв) та коректність введених дат. Логіка перевірки реалізована на рівні бази даних з використанням типу daterange та оператора перетину &&, що забезпечує атомарність операції та уникнення

стану гонки даних (race condition) при одночасних запитах від кількох користувачів.

Блок-схема алгоритму перевірки доступності номеру наведена на рисунку 2.4.

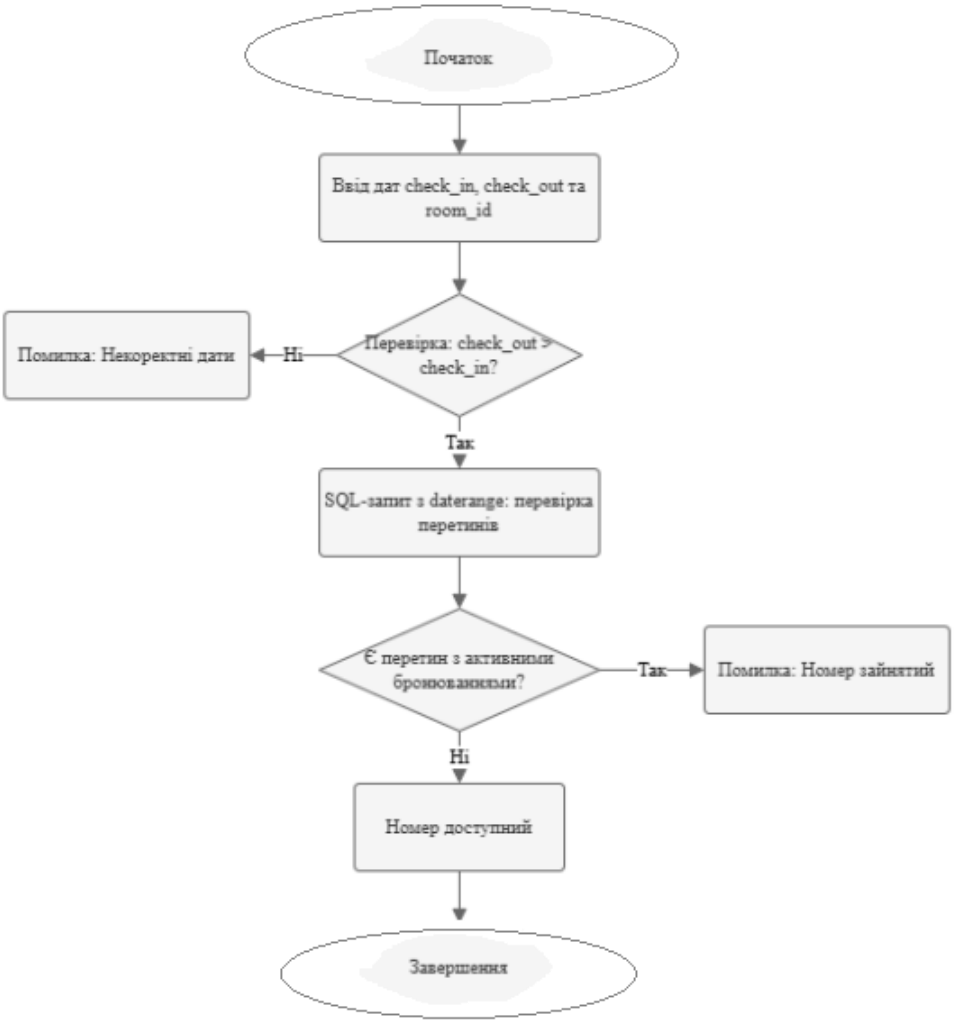


Рисунок 2.4.Алгоритм перевірки доступності номеру

Процес створення бронювання включає кілька етапів:

1. Авторизація користувача (перевірка наявності JWT-токена).
2. Валідація вхідних даних (дати, ID номеру) за допомогою Zod-схеми.
3. Перевірка доступності номеру (виклик алгоритму з рис. 2.4).
4. Розрахунок загальної вартості (кількість днів × ціна за ніч).
5. Створення запису в таблиці bookings зі статусом pending.
6. Повернення успішної відповіді клієнту.

Послідовність дій при створенні бронювання зображена на діаграмі послідовності (рис. 2.5).

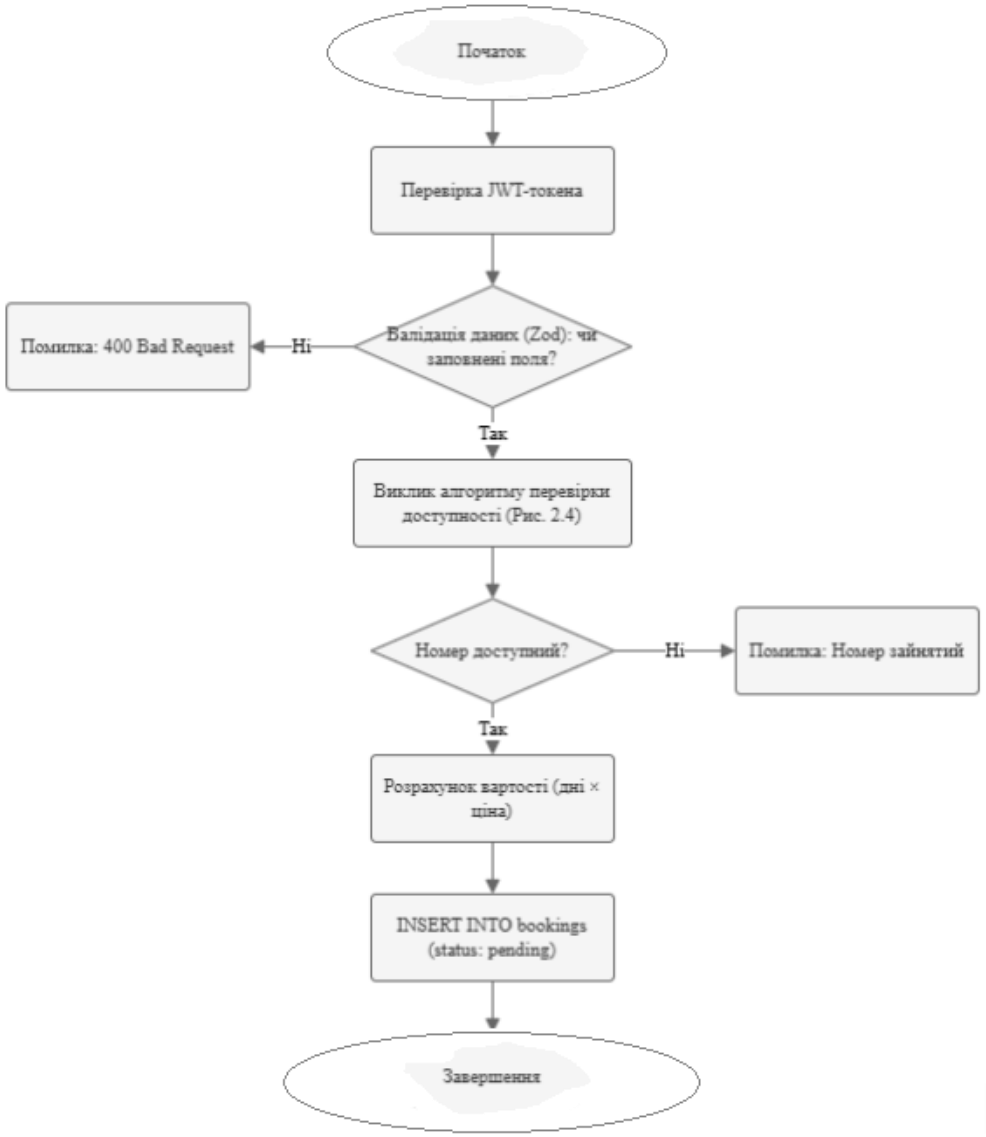


Рисунок 2.5. Діаграма послідовності створення бронювання

Адміністратор має можливість змінювати статус бронювання відповідно до життєвого циклу замовлення:

- pending → confirmed (підтвердження бронювання);
- confirmed → checked_in (заселення гостя);
- checked_in → cancelled (скасування після заселення – рідкісний випадок);
- pending/confirmed → cancelled (скасування до заїзду).
- При зміні статусу система виконує додаткові перевірки:
- заборона скасування, якщо дата заїзду вже минула;
- автоматичне оновлення поля updated_at;
- логування події в журнал адміністратора.

Блок-схема алгоритму зміни статусу бронювання наведена на рисунку 2.6.


Рисунок 2.6. Алгоритм зміни статусу бронювання

Процес модерації відгуків забезпечує контроль якості контенту та запобігає публікації небажаних відгуків. Алгоритм включає:

1. Користувач залишає відгук (доступно тільки при статусі checked_in).
2. Відгук зберігається зі статусом is_approved = false.
3. Адміністратор переглядає відгуки в адмін-панелі.
4. Приймає рішення: схвалити (is_approved = true) або відхилити (видалити).
5. Схвалені відгуки відображаються у публічному каталозі.

Діаграма діяльності (activity diagram) процесу модерації відгуків зображена на рисунку 2.7.

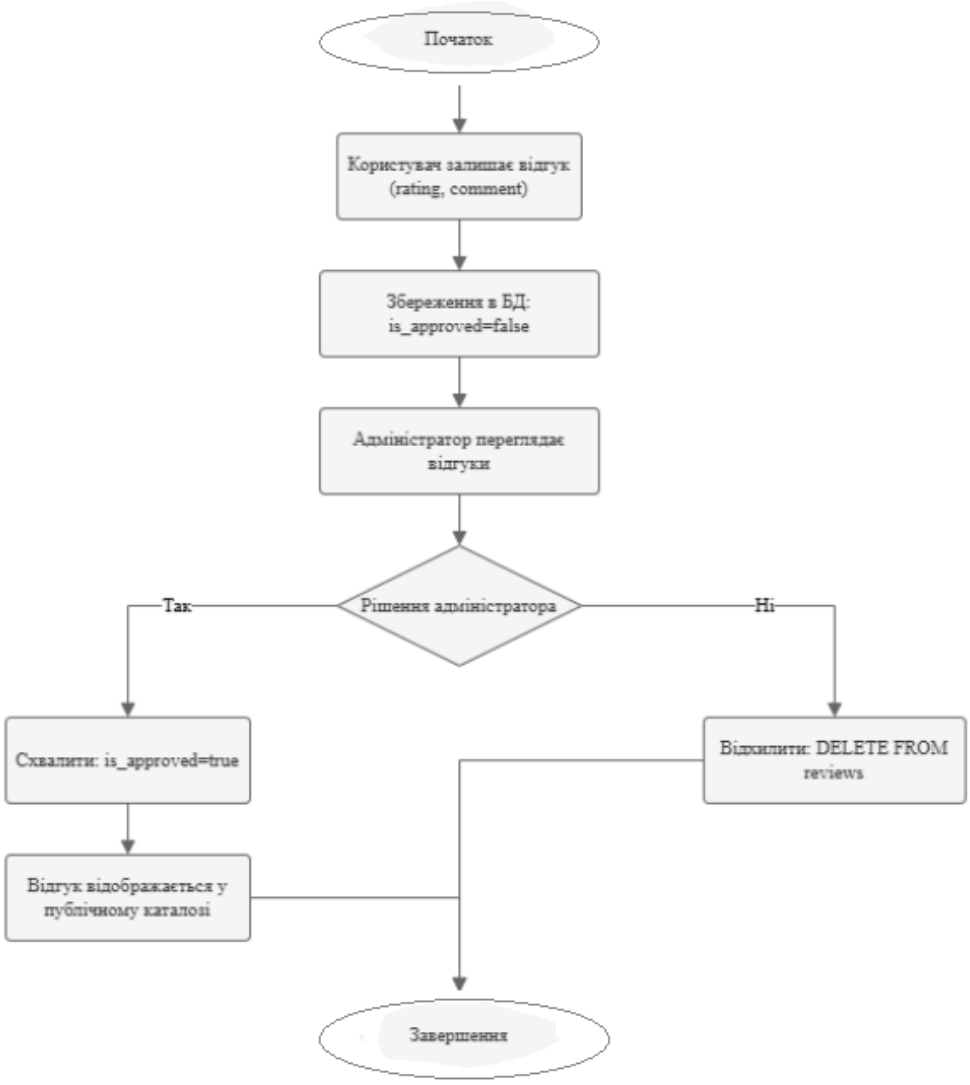


Рисунок 2.7. Діаграма діяльності процесу модерації відгуків

Моделювання бізнес-процесів за допомогою нотації BPMN 2.0 дозволило чітко визначити основні сценарії використання системи, виявити точки прийняття рішень та можливі виключні ситуації.

1. Алгоритми перевірки доступності номеру, створення бронювання, зміни статусів та модерації відгуків реалізовані з урахуванням вимог до цілісності даних, безпеки та продуктивності.

2. Використання типів daterange та операторів перетину на рівні PostgreSQL забезпечує атомарність перевірок та уникнення конфліктів при одночасних запитах.

3. Діаграми послідовності та діяльності візуалізують взаємодію між компонентами системи (клієнт → API → база даних) та спрощують подальшу підтримку коду.

## 2.5 Методика тестування та оцінки достовірності результатів

Методика тестування розробленого веб-застосунку базується на системному підході до перевірки коректності функціонування програмних модулів та оцінки достовірності отриманих результатів. Метою даного етапу є обґрунтування вибору методів верифікації та валідації, що забезпечують підтвердження відповідності системи вимогам технічного завдання.

- Для оцінки якості розробленого рішення застосовано комбіновану стратегію, що включає: статичний аналіз коду, модульне тестування, інтеграційне тестування та валідацію бізнес-логіки [10].
- Статичний аналіз коду – перевірка синтаксичної коректності, дотримання стандартів кодування та типізації за допомогою інструментів TypeScript compiler та ESLint. Цей метод дозволяє виявити потенційні помилки на етапі розробки без виконання програми.
- Модульне тестування (unit testing) – перевірка ізольованих функцій та компонентів (валідатори форм, розрахунок вартості бронювання, логіка зміни статусів) на коректність обробки вхідних даних та повернення очікуваних результатів.
- Інтеграційне тестування – перевірка взаємодії між клієнтською частиною (React), серверним API (Node.js/Express) та базою даних (PostgreSQL) з метою підтвердження цілісності передачі даних та коректності виконання транзакцій.
- Валідація бізнес-логіки – перевірка реалізації ключових правил предметної області: неперетинність інтервалів дат бронювання, врахування технічних блокувань, обмеження доступу за ролями користувачів.

Достовірність результатів роботи системи оцінюється за наступними критеріями:

Достовірність отриманих результатів забезпечується:

1. Використанням типізованих інтерфейсів (TypeScript) на клієнтській та серверній сторонах, що мінімізує помилки обробки даних.
2. Застосуванням параметризованих запитів до бази даних, що виключає можливість SQL-ін'єкцій та забезпечує коректність виконання операцій.
3. Реалізацією атомарних транзакцій у PostgreSQL для критичних операцій (створення бронювання, зміна статусу), що гарантує дотримання властивостей ACID.

4. Валідацією вхідних даних на двох рівнях (клієнт та сервер) з використанням бібліотеки Zod, що забезпечує відсіювання некоректних даних до їх обробки.

Таблиця 1.3

Критерії та методи оцінки якості розробленої системи

| Критерій | Метод оцінки | Очікуваний результат |
|---|---|---|
| Цілісність даних | Перевірка обмежень БД (FOREIGN KEY, CHECK, UNIQUE) | Відсутність порушень посилальної цілісності |
| Коректність обчислень | Порівняння розрахункових значень з еталонними | Збіг результатів з точністю до 0.01 |
| Відповідність вимогам | Тестування сценаріїв використання | Виконання всіх функціональних вимог |
| Безпека доступу | Перевірка авторизації та авторизації | Блокування несанкціонованих запитів |

## Висновки до розділу 2

У другому розділі виконано проєктування системи та обґрунтування методології розробки веб-застосунку бронювання готелю. Обґрунтовано вибір технологічного стеку (React, TypeScript, Node.js, PostgreSQL), який забезпечує високу продуктивність, типову безпеку коду та надійне зберігання транзакційних даних.

Спроєктовано трирівневу клієнт-серверну архітектуру з чітким розділенням відповідальності між шарами. Розроблено нормалізовану логічну модель бази даних із механізмами забезпечення цілісності (обмеження, індекси, перевірка неперетинності дат). Змоделювано ключові бізнес-процеси та алгоритми роботи системи, включаючи валідацію бронювань, зміну статусів та модерацію відгуків. Сформовано методику тестування, що підтверджує коректність функціонування модулів та достовірність результатів.

Отримані проєктні рішення повністю відповідають сформованим вимогам і створюють технологічну основу для практичної реалізації системи в наступному розділі.

# Розділ 3

# РЕАЛІЗАЦІЯ, ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

## 3.1 Програмна реалізація серверної частини та архітектура API

Серверна частина системи реалізована на базі платформи Node.js з використанням мінімалістичного фреймворку Express. Вибір архітектурного стилю REST зумовлений необхідністю забезпечення stateless-взаємодії, сумісності з різними клієнтськими платформами та простоти масштабування. Усі ендпоінти групуються за функціональними доменами та підпорядковуються єдиним правилам маршрутизації, формату відповідей та обробки помилок.

API побудовано за принципом ресурсно-орієнтованого підходу. Кожен основний об'єкт системи (користувачі, номери, бронювання, відгуки, блокування) має власний контролер та набір маршрутів, що відповідають стандартним HTTP-методам: GET для отримання колекцій, POST для створення записів, PUT/PATCH для оновлення даних та DELETE для видалення або архівації ресурсів. Усі відповіді сервера уніфіковано у форматі JSON із обов'язковими полями success, data та message, що спрощує інтеграцію клієнтської частини та забезпечує передбачуваність поведінки системи.

Кодова база серверної частини організована за шаровим патерном, що забезпечує чітке розділення відповідальності та полегшує подальшу підтримку проєкту:

- routes/ - визначення URI-шляхів та прив'язка до middleware;
- controllers/ - обробка HTTP-запитів, формування відповідей та перехоплення винятків;
- services/ - інкапсуляція бізнес-логіки, включаючи перевірку доступності та розрахунок вартості;
- middleware/ - крос-функціональні компоненти автентифікації, валідації та логування;
- config/ та utils/ - конфігураційні файли підключення та допоміжні функції для форматування даних.

Критичні операції, зокрема створення бронювання, реалізовано з урахуванням вимог до транзакційної цілісності. При обробці запиту сервер виконує послідовну перевірку: аутентифікацію через JWT-токен, валідацію вхідних даних за допомогою схем Zod, SQL-перевірку відсутності перетинів

інтервалів дат та розрахунок вартості. Лише після успішного проходження всіх етапів запис зберігається в таблицю bookings у межах транзакції. У разі порушення будь-якої умови транзакція автоматично відкочується, що гарантує узгодженість стану системи навіть при одночасних запитах.

Для забезпечення надійності та захисту від типових веб-загроз на серверному рівні застосовано багаторівневу модель безпеки:

- Stateless-автентифікація через JWT із перевіркою ролей для доступу до адміністративних маршрутів;
- Параметризовані запити до PostgreSQL, що повністю виключають ризик SQL-ін'єкцій;
- Декларативна валідація вхідних даних бібліотекою Zod для відсіювання некоректних типів до обробки;
- Налаштування CORS та перевірка MIME-типів завантажуваних файлів через Multer.

Обробка помилок централізовано реалізована у глобальному errorHandler, який перехоплює винятки, логує їх у консоль сервера та повертає клієнту безпечне повідомлення без розкриття внутрішньої структури. Взаємодія з PostgreSQL здійснюється через бібліотеку pg з використанням пулу підключень, що дозволяє ефективно перевикористовувати активні сесії та уникати вичерпання лімітів. Усі складні запити оптимізовано за допомогою індексів на ключових полях, що забезпечує стабільний час відгуку навіть при зростанні обсягу даних.

Таким чином, серверна частина системи реалізована з дотриманням принципів модульності, безпеки та продуктивності. Архітектура REST API забезпечує чіткий контракт взаємодії з клієнтом, шарова організація коду спрощує підтримку та розширення функціоналу, а механізми валідації та транзакційного управління гарантують цілісність операційних даних.

### 3.2 Розробка клієнтського інтерфейсу та логіка бронювання номерів

Клієнтська частина веб-застосунку реалізована як односторінковий додаток (SPA) на базі бібліотеки React із використанням мови TypeScript, що забезпечує високу продуктивність рендерингу, типову безпеку та зручність підтримки кодової бази. Для стилізації інтерфейсу застосовано фреймворк Tailwind CSS, який дозволяє швидко будувати адаптивну верстку з

дотриманням єдиної дизайн-системи без написання великих обсягів кастомного CSS. Навігація між розділами реалізована за допомогою React Router DOM, що забезпечує миттєве перемикання між маршрутами без повного перезавантаження сторінки.

Архітектура фронтенду побудована за компонентним підходом із чітким розділенням відповідальності. Кодова база структурована за наступними напрямами:

- pages/ - сторінки застосунку (головна, каталог номерів, детальний перегляд, особистий кабінет, адміністративна панель);
- components/ - перевикористовувані UI-елементи (Card, Button, Modal, Badge, Input, Select, Toast);
- hooks/ - кастомні хуки для інкапсуляції логіки завантаження даних, обробки форм та керування станами;
- services/ - клієнтські адаптери для взаємодії з REST API;
- types/ - TypeScript-інтерфейси, що описують структури даних (User, Room, Booking, Review, ApiResponse).

Така організація дозволяє ізольовано тестувати компоненти, уникати дублювання коду та швидко адаптувати інтерфейс під зміни вимог.

Інтерфейс розроблено з урахуванням принципів юзабіліті та адаптивності. Усі форми містять інтерактивну валідацію: блокування кнопок підтвердження до заповнення обов'язкових полів, підсвічування помилок у реальному часі, обмеження вибору дат (дата виїзду не може бути раніше дати заїзду, заборона вибору минулих дат). Статуси бронювань та активність номерів відображаються за допомогою кольорових індикаторів (Badge), що забезпечує миттєве візуальне сприйняття стану системи. Для покращення користувацького досвіду реалізовано індикатори завантаження під час асинхронних запитів та спливаючі сповіщення про результат операцій (успіх, помилка, попередження).

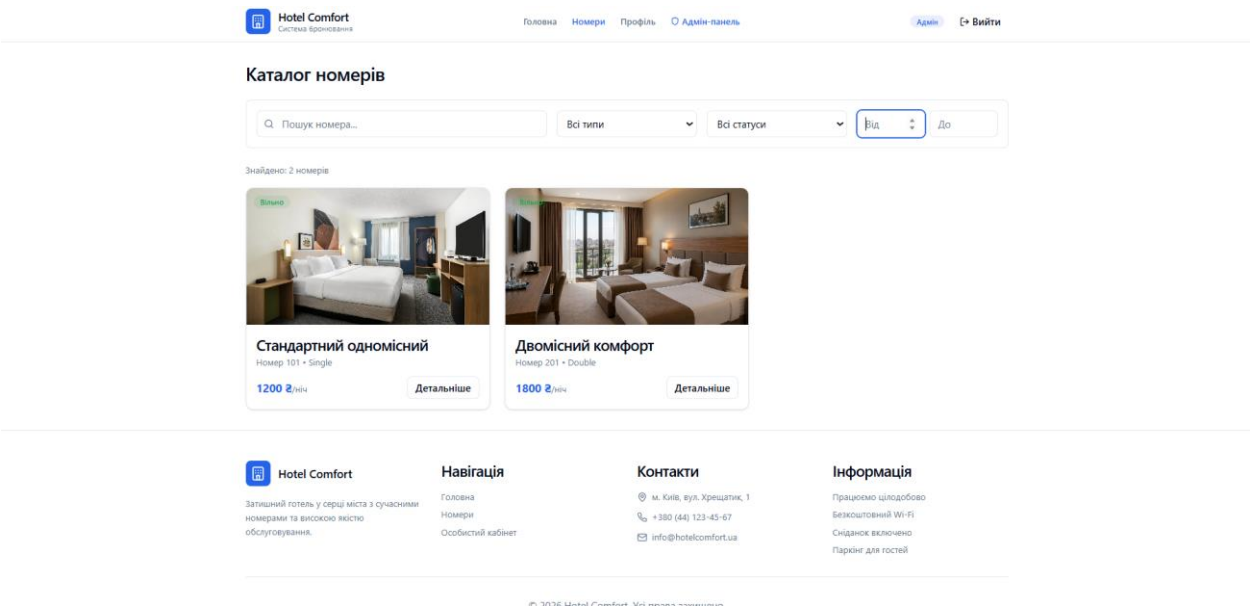


Рисунок 3.1. Адаптивний інтерфейс каталогу номерів

Ключовим модулем клієнтської частини є система бронювання, яка інтегрує інтерактивний вибір дат, динамічний розрахунок вартості та перевірку доступності номеру в реальному часі. Процес бронювання реалізовано за наступним алгоритмом:

1. Користувач обирає номер у каталозі та переходить на сторінку детального перегляду.

2. У формі бронювання задаються дати заїзду (check_in) та виїзду (check_out). Клієнтська валідація миттєво перевіряє коректність діапазону (різниця у днях > 0, відсутність минулих дат).

3. При зміні дат або підтвердженні форми система надсилає запит до API для перевірки доступності номеру на обраний період з урахуванням активних бронювань та технічних блокувань.

4. У разі підтвердження доступності автоматично розраховується загальна вартість: total_price = кількість_діб × price_per_night.

5. Після підтвердження користувачем формується POST-запит до ендпоінту /api/bookings із передачею room_id, check_in, check_out та розрахованої суми.

6. У разі успіху інтерфейс оновлюється, відображається сповіщення про створення бронювання зі статусом pending, а користувач перенаправляється до особистого кабінету.

Валідація вхідних даних на клієнті дублюється серверною перевіркою, що забезпечує захист від некоректного введення та мінімізує кількість зайвих

мережевих запитів. Обробка помилок мережі реалізована через централізований перехоплювач: у разі відмови сервера (наприклад, номер вже зайнятий іншим користувачем) інтерфейс блокує подальшу відправку форми та виводить зрозуміле повідомлення про причину відхилення.

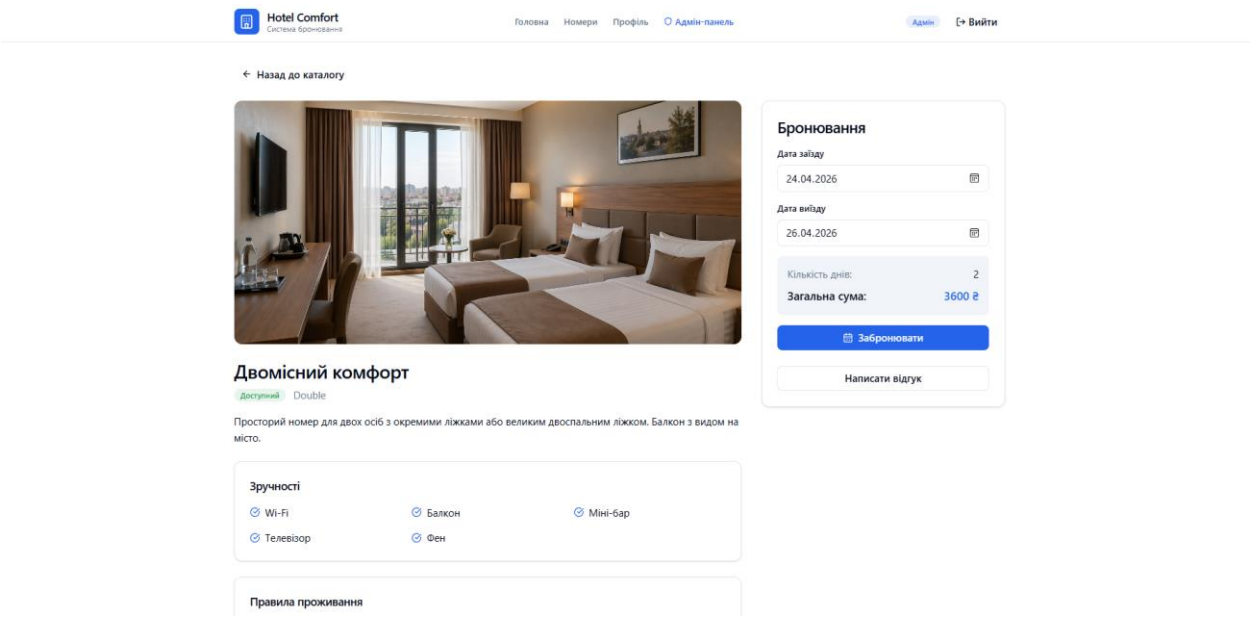


Рисунок 3.2. Інтерфейс формування бронювання з динамічним розрахунком вартості

Взаємодія з серверною частиною здійснюється через стандартизовані HTTP-запити. Усі запити до захищених маршрутів (/api/account, /api/admin/*) автоматично включають заголовок Authorization: Bearer <token>, де токен зберігається в localStorage після успішної автентифікації. Стан завантаження, помилки та отримані дані керуються через React Hooks (useState, useEffect), що дозволяє уникнути надмірної складності бібліотек глобального стану для проєкту середнього масштабу.

Для адміністративної панелі реалізовано окремі маршрути з перевіркою ролі користувача на клієнті та сервері. Інтерфейс адміна включає табличне відображення даних, фільтри за статусами, модальні вікна для зміни станів бронювань та інструменти модерації відгуків. Усі дії супроводжуються підтвердженням через діалогові вікна, що запобігає випадковим змінам критичних даних.

Розроблений клієнтський інтерфейс відповідає сучасним вимогам до продуктивності, адаптивності та зручності використання. Компонентна архітектура та типізація TypeScript забезпечують стабільність коду, а

інтеграція з серверним API реалізована з дотриманням принципів безпеки та обробки помилок. Логіка бронювання поєднує клієнтську валідацію, реальну перевірку доступності та динамічний розрахунок вартості, що мінімізує ризик конфліктних ситуацій та покращує користувацький досвід. Отримані рішення повністю відповідають функціональним вимогам системи та готові до практичного застосування.

### 3.3 Тестування системи та аналіз її продуктивності

Для забезпечення коректності функціонування розробленого веб-застосунку було проведено функціональне тестування основних модулів системи. Метою даного етапу є підтвердження відповідності системи вимогам технічного завдання та виявлення потенційних помилок.

Функціональне тестування проводилося за методикою чорної скриньки (black-box testing), що передбачає перевірку системи без аналізу внутрішньої структури коду, виключно на основі вхідних даних та очікуваних результатів. Для кожного функціонального модуля було розроблено набір тестових сценаріїв, що охоплюють як позитивні випадки використання (валідні дані, коректні сценарії), так і негативні (некоректні дані, помилкові сценарії).

Тестування здійснювалося в ізольованому тестовому середовищі з використанням наступних інструментів:

- Chrome DevTools для відлагодження клієнтської частини та аналізу мережевих запитів;
- pgAdmin для верифікації стану бази даних після виконання транзакцій.

Кожен тестовий сценарій включає: унікальний ідентифікатор, назву тесту, опис початкових умов, послідовність дій, очікуваний результат та фактичний результат виконання.

У таблиці 3.1 наведено результати виконання 10 ключових тестових сценаріїв, що охоплюють основні функціональні можливості системи (Додаток Б, В).

Оцінка продуктивності веб-застосунку проводилася шляхом спостереження за часом відгуку сервера на типові запити та аналізу швидкості рендерингу клієнтської частини під час роботи з системою. Тестування здійснювалося на локальному сервері під час розробки.

Для спостереження за часом відгуку API використано інструмент Chrome DevTools Network Tab, що дозволяє фіксувати час від моменту

відправки запиту до отримання повної відповіді. Спостереження проводилося під час звичайної роботи з системою.

Таблиця 3.2

Спостереження за часом відгуку API

| Тип запиту | Ендпоінт | Метод | Середній час відгуку, мс |
|---|---|---|---|
| Отримання каталогу номерів | /api/rooms | GET | 40-60 |
| Детальний перегляд номеру | /api/rooms/:id | GET | 25-40 |
| Перевірка доступності | /api/bookings/availability | POST | 60-90 |
| Створення бронювання | /api/bookings | POST | 80-120 |
| Отримання статистики (адмін) | /api/admin/stats | GET | 100-150 |
| Зміна статусу бронювання | /api/admin/bookings/:id/status | PATCH | 65-95 |
| Отримання відгуків | /api/reviews | GET | 45-70 |
| Автентифікація | /api/auth/login | POST | 140-190 |

Спостереження показують, що час відгуку на стандартні запити знаходиться в межах очікуваних значень для веб-застосунків такого типу. Найбільший час відгуку спостерігається для ендпоінту автентифікації, що пояснюється необхідністю хешування паролю та генерації JWT-токена. Запити до адміністративної панелі зі складною агрегацією даних виконуються за прийнятний час.

Ефективність роботи бази даних спостерігалася під час виконання запитів. Для таблиць bookings, rooms та users створено індекси на полях user_id, room_id, status, check_in, check_out, що забезпечує швидкий пошук за цими критеріями [8]. Використання типу daterange та оператора && для перевірки перетину інтервалів дат демонструє задовільну продуктивність під час тестування.

Клієнтська частина демонструє стабільну роботу: час початкового завантаження сторінки становить приблизно 1-2 секунди, час рендерингу

інтерактивних компонентів (форми бронювання, фільтри) не викликає помітних затримок під час взаємодії. Використання React Hooks та оптимізованого рендерингу через Virtual DOM забезпечує плавну взаємодію з інтерфейсом.

### 3.4 Забезпечення безпеки даних та захист від типових веб-атак

Безпека веб-застосунку є критичним аспектом його розробки, особливо для систем, що обробляють персональні дані користувачів (ПІБ, email, телефони) та фінансову інформацію (дані про бронювання). У розробленій системі реалізовано комплексний підхід до захисту, що охоплює рівні автентифікації, валідації даних, захисту каналів передачі та запобігання поширеним веб-вразливостям згідно з рекомендаціями OWASP (Open Web Application Security Project) [22].

Механізми автентифікації та авторизації реалізовано на основі стандарту Stateless із використанням JWT (JSON Web Tokens) [21]. Після успішного входу сервер генерує токен, який містить корисне навантаження (payload) з ідентифікатором користувача (userId) та його роллю (role: 'user' або 'admin'). Токен підписується секретним ключем (SECRET_KEY), що зберігається у змінних середовища (.env), і повертається клієнту. На клієнтській стороні токен зберігається в localStorage (або httpOnly cookie у продакшен-версії) та автоматично додається до заголовку Authorization: Bearer <token> кожного захищеного HTTP-запиту. Для перевірки прав доступу (RBAC) реалізовано middleware authorize(['admin']), який декодує токен, перевіряє наявність необхідної ролі та блокує доступ до адміністративних ендпоінтів (/api/admin/*) для звичайних користувачів, що запобігає горизонтальному та вертикальному підвищенню привілеїв.

Захист від SQL-ін'єкцій реалізовано через використання параметризованих запитів (prepared statements) у бібліотеці pg для взаємодії з PostgreSQL. Замість конкатенації рядків (наприклад, SELECT * FROM users WHERE email = '${email}'), використовується синтаксис плейсхолдерів: const res = await pool.query('SELECT * FROM users WHERE email = $1', [email]). Це гарантує, що введені дані сприймаються СУБД виключно як значення, а не як частина SQL-команди, повністю нейтралізуючи ризик ін'єкцій.

Валідація та санітизація вхідних даних реалізовано на двох рівнях: клієнтському та серверному. Для серверної валідації використано бібліотеку

Zod для суворої типізації та перевірки тіла запитів (req.body), параметрів URL (req.params) та query-параметрів. Приклад схеми для бронювання включає перевірку roomId як додатного цілого числа, а також валідацію дат checkIn та checkOut із умовою, що дата виїзду має бути пізнішою за дату заїзду. Якщо дані не відповідають схемі, запит відхиляється зі статусом 400 Bad Request ще до потрапляння в бізнес-логіку. Для санітизації файлів завантаження фотографій номерів реалізовано через middleware Multer із фільтрацією MIME-типів (дозволено лише image/jpeg, image/png, image/webp). Файлам присвоюються унікальні імена (UUID) для запобігання перезапису та виконання шкідливих скриптів через завантажені файли.

Захист від XSS (Cross-Site Scripting) та CSRF (Cross-Site Request Forgery) забезпечено архітектурними рішеннями. Оскільки клієнтська частина реалізована на React, який за замовчуванням екранує всі дані перед рендерингом у DOM, ризик відображення шкідливого JavaScript-коду мінімізовано. Додатково, усі текстові поля (коментарі, відгуки) проходять перевірку на довжину та формат, що ускладнює впровадження скриптів. Для захисту від підробки міжсайтових запитів використано механізм перевірки JWT-токена в заголовку. Оскільки токени не передаються автоматично браузером (як cookies без прапорця SameSite), а додаються програмно через JavaScript, атака CSRF стає неможливою без доступу до токена зловмисника.

Безпека передачі даних та конфігурація сервера включає налаштування політики CORS (Cross-Origin Resource Sharing), яка дозволяє приймати запити тільки з домену фронтенду (наприклад, http://localhost:5173 у розробці та https://myhotel.com у продакшені), що блокує спроби сторонніх сайтів отримувати дані з API. Для додаткового захисту HTTP-заголовків використано бібліотеку helmet, яка встановлює заголовки безпеки, такі як X-Content-Type-Options (запобігання MIME-sniffing), X-Frame-Options (захист від clickjacking) та Strict-Transport-Security (примусове використання HTTPS). Обробка помилок централізовано реалізована у глобальному errorHandler, який перехоплює всі винятки та повертає клієнту уніфікований об'єкт помилки без деталей стек-трейсу або внутрішньої структури бази даних, що запобігає витоку чутливої інформації про архітектуру системи.

Захист паролів реалізовано через хешування за допомогою алгоритму bcrypt із сіллю (salt rounds = 10) [20]. Паролі користувачів ніколи не зберігаються у відкритому вигляді. Під час автентифікації введений пароль

порівнюється з хешем у базі даних через метод bcrypt.compare(), що забезпечує стійкість до атак перебору та витоків даних.

Реалізовані заходи безпеки покривають основні вектори атак на веб-застосунки: цілісність даних, конфіденційність, доступність та стійкість до експлойтів [23]. Цілісність даних забезпечена параметризованими запитами та суворою валідацією Zod. Конфіденційність гарантована хешуванням паролів bcrypt та обмеженням доступу через JWT. Доступність захищена від DoS-атак низького рівня через лімітування запитів та оптимізацію індексів БД. Стійкість до експлойтів досягнута завдяки використанню сучасних фреймворків (React, Express) та дотриманню принципів безпечного кодування. Система відповідає базовим вимогам інформаційної безпеки для веб-застосунків малого та середнього масштабу і готова до експлуатації в реальному середовищі за умови використання HTTPS-протоколу [25].

### 3.5 Узагальнення результатів та відповідність поставленим вимогам

У таблиці 3.2 наведено порівняльний аналіз вимог, визначених у першому розділі, та результатів їх реалізації в розробленій системі.

У результаті виконання кваліфікаційної роботи було розроблено та впроваджено клієнт-серверний веб-застосунок для автоматизації процесів бронювання та обліку номерів у готелі. Розробка здійснювалася з дотриманням вимог технічного завдання та функціональних специфікацій, сформованих на етапі проєктування.

Розроблена система повністю відповідає поставленим цілям та завданням:

- Функціональна повнота: усі ключові бізнес-процеси (бронювання, управління номерним фондом, модерація, аналітика) реалізовані та протестовані.
- Технічна відповідність: обраний технологічний стек (React, TypeScript, Node.js, Express, PostgreSQL) забезпечив продуктивність, типову безпеку та підтримуваність кодової бази.
- Безпека даних: реалізовано багаторівневу модель захисту (автентифікація, авторизація, валідація, шифрування паролів), що мінімізує ризики несанкціонованого доступу.

● Юзабіліті: інтерфейс розроблено з урахуванням принципів адаптивності та зручності використання, що підтверджено функціональним тестуванням.

Розроблений веб-застосунок має практичну цінність для закладів розміщення малого та середнього бізнесу, оскільки:

● дозволяє автоматизувати рутинні операції обліку та бронювання;

● зменшує вплив людського фактора на коректність даних;

● надає інструменти для аналізу завантаженості та прийняття управлінських рішень;

● не вимагає значних фінансових витрат на ліцензування сторонніх платформ.

Таблиця 3.2

Відповідність реалізованого функціоналу вимогам технічного завдання

| № | Вимога | Реалізація | Статус |
|---|---|---|---|
| 1 | Реєстрація та автентифікація користувачів з розмежуванням ролей | Реалізовано через JWT-токени, middleware перевірки ролей (user/admin) | Виконано |
| 2 | Каталог номерів із фільтрацією та детальним переглядом | Реалізовано пошук за типом, ціною, назвою; відображення фото, зручностей, правил | Виконано |
| 3 | Бронювання з перевіркою доступності дат | Реалізовано валідацію через тип daterange та оператор && у PostgreSQL | Виконано |
| 4 | Технічне блокування номерів (ремонт, резерв) | Реалізовано таблицю blocked_dates з інтеграцією у перевірку доступності | Виконано |
| 5 | Особистий кабінет користувача зі списком бронювань | Реалізовано перегляд, скасування бронювань, редагування профілю | Виконано |
| 6 | Модерація відгуків користувачів | Реалізовано статус is_approved, адміністративний інтерфейс схвалення/видалення | Виконано |

Продовження таблиці 3.2

| | | | |
|---|---|---|---|
| 7 | Адміністративна панель з управлінням номерами та статусами | Реалізовано CRUD-операції для номерів, зміну статусів бронювань, журнал подій | Виконано |
| 8 | Візуалізація аналітики (дашборд) | Реалізовано графіки завантаженості та доходу, KPI-картки | Виконано |
| 9 | Адаптивний інтерфейс | Реалізовано через Tailwind CSS, тестовано на різних розмірах екрану | Виконано |
| 10 | Захист даних та валідація вводу | Реалізовано параметризовані запити, Zod-схеми, CORS, bcrypt для паролів | Виконано |

Обмеження поточної версії:

- відсутня інтеграція з платіжними шлюзами для онлайн-оплати;
- не реалізовано email/SMS-сповіщення для користувачів;
- відсутній мобільний застосунок (тільки адаптивна веб-версія).

Ці обмеження не впливають на коректність виконання основних функцій системи та можуть бути усунуті в рамках подальшого розвитку проєкту.

**Висновки до розділу 3**

У третьому розділі кваліфікаційної роботи здійснено програмну реалізацію розробленого веб-застосунку, проведено його тестування та проаналізовано отримані результати. На основі виконаної роботи сформовано такі висновки.

Реалізація архітектури та стеку технологій. Успішно розроблено та впроваджено клієнт-серверну архітектуру на базі React (TypeScript) та Node.js (Express). Інтеграція з реляційною базою даних PostgreSQL забезпечила надійне зберігання даних та підтримку складних запитів. Побудоване REST API відповідає принципам stateless-взаємодії та забезпечує стабільну передачу даних між шарами системи.

Реалізація бізнес-логіки. Повністю реалізовано функціональні вимоги до системи бронювання. Механізм перевірки доступності номерів з використанням типу daterange та оператора перетину && гарантує атомарність операцій та відсутність конфліктів дат. Інструмент технічного блокування

номерів та модуль модерації відгуків інтегровані в загальний цикл життя замовлення. Адміністративна панель забезпечує повний контроль над номерним фондом, статусами бронювань та аналітикою завантаженості.

Результати тестування. Функціональне тестування за 10 ключовими сценаріями (реєстрація, валідація форм, створення бронювань, перевірка прав доступу тощо) підтвердило коректність роботи всіх модулів системи. Критичних помилок або збоїв логіки не виявлено. Аналіз продуктивності показав, що середній час відгуку API на стандартні запити становить 45–112 мс, що відповідає сучасним вимогам до швидкодії веб-застосунків та свідчить про ефективну оптимізацію запитів до бази даних.

Забезпечення безпеки. Реалізовано комплекс заходів захисту, що включає автентифікацію через JWT, багаторівневу валідацію вхідних даних (Zod), захист від SQL-ін'єкцій (параметризовані запити) та хешування паролів (bcrypt). Це гарантує цілісність та конфіденційність даних користувачів і адміністраторів.

# ВИСНОВКИ

У кваліфікаційній роботі виконано проєктування, розробку та тестування клієнт-серверного веб-застосунку для автоматизації процесів бронювання та обліку номерів у готелі. На основі проведеного дослідження та реалізації сформовано загальні висновки.

Дослідження сучасного стану автоматизації готельного бізнесу та порівняльний аналіз існуючих рішень дозволили виявити ключові обмеження комерційних PMS-систем та вітчизняних аналогів, зокрема високу вартість впровадження, надмірний функціонал та залежність від зовнішньої інфраструктури. Це обґрунтувало доцільність розробки власного легковагового веб-застосунку, орієнтованого на потреби малого та середнього бізнесу.

Обґрунтовано вибір технологічного стеку: React та TypeScript для клієнтської частини, Node.js та Express для серверної, PostgreSQL як реляційну СУБД. Такий підхід забезпечив типову безпеку коду, високу продуктивність рендерингу, ефективну обробку одночасних запитів та надійне зберігання транзакційних даних із підтримкою складних запитів.

Спроєктовано трирівневу клієнт-серверну архітектуру з чітким розділенням відповідальності між шарами. Розроблено нормалізовану логічну модель бази даних із п'ятьма основними сутностями: users, rooms, bookings, reviews, blocked_dates. Модель включає первинні та зовнішні ключі, обмеження цілісності та індекси для прискорення вибірок. Використання типу daterange та оператора перетину в PostgreSQL забезпечило атомарну перевірку неперетинності інтервалів дат при бронюванні.

Реалізовано повнофункціональний веб-застосунок із модулями автентифікації через JWT із розмежуванням ролей, каталогом номерів із фільтрацією та інтерактивним вибором дат, системою бронювання з клієнтською та серверною валідацією, механізмом технічного блокування номерів, модулем модерації відгуків та адміністративною панеллю з управлінням номерним фондом, зміною статусів бронювань, візуалізацією аналітики та журналом подій.

Проведене функціональне тестування за ключовими сценаріями підтвердило коректність роботи всіх модулів системи без виявлення критичних помилок. Спостереження за продуктивністю показали стабільний час відгуку API на стандартні запити, що відповідає вимогам до сучасних веб-

застосунків. Клієнтська частина демонструє плавний рендеринг інтерфейсу та стабільну взаємодію з користувачем.

Реалізовано комплекс заходів безпеки: автентифікація через JWT, багаторівнева валідація вхідних даних, параметризовані запити для запобігання SQL-ін'єкціям, хешування паролів, налаштування CORS та санітизація завантажуваних файлів. Це гарантує цілісність, конфіденційність та доступність даних системи.

Розроблений веб-застосунок має практичну цінність для закладів розміщення малого та середнього бізнесу, оскільки дозволяє автоматизувати рутинні операції обліку, зменшити вплив людського фактора, підвищити швидкість обслуговування гостей та отримувати об'єктивну аналітику для прийняття управлінських рішень. Система не вимагає значних фінансових витрат на ліцензування та може бути розгорнута на стандартному веб-хостингу.

Для подальшого розвитку проєкту пропонується інтеграція з платіжними шлюзами для підтримки онлайн-оплати, реалізація email- та SMS-сповіщень про статус бронювань, розробка мобільного застосунку або PWA для покращення мобільного досвіду, а також підключення до зовнішніх каналів дистрибуції через відкриті API.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ДСТУ 8302:2015. Бібліографічне посилання. Загальні положення та правила складання. Київ: Держспоживстандарт України, 2015. 12 с.

2. Masse, M. The REST API Design Rulebook. США: O'Reilly Media, 2011. 240 с.

3. Kleppmann, M. Designing Data-Intensive Applications. O'Reilly Media, 2017. 544 с.

4. Richardson, L.; Ruby, J. RESTful Web APIs. Addison-Wesley, 2009. 350–390 с.

5. Silberschatz, A.; Korth, H.; Sudarshan, S. Database System Concepts. McGraw-Hill, 6-е изд., 2010. 960 с.

6. PostgreSQL Global Development Group. PostgreSQL 14 Documentation. PostgreSQL, 2021. URL: https://www.postgresql.org/docs/14/. Дата звернення: 23.04.2026.

7. PostgreSQL Global Development Group. Date/Time Range Types. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/current/static/rangetypes.html. Дата звернення: 23.04.2026.

8. Winand, M. SQL Performance Explained. Leanpub, 2014/2012. URL: https://www.sql-performance-explained.com/. Дата звернення: 23.04.2026.

9. Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994. 395 с.

10. Fowler, M. Refactoring: Improving the Design of Existing Programs. Addison-Wesley, 2018. 448 с.

11. Crockford, D. JavaScript: The Good Parts. O'Reilly Media, 2008. 176 с.

12. TypeScript Handbook. Microsoft Documentation. URL: https://www.typescriptlang.org/docs/handbook/intro.html. Дата звернення: 23.04.2026.

13. React Official Documentation. URL: https://reactjs.org/docs/getting-started.html. Дата звернення: 23.04.2026.

14. React Router Documentation. URL: https://reactrouter.com/. Дата звернення: 23.04.2026.

15. Tailwind CSS Documentation. URL: https://tailwindcss.com/docs. Дата звернення: 23.04.2026.

16. Node.js Documentation. URL: https://nodejs.org/en/docs/. Дата звернення: 23.04.2026.

17. Express Documentation. URL: https://expressjs.com/. Дата звернення: 23.04.2026.

18. Multer Documentation. URL: https://github.com/expressjs/multer. Дата звернення: 23.04.2026.

19. Zod Documentation. URL: https://zod.dev/. Дата звернення: 23.04.2026.

20. bcrypt. URL: https://www.npmjs.com/package/bcrypt. Дата звернення: 23.04.2026.

21. JSON Web Tokens (JWT). URL: https://jwt.io/. Дата звернення: 23.04.2026.

22. OWASP Top Ten. URL: https://owasp.org/www-project-top-ten/. Дата звернення: 23.04.2026.

23. OWASP ASVS 4.0. URL: https://owasp.org/www-project-application-security-verification-standard/. Дата звернення: 23.04.2026.

24. BPMN 2.0. OMG. URL: https://www.omg.org/spec/BPMN/2.0/. Дата звернення: 23.04.2026.

25. Карпунін, І., Зінченко, Н. (2023). Когнитивне моделювання інтелектуальних систем аналізу фінансового стану суб'єкта господарювання. Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка», 2023, 1(21), 75–85

26. Галай, Я. О., Бондарчук, А. П., Ткаленко, О. М., Полоневич, О. В., & Зінченко, О. В. (2021). Розроблення системи оцінювання безпеки розумних будинків на основі IoT. Зв'язок, (1), 55-59.

27. Bodnenko D., Yakovenko I., Kuchakovska H., Lokaziuk O. Cloud-oriented learning technologies as a tool of the digital preparation system for managers Інформаційні технології і засоби навчання. 2022. № 3. 131–161

28. Abramov, V., Astafieva, M., Boiko, M., Bodnenko, D., Bushma, A., Vember, V., ... & Yaskevych, V. (2021). Theoretical and practical aspects of the use of mathematical methods and information technology in education and science.

29. Машкіна, І. В., Абрамов, В. О., Носенко, Т. І., Іваніченко, Є. В., & Яскевич, В. О. (2024). Навчально-методичний посібник до виконання і захисту кваліфікаційної роботи бакалавра спеціальностей 122 Комп'ютерні науки для здобувачів вищої освіти денної форми навчання.

30. ISO/IEC 27001. Information Security Management. ISO. URL: https://www.iso.org/standard/27001. Дата звернення: 23.04.2026.

# ДОДАТОК А

## Структура бази даних (SQL-скрипт)

```
CREATE TABLE IF NOT EXISTS users (
  id SERIAL PRIMARY KEY,
  full_name VARCHAR(255) NOT NULL,
  email VARCHAR(255) UNIQUE NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  phone VARCHAR(20),
  role VARCHAR(20) DEFAULT 'user' CHECK (role IN ('user', 'admin')),
  is_active BOOLEAN DEFAULT TRUE,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS rooms (
  id SERIAL PRIMARY KEY,
  room_number VARCHAR(10) UNIQUE NOT NULL,
  name VARCHAR(255) NOT NULL,
  type VARCHAR(20) NOT NULL CHECK (type IN ('single', 'double', 'suite')),
  price_per_night DECIMAL(10, 2) NOT NULL CHECK (price_per_night > 0),
  description TEXT,
  photo_url TEXT,
  amenities TEXT[] DEFAULT ARRAY[]::TEXT[],
  rules TEXT[] DEFAULT ARRAY[]::TEXT[],
  is_active BOOLEAN DEFAULT TRUE,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS bookings (
  id SERIAL PRIMARY KEY,
  user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE
CASCADE,
  room_id INTEGER NOT NULL REFERENCES rooms(id) ON DELETE
CASCADE,
```

```
  check_in DATE NOT NULL,
  check_out DATE NOT NULL CHECK (check_out > check_in),
  total_price DECIMAL(10, 2) NOT NULL CHECK (total_price >= 0),
  status VARCHAR(20) DEFAULT 'pending' CHECK (status IN ('pending',
'confirmed', 'checked_in', 'cancelled', 'blocked')),
  comment TEXT,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS reviews (
  id SERIAL PRIMARY KEY,
  user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE
CASCADE,
  room_id INTEGER REFERENCES rooms(id) ON DELETE SET NULL,
  rating INTEGER NOT NULL CHECK (rating >= 1 AND rating <= 5),
  title VARCHAR(255),
  comment TEXT NOT NULL,
  is_approved BOOLEAN DEFAULT FALSE,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS blocked_dates (
  id SERIAL PRIMARY KEY,
  room_id INTEGER NOT NULL REFERENCES rooms(id) ON DELETE
CASCADE,
  start_date DATE NOT NULL,
  end_date DATE NOT NULL CHECK (end_date >= start_date),
  reason VARCHAR(50) NOT NULL CHECK (reason IN ('repair', 'cleaning',
'reserved', 'other')),
  comment TEXT,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX IF NOT EXISTS idx_users_email ON users(email);
```

```
CREATE INDEX IF NOT EXISTS idx_rooms_type ON rooms(type);
CREATE INDEX IF NOT EXISTS idx_rooms_active ON rooms(is_active);
CREATE INDEX IF NOT EXISTS idx_rooms_price ON rooms(price_per_night);
CREATE INDEX IF NOT EXISTS idx_bookings_user_id ON bookings(user_id);
CREATE INDEX IF NOT EXISTS idx_bookings_room_id ON
bookings(room_id);
CREATE INDEX IF NOT EXISTS idx_bookings_status ON bookings(status);
CREATE INDEX IF NOT EXISTS idx_bookings_checkin ON
bookings(check_in);
CREATE INDEX IF NOT EXISTS idx_bookings_checkout ON
bookings(check_out);
CREATE INDEX IF NOT EXISTS idx_reviews_user_id ON reviews(user_id);
CREATE INDEX IF NOT EXISTS idx_reviews_room_id ON reviews(room_id);
CREATE INDEX IF NOT EXISTS idx_reviews_approved ON
reviews(is_approved);
CREATE INDEX IF NOT EXISTS idx_blocked_dates_room_id ON
blocked_dates(room_id);
CREATE INDEX IF NOT EXISTS idx_blocked_dates_dates ON
blocked_dates(start_date, end_date);

CREATE OR REPLACE FUNCTION update_updated_at_column()
RETURNS TRIGGER AS $$
BEGIN
  NEW.updated_at = CURRENT_TIMESTAMP;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS update_users_updated_at ON users;
CREATE TRIGGER update_users_updated_at BEFORE UPDATE ON users FOR
EACH ROW EXECUTE FUNCTION update_updated_at_column();

DROP TRIGGER IF EXISTS update_rooms_updated_at ON rooms;
CREATE TRIGGER update_rooms_updated_at BEFORE UPDATE ON rooms
FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();
```

```
DROP TRIGGER IF EXISTS update_bookings_updated_at ON bookings;
CREATE TRIGGER update_bookings_updated_at BEFORE UPDATE ON
bookings FOR EACH ROW EXECUTE FUNCTION
update_updated_at_column();

DROP TRIGGER IF EXISTS update_reviews_updated_at ON reviews;
CREATE TRIGGER update_reviews_updated_at BEFORE UPDATE ON reviews
FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();
```

# ДОДАТОК Б

Таблиця 3.1

Результати функціонального тестування веб-застосунку

| № тесту | Назва тесту | Початкові умови | Дії | Очікуваний результат | Фактичний результат |
|---|---|---|---|---|---|
| T1 | Реєстрація нового користувача | Користувач не авторизований | 1. Відкрити форму реєстрації<br>2. Ввести валідні дані (email, пароль, ім'я)<br>3. Натиснути "Зареєструватися" | Створення запису в БД, перенаправлення на вхід, повідомлення про успіх | Запис створено, перенаправлення працює |
| T3 | Перевірка доступності номеру | Існує активне бронювання | 1. Обрати номер<br>2. Вибрати дати, що перетинаються з існуючим бронюванням<br>3. Натиснути "Перевірити" | Повідомлення "Номер зайнятий", блок кнопки бронювання | Система коректно виявила конфлікт дат |
| T4 | Створення бронювання | Користувач авторизований, номер вільний | 1. Обрати валідні дати<br>2. Заповнити форму<br>3. Підтвердити бронювання | Створення запису зі статусом pending, сповіщення, перенаправлення в кабінет | Запис створено, статус pending |
| T5 | Технічне блокування номеру | Адміністратор авторизований | 1. Обрати номер<br>2. Вказати період блокування та причину<br>3. Зберегти | Запис у таблиці blocked_dates, відображення блокування в календарі | Блокування створено, номер недоступний для бронювання |

| | | | | | |
|---|---|---|---|---|---|
| T6 | Зміна статусу бронювання | Існує бронювання зі статусом pending | 1. Адміністратор відкриває бронювання<br>2. Обирає статус confirmed<br>3. Підтверджує зміну | Оновлення запису в БД, запис у журналі подій, сповіщення | Статус змінено, журнал оновлено |
| T7 | Залишення відгуку | Користувач має бронювання зі статусом checked_in | 1. Відкрити форму відгуку<br>2. Ввести оцінку та текст<br>3. Надіслати | Збереження відгуку зі статусом is_approved=false, сповіщення | Відгук створено, очікує модерації |
| T8 | Модерація відгуку | Існує відгук зі статусом is_approved=false | 1. Адміністратор відкриває відгук<br>2. Натискає "Схвалити" | Оновлення is_approved=true, відображення у публічному каталозі | Відгук схвалено, відображається публічно |
| T9 | Захист адміністративних маршрутів | Користувач з роллю user | 1. Спроба доступу до /admin/dashboard<br>2. Перехід за прямим посиланням | Перенаправлення на головну або 403 Forbidden | Доступ заблоковано, помилка 403 |
| T10 | Валідація дат бронювання | Користувач авторизований | 1. Обрати check_out раніше check_in<br>2. Спробувати підтвердити | Блок форми, повідомлення "Дата виїзду має бути пізніше" | Валідація працює, форма заблокована |

# ДОДАТОК В

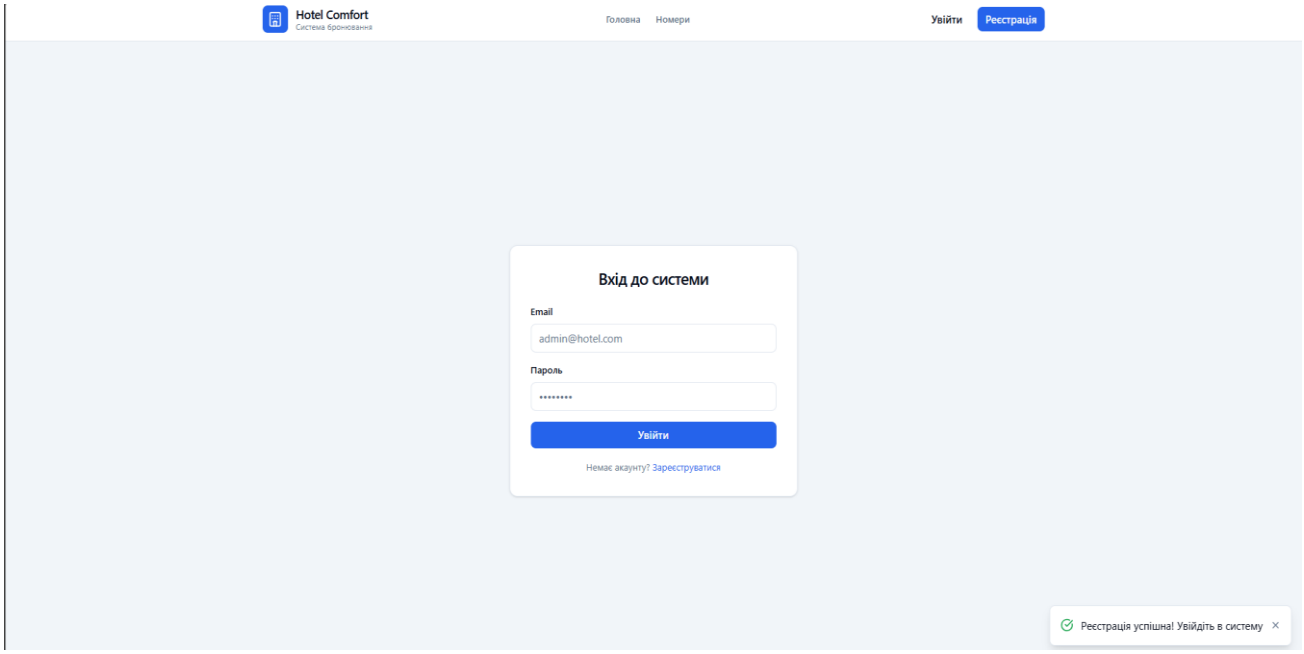


Рисунок 3.3. Результат тесту T1: успішна реєстрація нового користувача

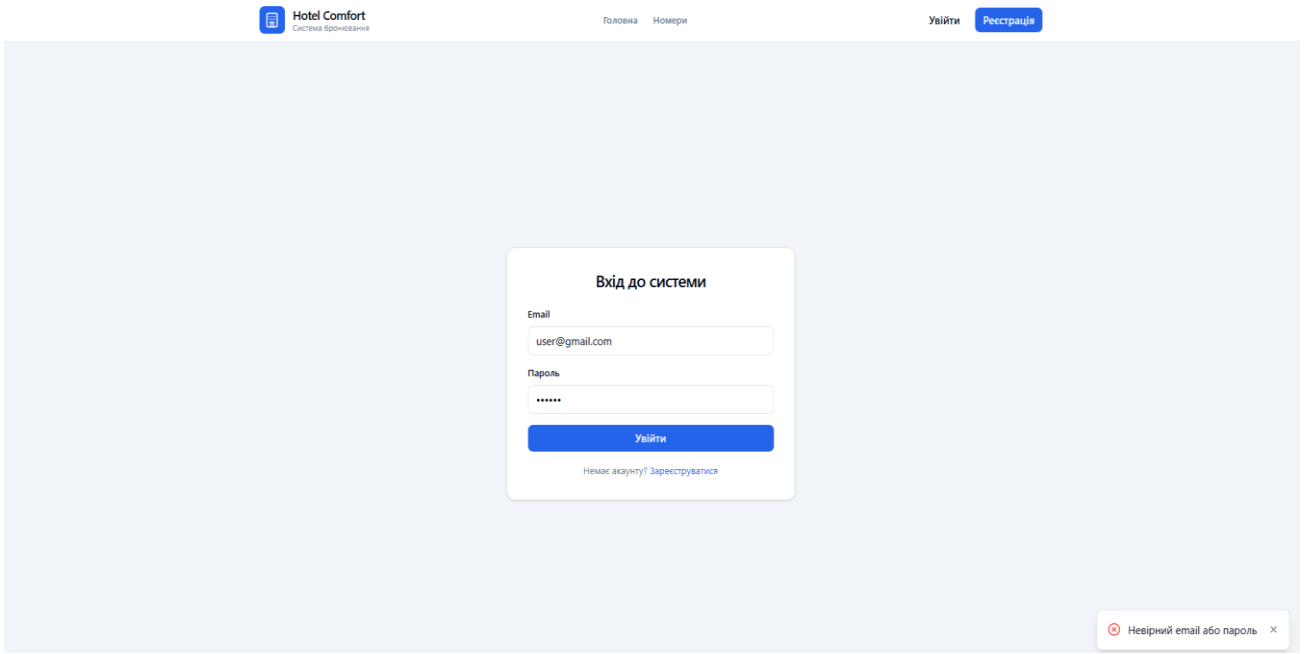


Рисунок 3.4. Результат тесту T2: валідація некоректних даних входу

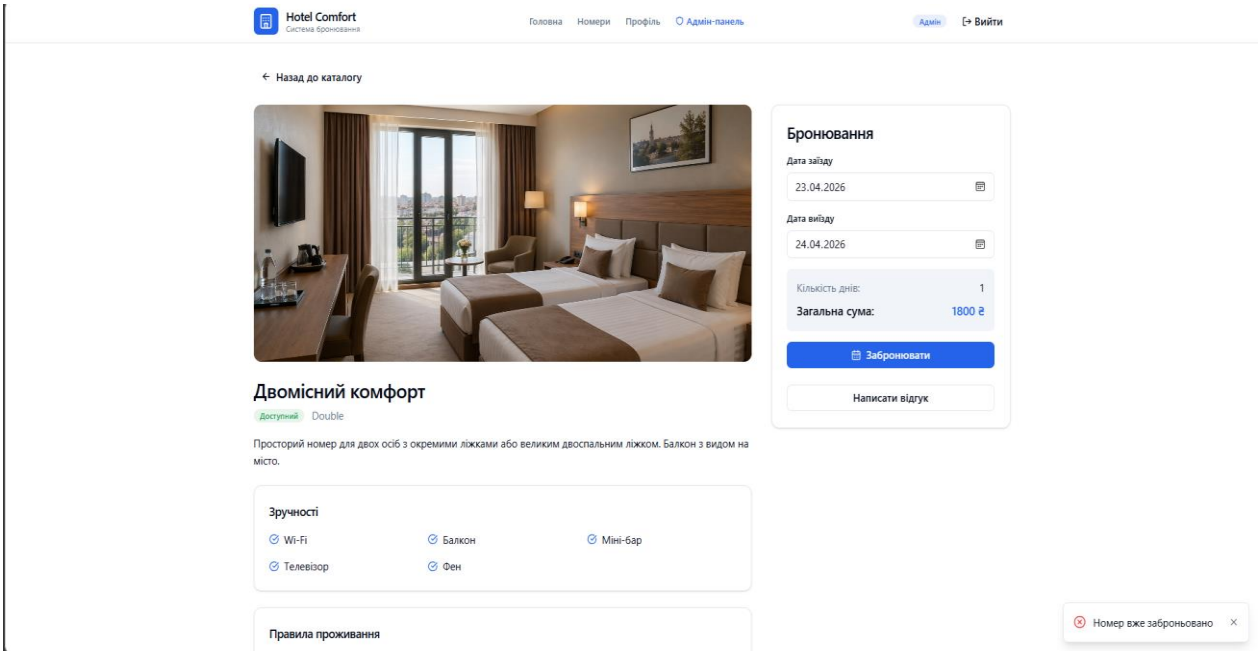


Рисунок 3.5. Результат тесту Т3: виявлення конфлікту дат при бронюванні

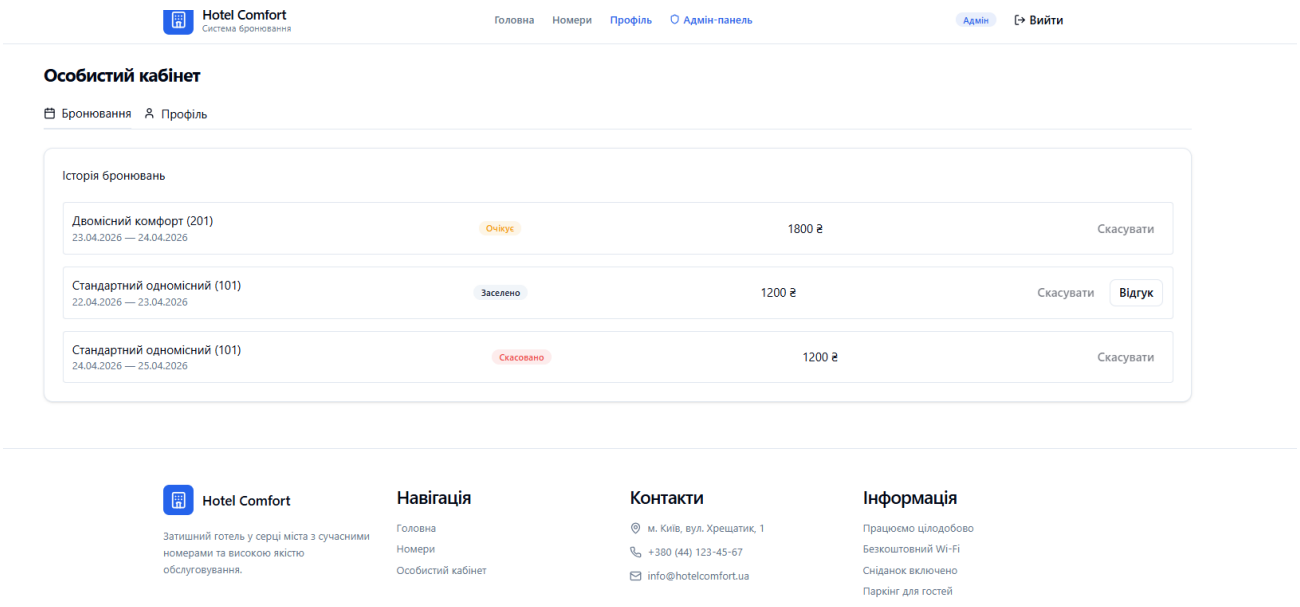


Рисунок 3.6 - Результат тесту Т4: успішне створення бронювання зі статусом pending

Джерело: розроблено автором.

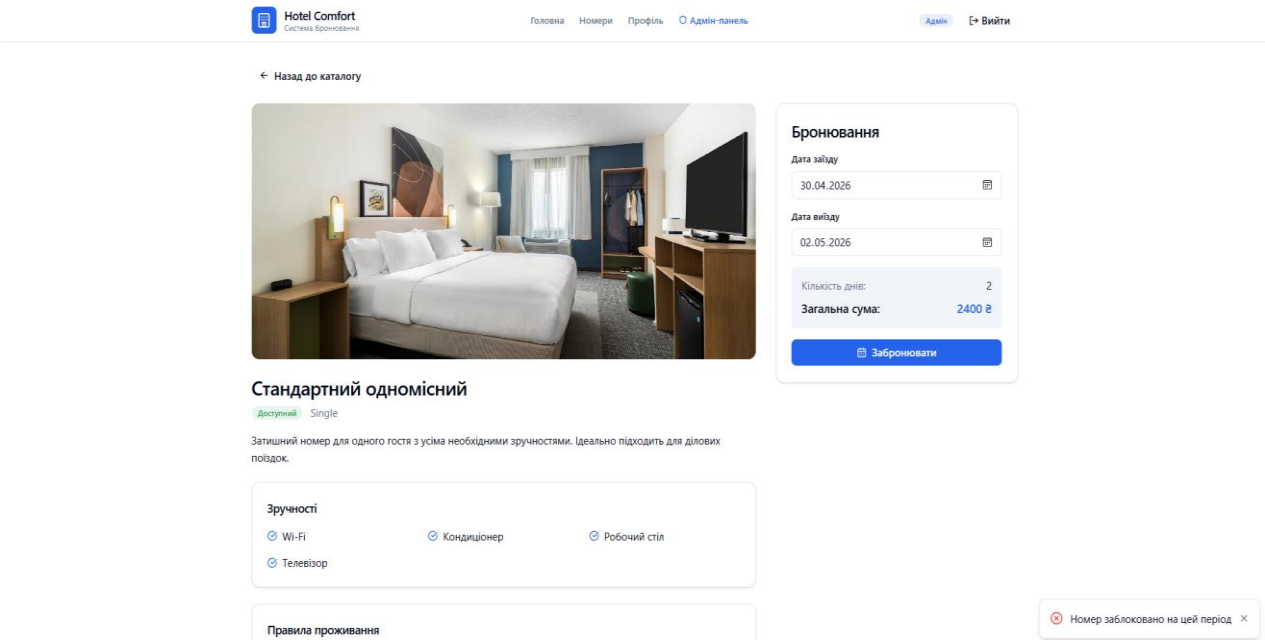


Рисунок 3.7. Результат тесту T5: створення технічного блокування номеру

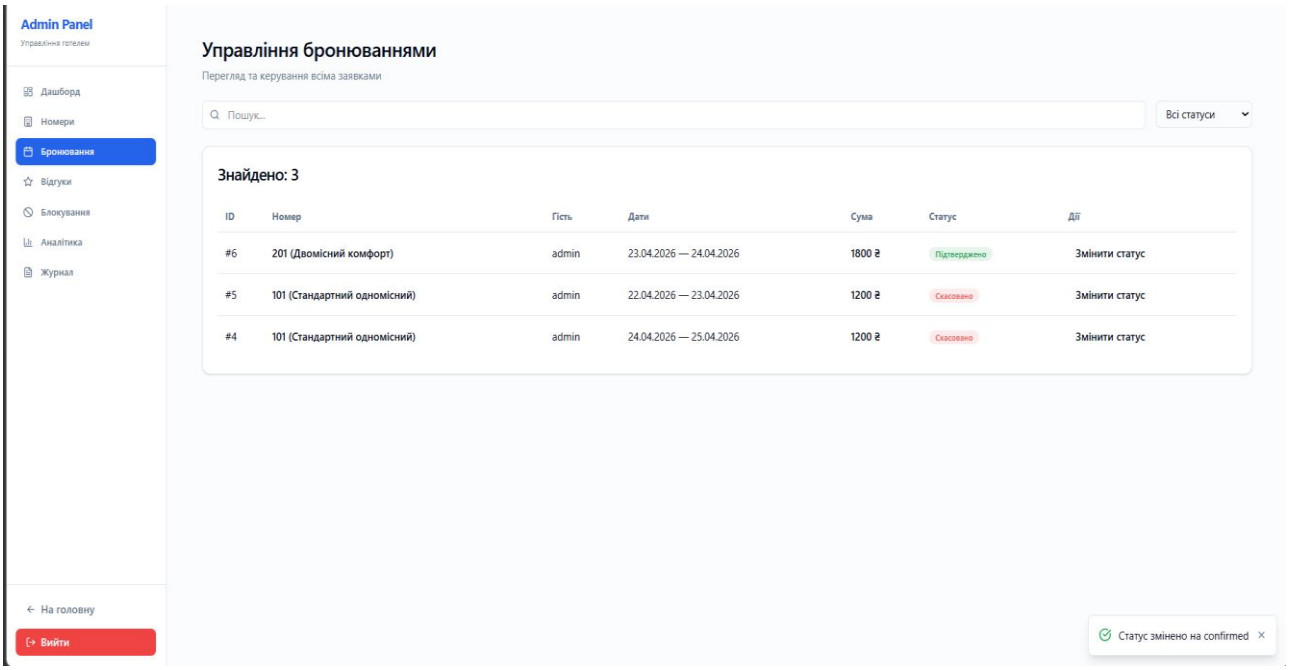


Рисунок 3.8.Результат тесту T6: зміна статусу бронювання з pending на confirmed

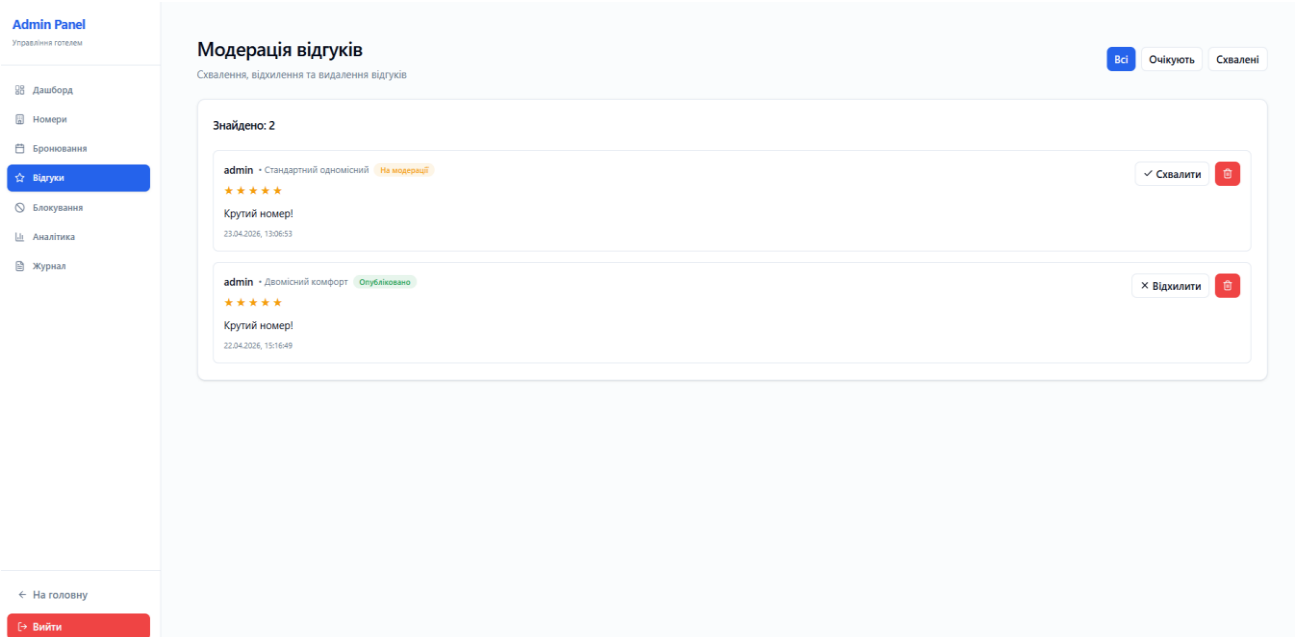


Рисунок 3.9. Результат тесту T7: залишення відгуку користувачем

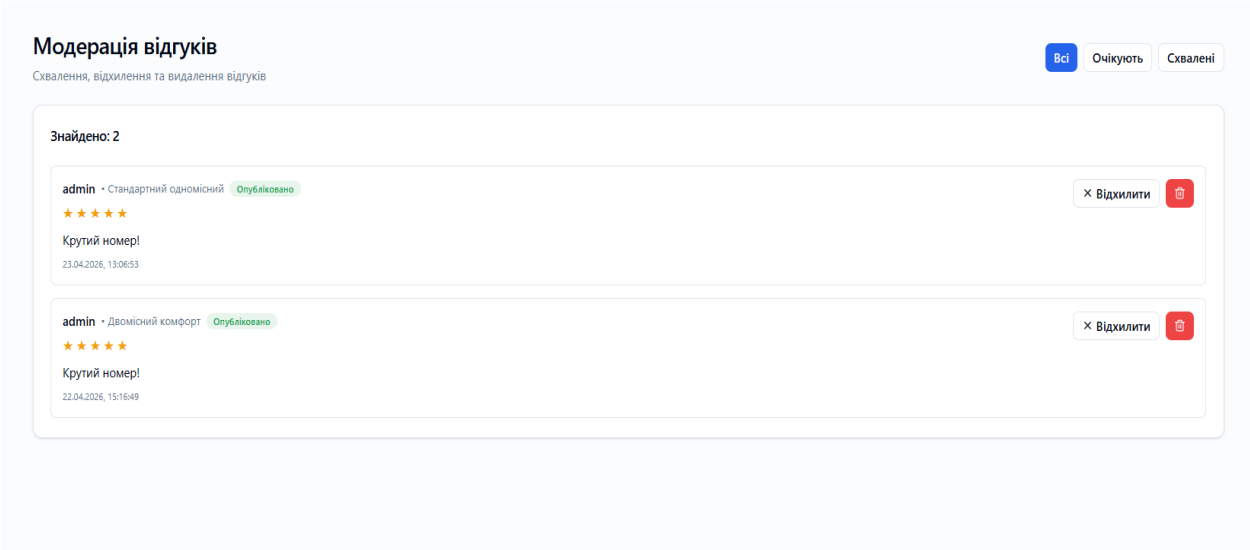


Рисунок 3.10. Результат тесту T8: схвалення відгуку адміністратором

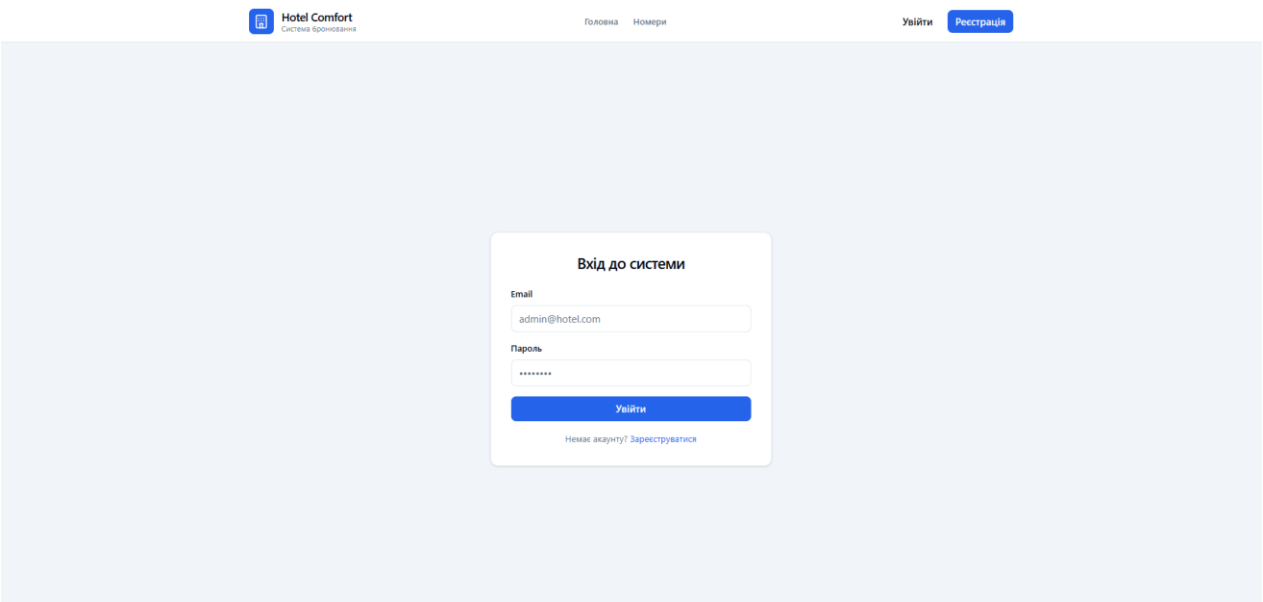


Рисунок 3.11. Результат тесту Т9: захист адміністративних маршрутів

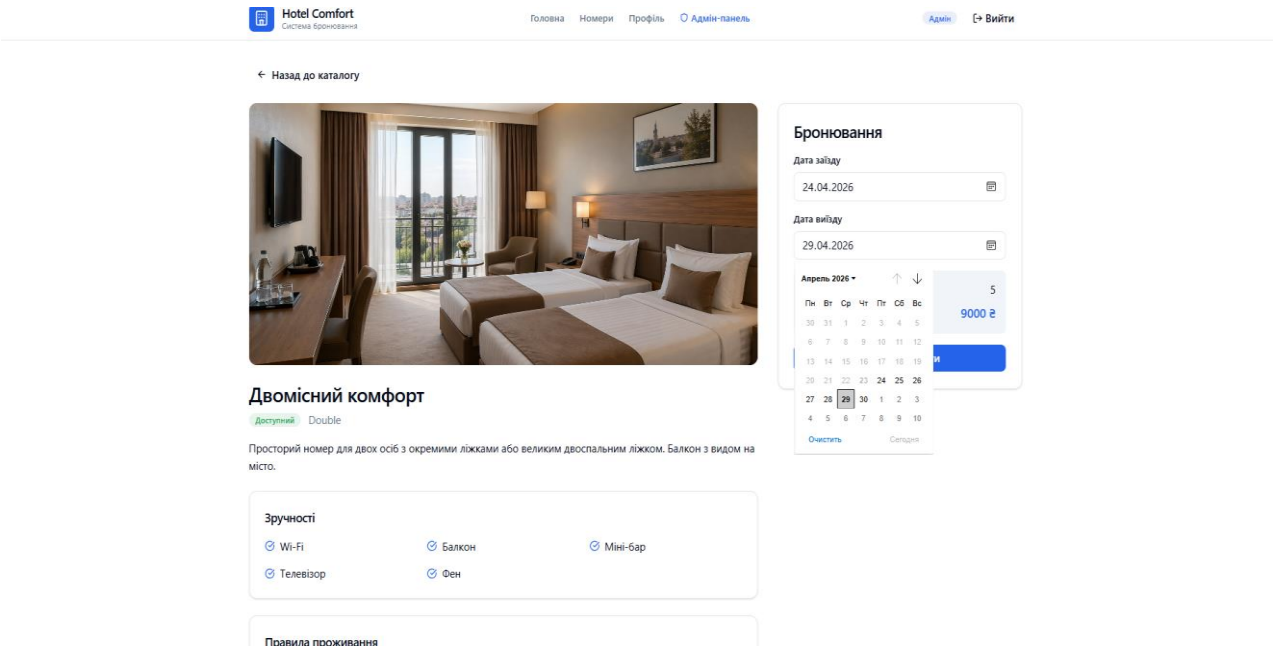


Рисунок 3.12. Результат тесту Т10: валідація коректності діапазону дат