МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ І ПРИКЛАДНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

**ПОЛЬГУН ДЕНИС ПАВЛОВИЧ**

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій,
д-р. техн. наук, професор
_____Наталія ВЕСЕЛОВСЬКА
«____»_____ 2026 р.

# РОЗРОБКА ВЕБ-ПЛАТФОРМИ ДЛЯ ОНЛАЙН-БРОНЮВАННЯ З РЕАЛІЗАЦІЄЮ РОЛЕЙ КОРИСТУВАЧІВ ТА СИСТЕМИ ПОВІДОМЛЕНЬ

Спеціальність 122 Комп’ютерні науки

**Кваліфікаційна (бакалаврська) робота**

Керівник:
Юрій ПОРЕМСЬКИЙ, старший
викладач кафедри інформаційних
технологій,
канд. техн. наук
_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця – 2026

## АНОТАЦІЯ

**Польгун Д. П.** Розробка веб-платформи для онлайн-бронювання з реалізацією ролей користувачів та системи повідомлень. Спеціальність 122 «Комп’ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця, 2026.

У кваліфікаційній (бакалаврській) роботі досліджено принципи побудови веб-платформ для онлайн-бронювання послуг. Розроблено сайт для онлайн-бронювання послуг із підтримкою ролей. Реалізовано механізми реєстрації та авторизації, систему бронювання, внутрішні повідомлення та синхронізацію даних у реальному часу. Для реалізації клієнтської частини використано React, React Router та Axios, серверної – Node.js і Express.js. Зберігання даних забезпечується PostgreSQL та Prisma ORM, а обмін подіями в режимі реального часу реалізовано за допомогою Socket.IO. Безпека доступу забезпечується JWT-аутентифікацією та моделлю рольового керування доступом RBAC.

Ключові слова: веб-платформа, онлайн-бронювання, React, Node.js, PostgreSQL, Prisma ORM, Socket.IO, JWT, RBAC, WebSocket.

## ABSTRACT

**Polhun D. P.** Development of a web platform for online booking with user role implementation and a messaging system. Specialty 122 "Computer Science". Vasyl Stus Donetsk National University, Vinnytsia, 2026.

This bachelor’s thesis explores the principles of building web platforms for online service booking. A website for online service booking with role support was developed. Registration and authorization mechanisms, a booking system, internal messaging, and real-time data synchronization were implemented. React, React Router, and Axios were used to implement the client-side, while Node.js and Express.js were used for the server-side. Data storage is provided by PostgreSQL and Prisma

ORM, and real-time event exchange is implemented using Socket.IO. Access security is ensured by JWT authentication and the RBAC role-based access control model.

Keywords: web platform, online booking, React, Node.js, PostgreSQL, Prisma ORM, Socket.IO, JWT, RBAC, WebSocket.

**ЗМІСТ**

ВСТУП…… 6
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ…… 11
1.1. Особливості систем онлайн-бронювання послуг…….. 11
1.2. Архітектурні особливості веб-платформ бронювання та засобів комунікації…… 15
1.3. Аналіз сучасних платформ для бронювання послуг…….. 18
1.4. Аналіз функціональних можливостей та недоліків існуючих рішень….. 23
1.5. Визначення цільової аудиторії та сценаріїв використання системи……. 25
1.6. Формування функціональних та нефункціональних вимог…….. 29
ВИСНОВКИ ДО РОЗДІЛУ 1…… 32
РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБ-ПЛАТФОРМИ ДЛЯ ОНЛАЙН-БРОНЮВАННЯ…… 33
2.1. Проєктування загальної архітектури веб-платформи для онлайн-бронювання…….. 33
2.2. Проєктування моделі користувачів та розмежування доступу……. 37
2.3. Проєктування системи бронювання та управління часовими інтервалами…….. 39
2.4. Проєктування системи повідомлень…… 43
2.5. Проєктування ER-діаграми…….. 47
ВИСНОВКИ ДО РОЗДІЛУ 2…… 51
РОЗДІЛ 3. РЕАЛІЗАЦІЯ ВЕБ-ПЛАТФОРМИ ДЛЯ ОНЛАЙН-БРОНЮВАННЯ…… 53
3.1. Вибір технологій та середовища розробки…….. 53
3.2. Реалізація серверної частини системи…….. 54
3.2.1. Реалізація REST API…….. 55
3.2.2. Реалізація RBAC-моделі…….. 57
3.2.3. Реалізація бізнес-логіки бронювання…….. 58
3.3. Реалізація бази даних…… 60

3.3.1. Реалізація структури БД у PostgreSQL…… 60
3.3.2. Реалізація зв’язків та обмежень……... 62
3.4. Реалізація клієнтської частини системи…….. 64
3.4.1. Реалізація React-архітектури…… 64
3.4.2. Реалізація інтерфейсу користувача…….. 66
3.4.3. Реалізація frontend RBAC…… 69
3.5. Реалізація real-time синхронізації…… 71
3.5.1. Реалізація WebSocket сервера…….. 71
3.5.2. Реалізація event-driven взаємодії…….. 73
3.5.3. Реалізація real-time оновлення інтерфейсу…….. 74
3.6. Реалізація системи повідомлень……... 76
3.6.1. Реалізація notifications……...77
3.6.2. Інтеграція повідомлень із системою бронювання…….. 79
3.7. Демонстрація роботи веб-платформи…… 81
3.8. Перевірка працездатності системи…….. 85
ВИСНОВКИ ДО РОЗДІЛУ 3…… 89
ВИСНОВКИ…….. 91
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ…… 93

# ВСТУП

У контексті інтенсивного розвитку інформаційних технологій і послідовної цифрової трансформації соціально-економічних практик зростає значущість веб-орієнтованих інформаційних систем, призначених для підтримки та часткової автоматизації повсякденних процесів взаємодії між споживачами й провайдерами послуг. Одним із найбільш поширених і водночас організаційно чутливих процесів є запис на послуги. Традиційно він здійснювався через телефонний зв’язок або шляхом особистого звернення до закладу, що історично відповідало доступним інструментам комунікації. Водночас такі підходи демонструють обмежену масштабованість: вони залежать від робочого часу персоналу, створюють додаткове навантаження на адміністративні ресурси та знижують загальну доступність сервісу для користувачів, які перебувають поза часовими або просторовими межами закладу. Додатковим чинником є вища ймовірність помилок через ручне внесення даних і недостатня прозорість керування розкладом, зокрема у випадках, коли зміни відбуваються часто або паралельно обробляється значна кількість запитів. За цих умов обгрунтованою є потреба у розробленні сучасних веб-платформ, які дають змогу здійснювати бронювання у зручний для користувача час незалежно від його місцезнаходження, а також забезпечують провайдерам інструменти для більш раціонального використання ресурсів і контролю навантаження. У практичному вимірі це означає підтримку актуальності даних про доступність послуг, зменшення кількості адміністративних дій та підвищення передбачуваності сервісних процесів.

Окремої уваги потребує проблема узгодженості дій між учасниками процесу бронювання, оскільки саме синхронізація інформації впливає на якість взаємодії та рівень довіри до платформи. У традиційних схемах, а також у низці спрощених веб-рішень, оперативне інформування сторін про зміну статусу бронювання часто є обмеженим або реалізується із затримками, що спричиняє непорозуміння, дублювання звернень, втрату часових слотів і, як наслідок, погіршення якості обслуговування. Такі ситуації особливо виразні у сферах із

високою варіативністю розкладу або частими змінами, наприклад у медичних, освітніх чи сервісних установах. У цьому контексті інтеграція системи бронювання з механізмами миттєвого обміну повідомленнями розглядається як один із ключових напрямів удосконалення подібних платформ. Використання технологій реального часу, зокрема WebSocket, забезпечує сталість з'єднання між клієнтом і сервером і дає змогу передавати оновлення щодо створення, підтвердження або скасування бронювання без необхідності повторного завантаження сторінки. Крім того, такі механізми відкривають можливість організації більш повноцінної комунікації між користувачами та провайдерами, що може охоплювати уточнення деталей, запит додаткової інформації або узгодження перенесення часу. Водночас практична цінність цієї інтеграції пов'язана не лише зі швидкістю обміну даними, а й зі зменшенням інформаційної асиметрії між учасниками та підвищенням прогнозованості процесу надання послуг.

**Актуальність обраної теми** визначається потребою підвищення ефективності процесів бронювання послуг, забезпечення більш оперативної та надійної взаємодії між користувачами і провайдерами, а також впровадження технологічних рішень, що підтримують функціональність у режимі реального часу. Розроблення веб-платформи, яка поєднує керування бронюванням із вбудованою системою миттєвих повідомлень, потенційно сприятиме підвищенню зручності користування, зменшенню кількості організаційних збоїв і скороченню часу реакції на зміну подій. У ширшому контексті цифрових сервісів такі характеристики корелюють із вимогами конкурентного середовища, у якому користувачі очікують не лише доступності сервісу, а й прозорості процесів, швидкого підтвердження дій і зрозумілих каналів комунікації. При цьому доцільно враховувати, що ефективність подібної платформи залежить від узгодженості її компонентів: логіки бронювання, коректності даних розкладу, політик доступу й механізмів повідомлень, а також від того, наскільки ці компоненти підтримують реальні сценарії використання.

**Метою даного дослідження** є розроблення веб-платформи для онлайн-бронювання послуг, яка забезпечує ефективне керування процесом запису користувачів з урахуванням часових обмежень, ролей учасників системи та реалізації інтегрованої системи повідомлень у режимі реального часу. Досягнення цієї мети передбачає створення програмного продукту, що поєднує функціональність бронювання, механізми рольового доступу та інструменти миттєвої комунікації, а також демонструє практичне застосування сучасних підходів до проєктування і реалізації веб-додатків. У межах такої мети важливо не лише реалізувати окремі модулі, але й забезпечити їхню узгоджену роботу, аби платформа підтримувала типові для предметної області сценарії: від перегляду доступних слотів і створення бронювання до оперативного інформування сторін ***та розв'язання ситуацій, пов'язаних зі зміною планів***.

Для досягнення поставленої мети необхідно вирішити такі **завдання дослідження:**

1. провести аналіз предметної області, зокрема наявних підходів до організації систем бронювання послуг та практик комунікації між користувачами, із урахуванням їхніх переваг, обмежень і типових джерел помилок;
2. дослідити принципи побудови веб-орієнтованих інформаційних систем, включаючи клієнт-серверну архітектуру та підходи до розроблення REST API, а також визначити вимоги до узгодженості даних між компонентами;
3. проаналізувати методи реалізації рольового керування доступом і на цій основі обгрунтувати модель розмежування прав користувачів у системі, враховуючи різні ролі учасників і потребу в контролі критичних операцій;
4. спроєктувати структуру бази даних, що забезпечує коректне зберігання інформації про користувачів, послуги, розклад, бронювання та повідомлення, а також підтримує цілісність даних і відстежуваність ключових подій;

5. розробити механізм бронювання, який враховує часові обмеження та доступність послуг і зменшує ризик конфліктів шляхом перевірки перетину часових інтервалів, зокрема в умовах паралельних запитів;
6. реалізувати систему повідомлень, що забезпечує автоматизоване інформування користувачів про події в системі та підтримує обмін повідомленнями між клієнтами і провайдерами послуг, у тому числі для уточнення деталей бронювання;
7. впровадити технології обміну даними в реальному часі з метою забезпечення оперативної доставки повідомлень без потреби ручного оновлення сторінки, а також визначити принципи обробки подій на стороні клієнта і сервера;
8. здійснити програмну реалізацію веб-платформи із застосуванням сучасних інструментів і технологій розроблення, забезпечивши узгодженість між інтерфейсом користувача, серверною логікою та сховищем даних;
9. провести тестування розробленої системи для перевірки її функціональності, надійності та відповідності визначеним вимогам, включаючи перевірку сценаріїв конфліктного бронювання та коректності роботи механізмів сповіщення.

**Об’єктом дослідження** є процеси організації та керування онлайн-бронюванням послуг у веб-орієнтованих інформаційних системах. Зазначені процеси охоплюють взаємодію між користувачами та провайдерами послуг, керування розкладом, обробку запитів на бронювання, а також інформаційну підтримку учасників системи через повідомлення про зміни та підтвердження дій. Додатково в межах об’єкта доцільно розглядати організаційні аспекти, пов’язані з розподілом відповідальності між сторонами та мінімізацією операційних втрат, що виникають через несвоєчасне оновлення інформації.

**Предметом дослідження** є методи та засоби розроблення веб-платформи для онлайн-бронювання послуг із реалізацією рольового доступу та системи повідомлень у режимі реального часу. До предметної області також належать алгоритми перевірки доступності часових слотів, механізми уникнення

конфліктів бронювання та технології, які забезпечують миттєву передачу даних між клієнтом і сервером у подієво-орієнтованому форматі. Крім того, до предмета дослідження логічно віднести питання узгодження станів бронювання і повідомлень, аби комунікаційний модуль відображав фактичні зміни в системі, а не лише інтерфейсні події.

Таким чином, у межах даної роботи розглядається комплексне завдання створення веб-платформи, що інтегрує бронювання, керування користувачами та систему повідомлень і забезпечує узгоджену взаємодію між усіма учасниками процесу. Реалізація такої системи дозволяє автоматизувати процедури запису на послуги, водночас підвищуючи рівень комунікації та зменшуючи кількість ситуацій, у яких якість обслуговування погіршується через інформаційні затримки. У підсумку очікуваним результатом є більш керований і прозорий процес взаємодії, у якому технологічні механізми підтримують практичні потреби користувачів і провайдерів, а ефективність сервісу визначається не лише наявністю функцій, але й узгодженістю їхньої роботи в реальних умовах експлуатації.

**Апробація результатів дослідження.** Результати дослідження не були предметом обговорення на наукових конференціях та не публікувалися у наукових виданнях.

# РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

## 1.1 Особливості систем онлайн-бронювання послуг

Системи онлайн-бронювання послуг розглядаються як суттєвий елемент сучасних інформаційних технологій, оскільки вони забезпечують автоматизацію процедур запису користувачів на отримання визначених сервісів у цифровому середовищі з передбачуваними правилами доступу. У найзагальнішому трактуванні система бронювання постає як програмно-інформаційний комплекс, призначений для вибору, резервування та подальшого керування доступом до ресурсів або послуг у заданий часовий проміжок. Її функціонування грунтується на узгодженій взаємодії клієнтської частини (користувацького інтерфейсу), що забезпечує подання запитів і відображення станів, та серверної частини, яка виконує обробку запитів, перевіряє наявність вільних інтервалів і підтримує збереження даних у базі. Характерною рисою актуальних рішень є інтеграція з веб-технологіями, що на практиці зменшує залежність доступу до сервісів від географічного розташування користувача, класу пристрою та локального програмного оточення, а також уможливлює стандартизовані сценарії взаємодії через мережу [20].

Сфера використання систем онлайн-бронювання є різноплановою та охоплює широкий спектр предметних областей, у межах яких наявний дефіцитний ресурс і часові обмеження на його використання. Найбільш типовою сферою вважається туристична індустрія, де відповідні системи застосовуються для резервування готелів, транспортних квитків і супутніх туристичних послуг, часто з потребою синхронізації між декількома постачальниками. Водночас аналогічні підходи активно впроваджуються в охороні здоров’я (запис до лікарів і планування прийомів), в освіті (реєстрація на заняття, консультації, екзаменаційні слоти), у сервісній сфері (салони краси, ремонтні роботи, консультаційні послуги), а також у корпоративному середовищі, де об’єктами

бронювання стають переговорні кімнати, транспорт або спільне обладнання. Попри подібність базових функцій у зазначених доменах, реалізація таких систем закономірно відрізняється: на практиці варіюють вимоги до підтвердження запису, правила скасування, політики оплати, допустима тривалість інтервалів, а також ступінь регламентованості процесу обслуговування, що визначається специфікою предметної області та організаційними обмеженнями.

Основний принцип роботи систем онлайн-бронювання полягає в організації керованого доступу до обмеженого ресурсу або послуги з урахуванням часових рамок, які зумовлюють конкуренцію між запитами. Користувач через інтерфейс системи обирає потрібну послугу, визначає бажаний час її отримання, після чого система здійснює перевірку доступності відповідного ресурсу з огляду на наявні записи. За відсутності конфліктів формується запис про бронювання, який зберігається в базі даних та використовується надалі для внесення змін, підтвердження або скасування, а також для аналітики завантаженості. Водночас критичною складовою є підтримання цілісності даних і запобігання ситуаціям, коли декілька користувачів паралельно намагаються зарезервувати один і той самий часовий інтервал. Для цього використовуються процедури контролю перетину часових проміжків, що дозволяють встановити факт накладання двох бронювань. Зокрема, якщо початок одного інтервалу є меншим за кінець іншого, а його кінець є більшим за початок іншого, такі інтервали трактуються як такі, що перетинаються, і відповідне бронювання не може бути виконане [11]. У прикладних реалізаціях ця логіка часто доповнюється транзакційними механізмами або блокуваннями на рівні сховища даних, аби мінімізувати ризики умов перегонів під час високої конкурентності запитів [21].

Центральним поняттям більшості систем онлайн-бронювання є тайм-слот, або часова комірка, що інтерпретується як інтервал часу, доступний для резервування в межах певного ресурсу чи виконавця. Тайм-слоти можуть задаватися фіксованими кроками (наприклад, по 30 або 60 хвилин), що спрощує

відображення розкладу й уніфікує правила перевірки, або формуватися динамічно, коли тривалість визначається параметрами конкретної послуги, технологічними паузами, вимогами до підготовки та доступністю виконавця. Запровадження тайм-слотної моделі структурує процес бронювання, зменшує складність перевірки доступності та сприяє раціональнішому використанню ресурсів, зокрема в умовах нерівномірного попиту. Додатково така модель є зрозумілою для користувачів, які отримують прозорий вибір часових вікон, і зручною для провайдерів послуг, оскільки полегшує нормування робочого часу та знижує ймовірність накладення записів за рахунок чітко визначених меж обслуговування.

Не менш значущою складовою систем онлайн-бронювання є розмежування ролей користувачів, що зазвичай реалізується на основі моделі рольового керування доступом. У межах цієї моделі кожному користувачеві призначається роль, яка визначає допустимі операції та рівень доступу до даних і функціональних модулів системи. Як підкреслюють сучасні дослідження, рольове керування доступом сприяє впорядкуванню контролю прав, полегшує адміністрування та загалом підвищує безпекові характеристики інформаційної системи [9]. У контексті бронювання часто виокремлюють принаймні кілька ролей: неавторизованого користувача, який обмежується переглядом інформації; клієнта, що створює та керує власними бронюваннями; провайдера послуг, відповідального за формування розкладу, підтвердження записів і обробку змін; адміністратора, який здійснює системне керування, налаштування довідників і політик доступу. Такий підхід забезпечує узгодженість взаємодії між учасниками процесу, оскільки права та відповідальність формалізуються у вигляді відтворюваних правил, а критичні операції можуть супроводжуватися журналюванням та аудитом.

З технічної точки зору сучасні системи онлайн-бронювання зазвичай реалізуються як веб-орієнтовані застосунки з клієнт-серверною архітектурою, де розподіл відповідальностей є принциповим для підтримання керованості та масштабованості. Серверна частина інкапсулює бізнес-логіку, включно з

перевіркою доступності, обробкою запитів, управлінням станами бронювань і доступом до даних, тоді як клієнтська частина забезпечує взаємодію з користувачем, валідацію введення та відображення актуального стану розкладу. Для побудови серверної логіки нерідко застосовують сучасні фреймворки, зокрема Express у середовищі Node.js, який описується як гнучкий і мінімалістичний інструментарій, придатний для реалізації веб-застосунків різного рівня складності [5]. Зберігання даних здебільшого здійснюється в реляційних базах даних, де принципове значення має коректне проєктування схеми: зокрема, вибір типів даних для часових значень та інтервалів, визначення обмежень цілісності, індексування полів, що беруть участь у пошуку конфліктів, і підтримка операцій, які гарантують узгодженість даних про бронювання [12]. Практична ефективність системи в таких умовах значною мірою залежить від того, наскільки послідовно поєднані логіка застосунку та можливості СУБД для контролю конкурентного доступу.

Таким чином, системи онлайн-бронювання послуг можуть бути описані як комплексні інформаційні системи, у яких поєднуються механізми управління часовими ресурсами, обробка користувацьких запитів і організація взаємодії між ролями з різними рівнями повноважень. Вони виконують помітну функцію в процесах цифровізації, пропонуючи інструмент для впорядкованої організації надання послуг і зменшення транзакційних витрат часу як для користувачів, так і для провайдерів. Водночас подальший розвиток таких систем потребує врахування не лише базового функціоналу, а й питань синхронізації даних у розподілених середовищах, масштабованості під навантаженням, а також інтеграції з сучасними комунікаційними технологіями та зовнішніми сервісами, що буде детально розглянуто в наступних підрозділах.

## 1.2 Архітектурні особливості веб-платформ бронювання та засобів комунікації

Сучасні веб-платформи, призначені для онлайн-бронювання послуг, здебільшого проєктуються на засадах клієнт-серверної архітектури, яку в науковій та інженерній літературі розглядають як одну з базових моделей побудови інформаційних систем [22]. У межах цієї моделі фіксується функціональний поділ між клієнтською і серверною складовими: клієнтська частина орієнтована на формування та відтворення інтерфейсу, а також на організацію взаємодії з користувачем; серверна частина відповідає за приймання та оброблення запитів, виконання бізнес-правил предметної області і контрольований доступ до даних. Такий розподіл, як правило, підвищує керованість розвитку системи, оскільки компоненти можна змінювати та масштабувати відносно незалежно; водночас він створює передумови для модульності й локалізації відповідальностей. На практичному рівні серверна частина часто реалізується за допомогою веб-фреймворків, зокрема Express, який забезпечує типові засоби маршрутизації та оброблення HTTP-запитів і тим самим знижує складність створення прикладних веб-сервісів [5].

Однією з центральних ланок клієнт-серверної взаємодії у веб-платформах є застосування прикладного програмного інтерфейсу, зокрема у межах архітектурного стилю REST (Representational State Transfer). REST API задає уніфікований механізм обміну представленнями ресурсів між клієнтом і сервером через HTTP-протокол, спираючись на поширені методи на кшталт GET, POST, PUT і DELETE, а також на домовленості щодо ідентифікації ресурсів і структури відповідей [23]. У системах бронювання REST-підхід використовується для реалізації типових операцій предметної області, серед яких отримання переліку доступних послуг, створення бронювань, зміна їхнього стану та керування користувачами й правами доступу. Водночас стандартизованість REST інтерпретується як чинник, що спрощує інтеграцію з зовнішніми сервісами (зокрема платіжними або календарними), полегшує

повторне використання компонентів і підтримує поступову еволюцію API без радикальної перебудови клієнтських застосунків.

Разом із тим традиційна модель взаємодії, що грунтується на HTTP, має низку обмежень, помітних у сценаріях, де критичною є актуальність даних у коротких часових інтервалах. У класичній схемі клієнт ініціює запит, сервер формує відповідь, після чого з'єднання завершується; таким чином, отримання оновленого стану потребує повторних звернень. Коли клієнт змушений періодично опитувати сервер для синхронізації, це створює додаткове навантаження на серверну інфраструктуру, збільшує частку «порожніх» відповідей і, залежно від частоти опитування, призводить або до зайвого трафіку, або до відчутних затримок у відображенні змін. У системах бронювання подібна інерційність може проявлятися у несвоєчасному оновленні відомостей про доступність тайм-слотів чи поточний статус бронювання; наслідком стають конфліктні ситуації (наприклад, конкурентні спроби забронювати один ресурс) і погіршення якості користувацької взаємодії, що особливо чутливо у високонавантажених або багатокористувацьких середовищах.

У цьому контексті доцільно розрізняти синхронну та асинхронну моделі взаємодії як різні способи організації обміну подіями та станами. Синхронна модель передбачає очікування відповіді на кожен запит і відтворює типовий режим роботи HTTP, коли клієнтський застосунок, по суті, пов'язує прогрес виконання з фактом отримання відповіді. Асинхронна модель, натомість, допускає відокремлення ініціювання дій від моменту отримання результату, а також підтримує сценарії, у яких передавання даних може бути ініційоване сервером у відповідь на подію без прямого запиту з боку клієнта. Саме така організація дедалі частіше використовується в сучасних веб-додатках, де від системи очікують низької затримки оновлень, більш природного реагування інтерфейсу на події та стійкості до коливань навантаження.

Одним із поширених і технологічно обгрунтованих підходів до реалізації асинхронної взаємодії у веб-додатках є використання WebSocket. На відміну від одноразових HTTP-обмінів, WebSocket забезпечує встановлення постійного

двостороннього каналу між клієнтом і сервером, який підтримує обмін даними в режимі наближеному до реального часу без необхідності повторного встановлення з’єднання для кожного повідомлення [3, 4]. Після початкової процедури узгодження параметрів з’єднання (handshake) подальша комунікація відбувається з меншими накладними витратами, що є практично значущим для застосунків із високою частотою оновлення та вимогами до низької затримки [1, 2]. У прикладному вимірі це означає, що система може швидше поширювати зміни стану ресурсів, а клієнтські інтерфейси – оперативніше відображати ці зміни без частого опитування серверної частини.

У площині онлайн-бронювання WebSocket розширює можливості організації взаємодії між учасниками процесу та дає змогу коректніше обслуговувати конкурентні сценарії. Зокрема, з’являються умови для реалізації механізмів негайного сповіщення про зміну статусу бронювання або про появу/зникнення доступних тайм-слотів, що набуває особливого значення тоді, коли кілька користувачів працюють з одним і тим самим ресурсом у близьких часових межах. Окрім цього, постійний канал зв’язку може бути використаний для побудови підсистеми повідомлень між клієнтами та надавачами послуг, що підвищує узгодженість дій (наприклад, уточнення деталей замовлення) і зменшує потребу в зовнішніх каналах комунікації. Практика також демонструє, що бібліотеки на кшталт Socket.IO можуть доповнювати базові можливості WebSocket, забезпечуючи сумісність із різними мережевими умовами, автоматичний вибір доступного механізму передавання та резервні сценарії у випадках, коли WebSocket є недоступним у конкретному середовищі [8].

Важливо зауважити, що асинхронна взаємодія не прив’язана до одного серверного середовища й може бути інтегрована у різні технологічні стеки з урахуванням їхніх моделей виконання та інструментарію. Наприклад, у Python-екосистемі для роботи з подіями в реальному часі застосовують Django Channels або FastAPI, які підтримують WebSocket і дозволяють будувати обробники подій та канали доставляння повідомлень з опорою на асинхронні механізми виконання [6, 7]. Така сумісність вказує на загальну переносність концепції та на

те, що вибір конкретних інструментів радше визначається вимогами до продуктивності, супровідності, досвідом команди та екосистемними обмеженнями, ніж принциповою придатністю підходу.

Отже, архітектура сучасних веб-платформ бронювання може розглядатися як поєднання усталених засад, зокрема клієнт-серверної моделі та REST API, із технологіями асинхронної взаємодії, орієнтованими на подієві оновлення. Використання WebSocket дає змогу частково компенсувати обмеження класичних HTTP-запитів, зменшити затримки в доставлянні змін стану та забезпечити оперативну синхронізацію між компонентами й учасниками системи. У цьому сенсі інтеграція механізмів бронювання з комунікаційними функціями в реальному часі може трактуватися як перспективний напрям розвитку веб-додатків, який створює підгрунтя для підвищення узгодженості взаємодій, зручності користування та конкурентоспроможності сервісів у багатокористувацьких сценаріях.

### 1.3 Аналіз сучасних платформ для бронювання послуг

В умовах поточної цифровізації сектору послуг спостерігається інтенсивне розповсюдження веб-платформ для онлайн-бронювання. Ці системи забезпечують автоматизацію процесів реєстрації користувачів, керування розкладом та оптимізацію взаємодії між клієнтами і постачальниками послуг. Їх застосування охоплює такі сектори, як освіта, консалтинг, медицина та індустрія краси, а також інші сфери, де критично важливою є координація часових ресурсів і оперативне інформування залучених сторін [24, 25]. З метою виявлення актуальних підходів до розробки систем бронювання, доцільним є проведення аналізу низки широко застосовуваних платформ, функціональні можливості яких релевантні до предметної області поточного дослідження.

Серед відомих платформ варто виділити Calendly, яка спеціалізується на автоматизації планування зустрічей та резервування часових інтервалів. Принцип функціонування даного сервісу базується на створенні користувачем

власного графіка доступності, з подальшим автоматичним відображенням системою виключно вільних часових слотів для клієнтів. Платформа забезпечує функціонал персональних облікових записів, інтеграцію з календарними сервісами, а також автоматичне генерування підтверджень і нагадувань про заплановані події. Простота використання та високий ступінь автоматизації сприяли позиціонуванню Calendly як одного з провідних інструментів для організації запису на послуги [15]. Сторінку бронювання платформи Calendly наведено на рисунку 1.1.

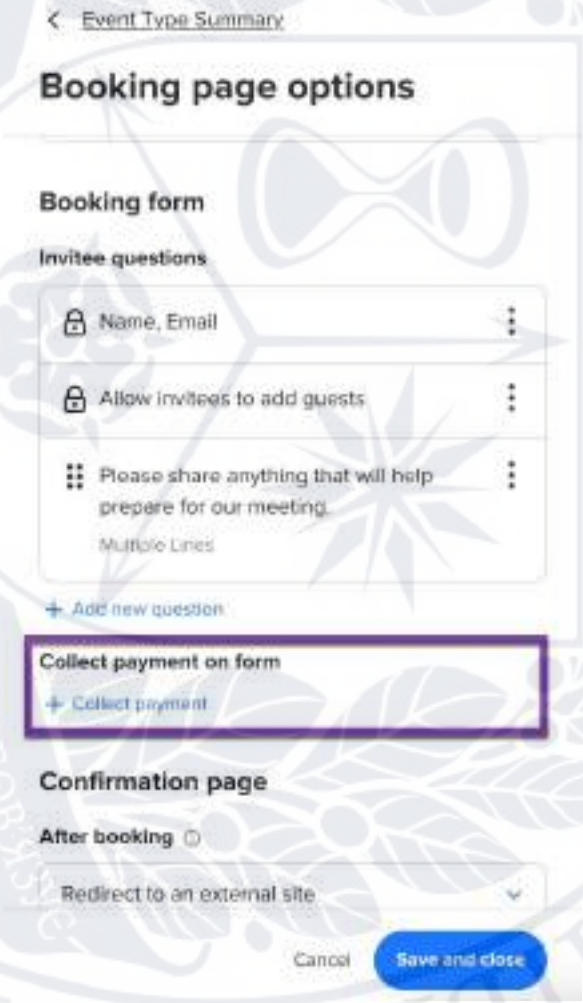


Рисунок 1.1 – Інтерфейс платформи Calendly

Платформа SimplyBook.me, розроблена для підприємств сфери послуг, надає розширений спектр функціональних можливостей. Система підтримує

керування послугами, розкладом персоналу, базою клієнтів та процесами бронювання через відповідний веб-інтерфейс. Користувачам надається можливість здійснювати запис на доступні часові слоти, переглядати історію попередніх бронювань та керувати поточними записами через особистий кабінет. Додатково, платформа реалізує механізми автоматичних сповіщень та нагадувань, що оптимізує взаємодію між клієнтами та постачальниками послуг [16]. Інтерфейс системи бронювання SimplyBook.me наведено на рисунку 1.2.

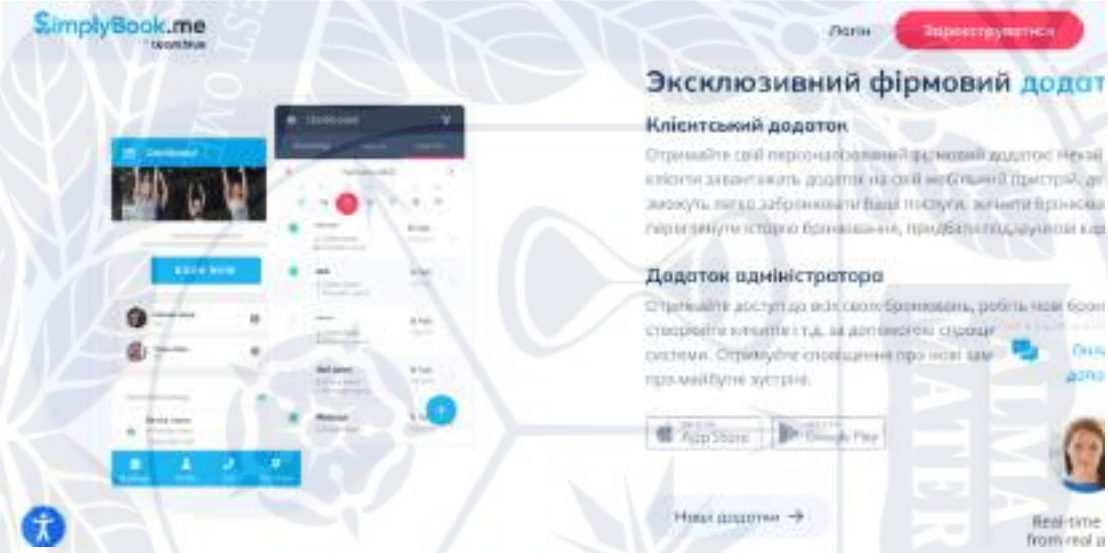


Рисунок 1.2 – Інтерфейс системи бронювання SimplyBook.me

Серед інших популярних рішень слід зазначити Booksy, яка активно застосовується в індустрії краси та сегменті персональних послуг. Ця платформа інтегрує функції пошуку фахівців, бронювання послуг та управління клієнтською базою. Користувачі мають змогу переглядати доступні послуги, обирати конкретного постачальника та бронювати вільні часові слоти. Персональний кабінет забезпечує моніторинг активних бронювань, доступ до історії відвідувань та отримання сповіщень про зміни статусу запису. Особливий акцент зроблено на мобільній взаємодії та своєчасному інформуванні користувачів [17]. Інтерфейс платформи Booksy наведено на рисунку 1.3.

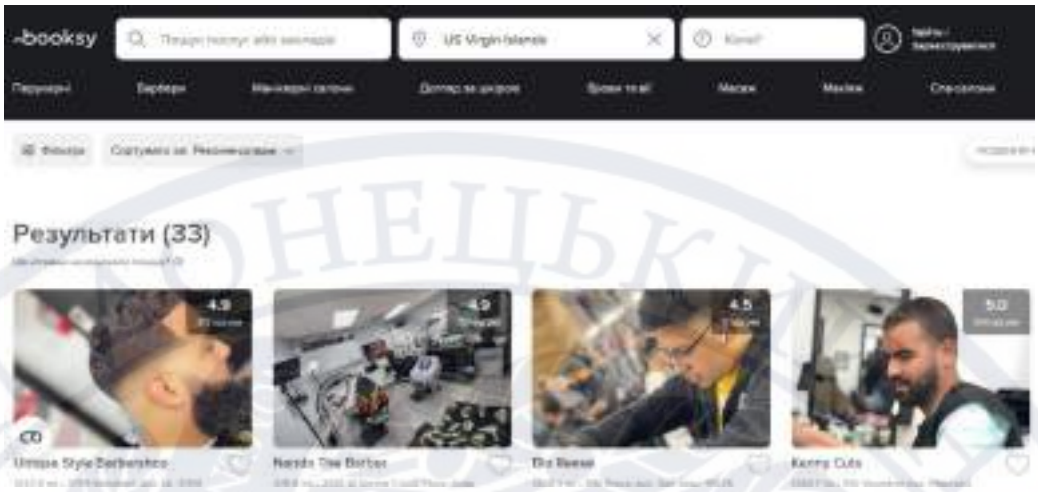


Рисунок 1.3 – Інтерфейс платформи Booksy

Платформа Setmore розроблена для організації запису клієнтів на різноманітні послуги та підтримує багатокористувацький режим функціонування. Система забезпечує формування індивідуальних розкладів для персоналу, керування переліком послуг та контроль за доступністю часових інтервалів. За допомогою персонального кабінету користувачі можуть переглядати деталі бронювань, вносити корективи до записів та отримувати актуальні сповіщення. Крім того, платформа реалізує інтеграцію з календарними та іншими сервісами планування [18]. Інтерфейс платформи Setmore наведено на рисунку 1.4.

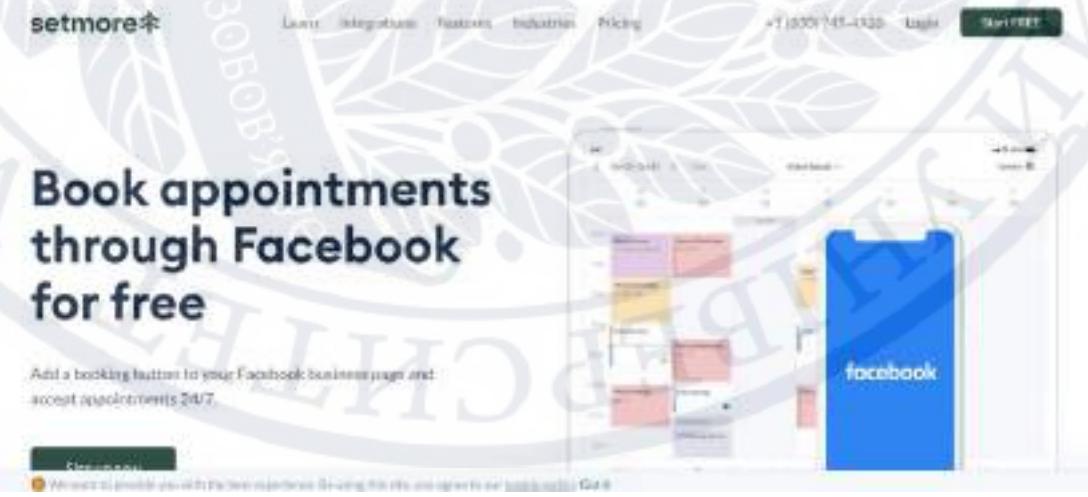


Рисунок 1.4 – Інтерфейс платформи Setmore

Серед спеціалізованих рішень також виокремлюється Acuity Scheduling, яка зосереджена на автоматизації процесів запису та управління робочими графіками. Система надає можливість гнучкого налаштування правил бронювання, тривалості послуг, доступності часових інтервалів та механізмів автоматичного підтвердження записів. Користувачам доступний особистий кабінет для перегляду власних бронювань та отримання сповіщень про зміни або нагадувань щодо майбутніх зустрічей. Гнучкість конфігурації платформи забезпечує її адаптивність для застосування в різних галузях діяльності [19]. Приклад сторінки Acuity Scheduling наведено на рисунку 1.5.

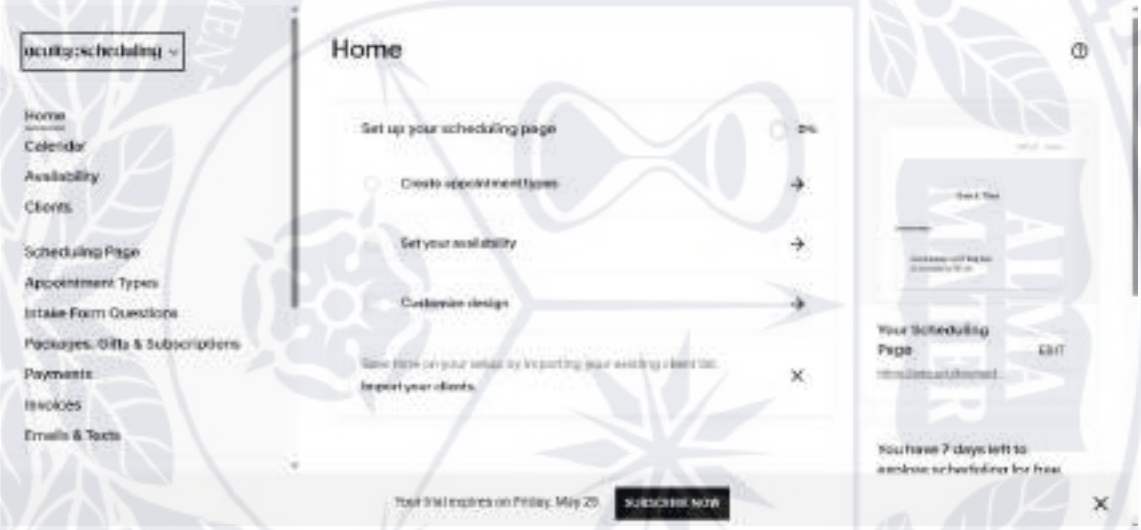


Рисунок 1.5 – Інтерфейс платформи Acuity Scheduling

Проведений аналіз демонструє, що сучасні системи онлайн-бронювання, незалежно від їхньої сфери застосування, реалізують аналогічний набір базових функцій. Ці функції включають створення та управління бронюваннями, застосування персоналізованих облікових записів, підтримку розкладу та механізми інформування користувачів про системні події. При цьому специфіка реалізації зазначених можливостей може варіюватися залежно від предметної області та обраних технологічних підходів. Аналіз представлених платформ дозволяє сформувати уявлення про актуальні методи побудови систем

бронювання та створює методологічну основу для подальшого вивчення їх функціональних аспектів та обмежень у наступному розділі.

### 1.4 Аналіз функціональних можливостей та недоліків існуючих рішень

Аналіз сучасних платформ онлайн-бронювання, зокрема Calendly, SimplyBook.me, Booksy, Setmore та Acuity Scheduling, дозволяє виявити характерні функціональні можливості та ключові обмеження. Ці системи орієнтовані на автоматизацію процесів запису, управління розкладом та організацію взаємодії між клієнтами й надавачами послуг.

Усі розглянуті платформи реалізують базову функціональність онлайн-бронювання, надаючи користувачам можливість переглядати послуги, обирати часові інтервали та створювати бронювання через веб-інтерфейс. Calendly та Acuity Scheduling зосереджені на автоматизованому плануванні зустрічей і синхронізації з зовнішніми календарями [15, 19]. SimplyBook.me і Setmore пропонують розширені інструменти керування послугами та персоналом [16, 18]. Платформа Booksy інтегрує механізми бронювання із засобами пошуку спеціалістів та управління клієнтською базою [17]. Таким чином, бронювання виступає центральним елементом усіх досліджених систем.

Важливою складовою є персоналізовані кабінети користувачів, що забезпечують централізоване управління записами та графіком роботи. Усі платформи підтримують автоматичне надсилання повідомлень про події (створення, підтвердження, скасування) за допомогою електронної пошти або інтеграції з календарними сервісами [15, 16, 18]. Ці повідомлення переважно виконують інформаційну функцію.

Незважаючи на широкий набір функціональних можливостей, аналіз виявляє низку характерних обмежень. Однією з найбільш помітних проблем є недостатня підтримка взаємодії в режимі реального часу. Більшість платформ використовують традиційну модель обміну даними, що призводить до затримки відображення актуальної інформації про стан бронювання або доступність

часових слотів [1, 2, 26, 27]. Це особливо помітно в багатокористувацьких сценаріях, де виникають проблеми синхронізації даних про доступність та ризик дублювання запитів [21].

Іншим обмеженням є недостатній рівень інтерактивної комунікації між учасниками процесу бронювання. Автоматизовані повідомлення не формують повноцінного внутрішнього комунікаційного середовища, змушуючи користувачів переходити до зовнішніх каналів зв'язку для уточнень. Це призводить до розпорошення інформації та ускладнює відстеження історії взаємодії.

Підсумовуючи, сучасні платформи онлайн-бронювання успішно реалізують базові механізми, проте їх функціональність здебільшого орієнтована на традиційну модель з відкладеним оновленням даних. Основними обмеженнями є недостатня підтримка режиму реального часу, обмежені можливості внутрішньої комунікації та складність забезпечення синхронізації даних у конкурентних сценаріях. Ці недоліки обгрунтовують доцільність розроблення веб-платформи, яка інтегрує технології WebSocket та внутрішню систему сповіщень для оперативної синхронізації даних та миттєвого інформування користувачів [3, 4, 36].

Короткий і наочний аналіз характеристик цих сервісів представлено у таблиці 1.1.

Таблиця 1.1 Аналіз характеристик сервісів-конкурентів

| | **Calendly** | **SimplyBook** | **Booksy** | **Setmore** | **Acuity** |
|---|---|---|---|---|---|
| Тайм-слоти | Так | Так | Так | Так | Так |
| Notification system | Так | Так | Так | Так | Так |
| Realtime-оновлення | Обмежено | Частково | Частково | Обмежено | Обмежено |

| | | | | | |
|---|---|---|---|---|---|
| WebSocket підтримка | Ні | Ні | Частково | Ні | Ні |
| RBAC та ролі користувачів | Частково | Так | Так | Так | Частково |

### 1.5 Визначення цільової аудиторії та сценаріїв використання системи

На етапі проєктування веб-платформи для онлайн-бронювання принципового значення набуває визначення цільової аудиторії та опис типових сценаріїв застосування системи. Ідентифікація груп користувачів дає змогу обгрунтовано сформулювати функціональні вимоги, зменшити ризик невідповідності реалізованих можливостей реальним потребам і водночас підтримати узгодженість архітектурних рішень. У практичному вимірі це також впливає на якість користувацького досвіду. Для розроблюваної платформи обгрунтовано виокремити три базові категорії користувачів: клієнтів, провайдерів послуг та адміністраторів. Кожна з них відрізняється мотивацією, контекстом використання та набором дозволених дій, що узгоджується з підходами, описаними в літературі щодо моделювання ролей у інформаційних системах [1, 28].

Клієнти становлять центральну групу користувачів, оскільки саме вони ініціюють ключовий для платформи бізнес-процес – бронювання. До цієї категорії належать особи, які прагнуть отримати доступ до переліку сервісів без додаткових комунікаційних бар’єрів, із можливістю швидко зіставити альтернативи за часом, параметрами або умовами надання. З погляду цілей користувача, основна послідовність дій включає пошук релевантної послуги, вибір доступного часового інтервалу та створення запису. Відповідно, система має забезпечувати не лише зрозумілу навігацію й однозначні формулювання елементів інтерфейсу, а й коректну візуалізацію доступності тайм-слотів, що

зменшує когнітивне навантаження та кількість помилкових кроків. Окрім операції створення бронювання, суттєвими є засоби керування власними записами: перегляд, за потреби – скасування або перенесення відповідно до правил платформи, а також доступ до історії звернень із фіксацією статусів. Така функціональність підвищує прозорість процесу та підтримує довіру до сервісу, оскільки користувачеві важливо розуміти, чи прийнято запит, чи очікується підтвердження, чи виникли обмеження. У сучасних умовах також зростає значущість каналів оперативної комунікації з провайдером для уточнення деталей (наприклад, підготовки до візиту, вимог до документів, зміни умов або часу). Наявність таких механізмів зазвичай розглядається як чинник, що сприяє задоволеності користувачів і зниженню частки невдалих бронювань [2, 29].

Провайдери послуг формують другу ключову групу, для якої система виступає інструментом організації робочих процесів і керування взаємодією з клієнтами. До провайдерів можуть належати як окремі фахівці, так і організації, що надають послуги різної природи, зокрема у сферах медицини, освіти, консалтингу або побутових сервісів. Їхній інтерес до платформи пов’язаний із необхідністю підтримувати актуальний розклад, коректно відображати доступність, уникати накладок у записах та забезпечувати дисциплінована обробку запитів. У цьому контексті основним завданням провайдера є формування пропозиції (перелік послуг і параметрів надання), визначення часових слотів та управління заявками на бронювання. Для цього система має надавати засоби створення й редагування описів послуг, налаштування тривалості прийомів, правил доступності, керування календарем, а також інструменти для підтвердження або відхилення заявок. Додатково важливо забезпечити доступ до інформації про клієнтів у межах, виправданих функціональною потребою та обмежених політиками конфіденційності, зокрема для коректної ідентифікації запису та узгодження деталей. Оперативність роботи провайдера значною мірою залежить від механізмів сповіщення про нові бронювання чи зміни, а також від можливості обміну повідомленнями з

клієнтами без переходу до зовнішніх каналів, що зазвичай зменшує фрагментацію комунікації та підвищує керованість процесу [2].

Третьою категорією є адміністратори, які забезпечують керованість платформи на рівні системних політик і технологічної стабільності. Їхня роль виходить за межі індивідуальних операцій бронювання та полягає у підтриманні коректного функціонування сервісу, управлінні обліковими записами, а також контролі за дотриманням правил використання. На практиці це передбачає наявність розширених прав доступу, які уможливлюють створення, редагування або видалення облікових записів, реагування на інциденти, блокування користувачів у разі порушень, модерацію вмісту та моніторинг активності. Окремим напрямом є аналіз агрегованих даних щодо використання платформи: статистика бронювань, частота відхилень, завантаженість провайдерів, типові часові піки, що може бути підставою для рішень щодо масштабування, оптимізації інтерфейсу або уточнення бізнес-правил. Вказаний розподіл повноважень доцільно співвідносити з моделлю контролю доступу на основі ролей (RBAC), яка у сучасних інформаційних системах використовується для формалізації прав і обмеження операцій відповідно до функціональних потреб ролі [1].

Визначення цільових груп користувачів логічно пов’язане з формуванням сценаріїв використання, які фіксують типові послідовності дій та реакції системи на події. Одним із базових сценаріїв виступає бронювання послуги, що ініціюється клієнтом. У межах цього сценарію користувач обирає послугу, переглядає доступні часові інтервали та формує запит на запис. З боку платформи це має супроводжуватися перевіркою доступності обраного слоту, фіксацією даних у сховищі та встановленням відповідного статусу бронювання. З огляду на характер задачі, у системі можуть застосовуватися підходи до виявлення конфліктів, зокрема перевірка перетину часових інтервалів або контроль унікальності записів для конкретного ресурсу в заданий період. Такі механізми мають значення для забезпечення узгодженості даних і запобігання ситуаціям подвійного резервування [3].

Наступним суттєвим сценарієм є підтвердження бронювання, що передбачає взаємодію між клієнтом і провайдером на етапі узгодження. Після створення запису провайдер отримує повідомлення або інший сигнал про новий запит і здійснює рішення щодо його прийняття або відхилення відповідно до власної політики й актуального навантаження. У разі підтвердження платформа оновлює стан бронювання та інформує клієнта, забезпечуючи зворотний зв’язок. Важливість цього сценарію полягає у зменшенні невизначеності для користувача та у формуванні передбачуваного процесу, де статуси мають однозначну інтерпретацію, а повідомлення – мінімізують ризик пропущених змін.

Окремого розгляду потребує сценарій обміну повідомленнями, що підтримує комунікацію між клієнтами та провайдерами у межах платформи. У цьому сценарії користувачі надсилають повідомлення для уточнення деталей бронювання, узгодження умов надання послуги, обговорення підготовчих вимог або вирішення ситуацій, що виникають у процесі взаємодії. Наявність такого каналу комунікації, як правило, розглядається як необхідний компонент сучасних веб-сервісів, оскільки він підвищує гнучкість взаємодії та зменшує залежність від зовнішніх засобів зв’язку. Особливої ваги набуває підтримка обміну в режимі, наближеному до реального часу, адже це скорочує затримки у реагуванні на зміни – наприклад, при потребі швидкого перенесення запису чи уточнення інформації напередодні прийому. У відповідних дослідженнях зазначається, що використання WebSocket дає можливість підтримувати постійне двостороннє з’єднання між клієнтом і сервером, зменшуючи накладні витрати та затримки, характерні для періодичних HTTP-запитів [4, 26].

Отже, окреслення цільової аудиторії та типових сценаріїв використання створює підстави для формалізації функціональних вимог до веб-платформи онлайн-бронювання. Виокремлення ролей та опис їхньої взаємодії із системою дозволяє пов’язати очікування користувачів із конкретними програмними механізмами, а також підготувати основу для подальших етапів проєктування. Зокрема, така деталізація є передумовою розроблення діаграм варіантів використання (Use Case), що буде розглянуто в наступних розділах роботи.

## 1.6 Формування функціональних та нефункціональних вимог

Завершальним етапом аналізу предметної області є формування функціональних і нефункціональних вимог до веб-платформи онлайн-бронювання послуг. На основі результатів аналізу існуючих рішень, визначення цільової аудиторії та опису основних сценаріїв використання було сформовано перелік вимог, які визначають функціональність майбутньої системи та її якісні характеристики.

Функціональні вимоги описують перелік операцій, які повинна підтримувати платформа для забезпечення повноцінного процесу бронювання послуг. Однією з базових вимог є підтримка реєстрації та авторизації користувачів. Система повинна забезпечувати створення облікових записів, автентифікацію користувачів та надання доступу до функціональності відповідно до їх ролей. Для реалізації контролю доступу передбачено використання моделі рольового керування доступом (RBAC), у межах якої користувач може належати до однієї з ролей: клієнт, провайдер послуг або адміністратор [9].

Ключовою функціональною вимогою є реалізація механізму бронювання. Система повинна надавати можливість перегляду доступних послуг і часових слотів, створення бронювань, зміни їх статусів та перегляду історії записів. Для забезпечення коректності роботи необхідно виконувати перевірку доступності обраного слоту та запобігати створенню конфліктних бронювань шляхом контролю перетину часових інтервалів [11]. Також система повинна підтримувати життєвий цикл бронювання через набір статусів: pending, confirmed, cancelled та completed.

Окремою функціональною вимогою є реалізація підсистеми внутрішніх повідомлень (In-App Notifications). Система повинна автоматично створювати повідомлення під час виникнення ключових подій, зокрема створення, підтвердження або скасування бронювання. Користувачі повинні мати

можливість переглядати список отриманих повідомлень та позначати їх як прочитані.

Важливою особливістю платформи є підтримка взаємодії в режимі реального часу. Після зміни статусу бронювання або створення нового повідомлення відповідні дані повинні автоматично відображатися в інтерфейсі користувачів без необхідності оновлення сторінки. Для реалізації цієї вимоги доцільно використовувати технологію WebSocket та бібліотеку Socket.IO, які забезпечують постійний двосторонній обмін даними між клієнтом і сервером [1-4, 8].

Поряд із функціональними вимогами були сформовані нефункціональні вимоги, які визначають якісні характеристики системи. Однією з основних вимог є продуктивність. Платформа повинна забезпечувати швидке виконання операцій авторизації, бронювання та отримання повідомлень, а також коректно працювати за одночасної взаємодії кількох користувачів.

Не менш важливою є вимога безпеки. Система повинна забезпечувати захист персональних даних користувачів, підтримувати автентифікацію за допомогою JWT-токенів та обмежувати доступ до функціональності відповідно до ролі користувача [30, 31, 33].

Окрему увагу приділено вимозі підтримки роботи в режимі реального часу. Система повинна забезпечувати оперативну синхронізацію даних між клієнтом і сервером, що особливо важливо під час зміни доступності часових слотів і статусів бронювань [1, 2, 3, 36].

Ще однією важливою нефункціональною вимогою є зручність використання. Інтерфейс платформи повинен бути інтуїтивно зрозумілим, забезпечувати швидкий доступ до основних функцій і мінімізувати кількість дій, необхідних для створення бронювання або перегляду інформації.

Таким чином, сформований набір функціональних і нефункціональних вимог визначає основні характеристики веб-платформи онлайн-бронювання послуг та створює основу для її подальшого проєктування і реалізації. Особлива увага приділяється забезпеченню коректної роботи механізму бронювання, рольовому контролю доступу, підтримці внутрішніх повідомлень та синхронізації даних у режимі реального часу, що відповідає поставленим цілям розроблення системи.

## ВИСНОВКИ ДО РОЗДІЛУ 1

У першому розділі було проведено аналіз предметної області систем онлайн-бронювання послуг та досліджено сучасні підходи до побудови веб-платформ даного типу. Розглянуто особливості організації процесу бронювання, принципи роботи з часовими слотами, рольове розмежування доступу та механізми взаємодії між учасниками системи.

У результаті аналізу архітектурних особливостей сучасних веб-застосунків встановлено, що найбільш доцільним підходом для реалізації платформи є клієнт-серверна архітектура з використанням REST API для обробки основних операцій та технологій обміну даними в режимі реального часу для оперативної синхронізації інформації між користувачами.

Було здійснено огляд сучасних платформ бронювання послуг, зокрема Calendly, SimplyBook.me, Booksy, Setmore та Acuity Scheduling. Аналіз показав, що всі розглянуті системи реалізують базові механізми створення та керування бронюваннями, проте мають певні обмеження щодо підтримки режиму реального часу та внутрішньої комунікації між учасниками процесу.

На підставі проведеного аналізу визначено основні категорії користувачів майбутньої системи – клієнтів, провайдерів послуг та адміністраторів, а також описано ключові сценарії їхньої взаємодії з платформою. Крім того, сформовано функціональні та нефункціональні вимоги до програмного продукту, що включають механізми авторизації, бронювання, рольового доступу, внутрішніх повідомлень та синхронізації даних у режимі реального часу.

Отримані результати становлять теоретичну та методологічну основу для подальшого проєктування веб-платформи онлайн-бронювання послуг, визначення її архітектури, структури бази даних та реалізації основних програмних компонентів, що розглядаються в наступному розділі.

# РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБ-ПЛАТФОРМИ ДЛЯ ОНЛАЙН-БРОНЮВАННЯ

## 2.1 Проєктування загальної архітектури веб-платформи для онлайн-бронювання

Проєктування архітектури веб-платформи розглядається як один із визначальних етапів створення інформаційної системи, оскільки саме прийняті архітектурні рішення суттєво впливають на масштабованість, продуктивність, надійність, а також на потенціал подальшої еволюції програмного продукту. Для систем онлайн-бронювання, яким властиві паралельна робота значної кількості користувачів, обробка даних із часовою чутливістю та потреба в оперативному обміні повідомленнями, принциповим є вибір архітектури, що дає змогу узгоджено поєднувати синхронні та асинхронні механізми взаємодії компонентів.

У межах цієї роботи пропонується застосування класичної клієнт-серверної архітектури, яка фактично стала типовим підходом для сучасних веб-застосунків. Вона передбачає поділ системи на кілька рівнів: клієнтський (frontend), серверний (backend) та рівень зберігання даних (database). Це, як правило, створює передумови для масштабування, підтримки змін та поступового розширення функціональних можливостей системи. Також, для підтримання взаємодії в режимі, наближеному до реального часу, до загальної структури вводиться окремий комунікаційний рівень на основі WebSocket, що дає змогу організувати подієво-орієнтований обмін даними [1-3, 36].

У загальному вигляді архітектура системи може бути представлена схемою, як на рис. 2.1.

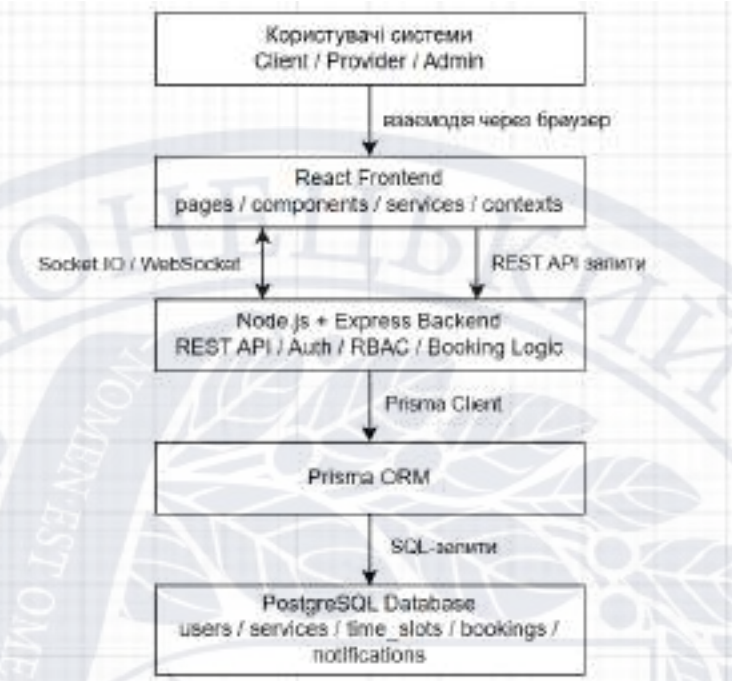


Рис. 2.1 – Загальна архітектура веб-платформи онлайн-бронювання

Клієнтський компонент реалізується із застосуванням бібліотеки React, що використовується для побудови інтерактивних односторінкових застосунків. Frontend забезпечує взаємодію користувача із системою, візуалізацію даних і ініціювання запитів до серверного компонента. Інтерфейс охоплює такі основні модулі: реєстрація й авторизація, каталог послуг, сторінка бронювання з вибором часових інтервалів, особистий кабінет, а також інтерфейс перегляду повідомлень і сповіщень. При цьому базові дані отримуються через REST API, тоді як зміни стану (зокрема надходження повідомлень або оновлення статусу бронювання) передаються через WebSocket-з’єднання, що зменшує затримки та дає змогу підтримувати актуальність інформації без перезавантаження сторінки [1, 3].

Серверний компонент реалізується у середовищі виконання Node.js із використанням фреймворку Express, який зазвичай описують як мінімалістичний і водночас достатньо гнучкий інструмент для побудови веб-застосунків [4, 32]. Backend зосереджує реалізацію бізнес-логіки: обробку запитів, керування ролями доступу, перевірку коректності операцій бронювання, формування часових інтервалів, а також взаємодію з базою даних. Обмін даними між frontend

і backend організовано через REST API, який відповідає принципам архітектурного стилю REST і використовує формат JSON як стандарт представлення даних.

Основні REST-ендпоінти системи включають:

- /auth/register – реєстрація користувача;
- /auth/login – авторизація;
- /services – отримання списку послуг;
- /slots – отримання доступних тайм-слотів;
- /bookings – створення та управління бронюваннями;
- /messages – робота з повідомленнями.

У межах традиційної клієнт-серверної моделі взаємодія між клієнтом і сервером зазвичай реалізується на основі HTTP-протоколу з моделлю «запит–відповідь». За такого підходу клієнт формує запит, сервер повертає відповідь, після чого з’єднання припиняється. Цей механізм є придатним для значної частини операцій, зокрема отримання переліку послуг, створення чи редагування бронювань, реєстрації користувачів тощо. Однак використання лише HTTP пов’язане з помітними обмеженнями: для підтримання актуальності стану системи клієнт змушений регулярно опитувати сервер, що може збільшувати затримки та підвищувати навантаження [1, 2].

Для зменшення зазначених обмежень у сучасних веб-застосунках поширеною є технологія WebSocket. На відміну від обміну через HTTP, WebSocket підтримує тривале з’єднання, завдяки чому сервер може ініціювати передачу даних без повторних запитів з боку клієнта [1–3]. Така властивість є суттєвою для реалізації підсистеми повідомлень і своєчасного інформування про зміну стану бронювання. Для спрощення інтеграції та уніфікації обробки подій може застосовуватися бібліотека Socket.IO, яка додатково надає механізми керування подіями та резервні сценарії роботи в умовах мережевих обмежень [5]. У межах WebSocket-взаємодії в системі визначаються ключові події, такі як:

- booking_created – створення нового бронювання;

- booking_confirmed – підтвердження бронювання;
- new_message – надходження нового повідомлення;
- notification – генерація системного сповіщення.

Зазначені події відповідають подієво-орієнтованому підходу (event-driven architecture): зміни стану системи слугують тригером для відповідних реакцій клієнтського застосунку. У практичному вимірі це може підвищувати оперативність оновлення інтерфейсу та сприйману зручність роботи із системою.

Рівень зберігання даних реалізовано на основі реляційної СКБД PostgreSQL, що забезпечує надійне зберігання, підтримку складних запитів і механізми підтримання цілісності. Реляційна модель є придатною для формалізації сутностей (користувачі, послуги, бронювання, повідомлення) та їхніх зв’язків, які було розглянуто в попередньому підрозділі. Додатково PostgreSQL підтримує транзакції й обмеження цілісності; ці механізми є важливими для зменшення ризику конфліктів у ситуаціях одночасного створення бронювань [6, 7].

Для забезпечення безпеки та контролю доступу передбачається застосування автентифікації на основі JSON Web Token (JWT), що дає змогу організувати захищену передачу даних між клієнтом і сервером та обмежувати доступ до ресурсів з урахуванням ролей користувачів. Такий підхід узгоджується з концепцією рольового керування доступом (RBAC), розглянутою в попередньому підрозділі [8, 9, 33].

Властивістю запропонованого підходу є модульність і потенціал до масштабування: компоненти системи можуть розгортатися окремо та масштабуватися незалежно, що полегшує адаптацію до зростання кількості користувачів і збільшення обсягу даних. За потреби архітектуру можна доповнювати суміжними технологіями, зокрема Redis для кешування та роботи з чергами повідомлень, а також Docker для контейнеризації застосунку й спрощення процедур розгортання.

Отже, запропонована архітектура поєднує традиційні принципи клієнт-серверної взаємодії з інструментами асинхронного обміну даними. Поєднання REST API для реалізації основних сценаріїв і WebSocket для підтримання взаємодії в режимі реального часу формує гібридну модель, яка є придатною для веб-платформ онлайн-бронювання з вимогами до актуальності станів і чутливості до затримок.

## 2.2 Проєктування моделі користувачів та розмежування доступу

Одним із визначальних компонентів проєктування веб-платформи для онлайн-бронювання є побудова формалізованої моделі користувачів, що задає параметри взаємодії різних категорій учасників із системою та фіксує межі доступу до її функціональних можливостей. Для інформаційних систем, у яких обробляються персональні дані, фінансова інформація і виконуються операції підвищеної критичності (зокрема створення й супровід бронювань), питання контролю доступу набуває принципового значення. З огляду на це в роботі застосовано модель рольового керування доступом RBAC (Role-Based Access Control), у межах якої права визначаються не індивідуально, а відповідно до ролі користувача в системі.

У логіці RBAC кожному обліковому запису призначається роль, що задає множину допустимих дій. Така організація дає змогу керувати доступом централізовано, зменшує адміністративне навантаження та, як правило, підвищує рівень захищеності, оскільки відпадає потреба у точковому налаштуванні привілеїв для кожного користувача [9, 10, 34, 35]. Додатковою перевагою є адаптивність: зміни у функціональності можуть бути реалізовані через введення нових ролей або коригування наборів дозволів без перебудови всієї схеми доступів.

У межах веб-платформи, що розробляється, передбачено чотири базові ролі: неавторизований користувач (guest), зареєстрований користувач (client), постачальник послуг (provider) та адміністратор (admin). Така схема

узгоджується з типовими сценаріями для систем онлайн-бронювання і забезпечує зрозуміле розмежування повноважень між основними групами користувачів. Узагальнення основних сценаріїв взаємодії користувачів із системою доцільно подати у вигляді діаграми варіантів використання. Діаграма на рисунку 2.2 відображає ключові ролі системи та функції, доступні кожній із них.


Рисунок 2.2 – Діаграма варіантів використання веб-платформи онлайн-бронювання

Як видно з рисунка, гість має доступ лише до відкритих функцій системи, зокрема перегляду послуг, реєстрації та авторизації. Клієнт після входу до системи може створювати бронювання, переглядати доступні часові слоти,

скасовувати власні записи та отримувати повідомлення. Провайдер послуг отримує доступ до керування послугами, часовими слотами та бронюваннями клієнтів. Адміністратор виконує функції загального керування користувачами системи. Такий розподіл сценаріїв відповідає RBAC-моделі та забезпечує логічне розмежування доступу між основними категоріями користувачів.

З технічного погляду RBAC реалізується шляхом збереження ролі користувача у відповідному полі бази даних (наприклад, role у таблиці users) і застосування процедур контролю доступу в серверній логіці. У контексті серверної розробки, зокрема при використанні фреймворку Express, це може бути організовано через middleware, який перевіряє роль перед виконанням визначених операцій [5, 32]. Така організація сприяє централізації правил доступу та зменшує ризик дублювання фрагментів коду в різних частинах застосунку.

У підсумку застосування RBAC у поєднанні з однозначно визначеними ролями та таблицею дозволів забезпечує впорядковану взаємодію користувачів із системою, підтримує належний рівень безпеки та спрощує подальше розширення функціональності веб-платформи. Це, своєю чергою, формує надійну базу для реалізації інших підсистем, зокрема механізмів бронювання та обміну повідомленнями, коректна робота яких прямо залежить від коректного розмежування прав доступу.

## 2.3 Проєктування системи бронювання та управління часовими інтервалами

Система бронювання розглядається як центральний компонент веб-платформи, оскільки саме вона опосередковує базову взаємодію між користувачами та надавачами послуг. Належна реалізація механізму бронювання вимагає врахування часових обмежень і доступності ресурсів, а також мінімізації конфліктів, що можуть виникати за умов одночасного звернення кількох користувачів. Відповідно, проєктування цієї підсистеми доцільно спирати на

формалізовані підходи, зокрема на застосування моделі скінченного автомата (finite state machine, FSM) для опису життєвого циклу бронювання, а також на алгоритмічні процедури опрацювання часових інтервалів.

Концептуальну основу системи бронювання становить використання тайм-слотів (time slots), тобто визначених інтервалів часу, доступних для запису на конкретну послугу. Тайм-слоти формуються з урахуванням розкладу надавача послуг, тривалості вибраної послуги та поточної зайнятості. Така дискретизація часу подає його у вигляді відокремлених одиниць, що, з одного боку, полегшує вибір доступного часу користувачем, а з іншого – спрощує обробку запитів на рівні системи.

Формування тайм-слотів може бути організоване двома базовими способами – статичним або динамічним. За статичного підходу часові інтервали створюються наперед, наприклад із фіксованим кроком (30 чи 60 хвилин), після чого зберігаються у базі даних. Натомість динамічний підхід передбачає генерацію слотів під час звернення користувача, з урахуванням фактичного розкладу та вже наявних бронювань. Динамічна схема, як правило, забезпечує вищу гнучкість і дає змогу точніше використовувати доступний час, зокрема за умов змінної тривалості послуг.

Визначальним елементом функціонування системи є процес створення бронювання, який реалізується як послідовний алгоритм. Спочатку користувач обирає послугу, після чого система відображає доступні часові інтервали для обраної дати. Далі користувач фіксує конкретний слот, а система виконує перевірку його доступності. Така перевірка охоплює аналіз уже створених бронювань, відповідність розкладу надавача та відсутність часових накладань. У разі позитивного результату формується новий запис у базі даних, що містить відомості про користувача, послугу, часовий інтервал і статус бронювання. Схема такого алгоритму зображена на рис. 2.3.

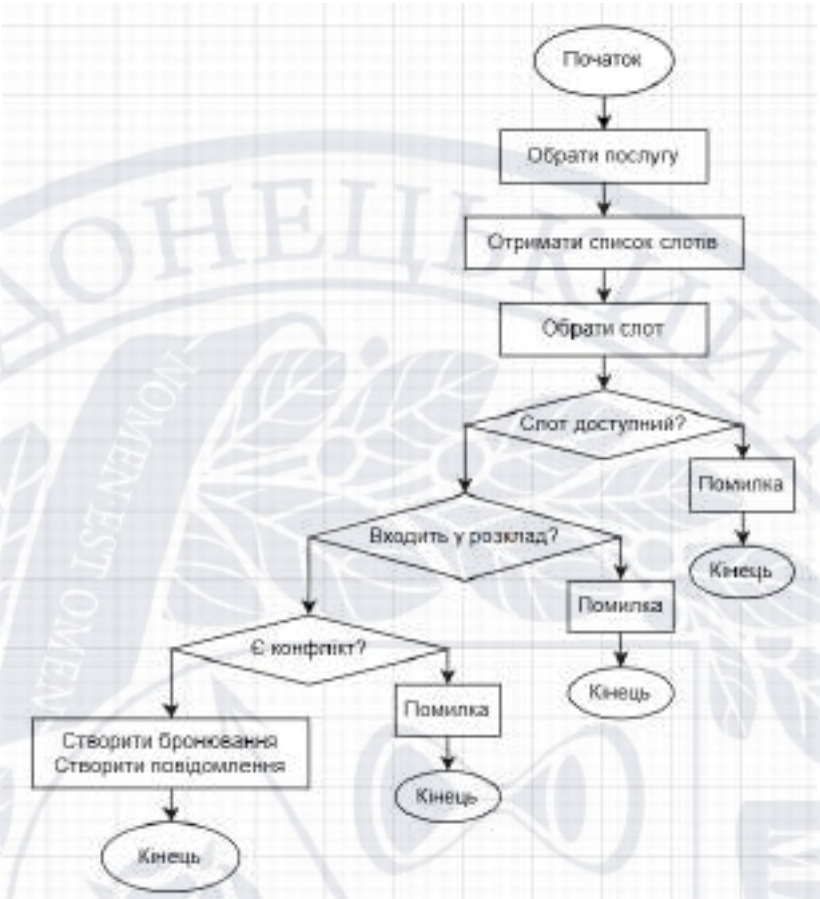


Рис. 2.3 – Алгоритм створення бронювання в системі онлайн-бронювання

Окремої уваги потребує перевірка конфліктів, оскільки система має унеможливлювати резервування одного й того самого інтервалу кількома користувачами одночасно. Для цього застосовується перевірка перетину часових інтервалів. Два інтервали трактуються як такі, що конфліктують, за виконання умов:

- початок нового інтервалу є ранішим за кінець наявного;
- кінець нового інтервалу є пізнішим за початок наявного.

Зазначене правило є усталеним у задачах планування та дає змогу ефективно виявляти накладання інтервалів [11]. Додатково на рівні бази даних можуть використовуватися спеціалізовані механізми, зокрема обмеження із застосуванням індексів типу GiST, які зменшують імовірність появи конфліктних записів шляхом автоматизованого контролю умов несумісності [14, 37].

Для формалізації логіки обробки бронювань у системі застосовується модель скінченного автомата (FSM), що дає змогу визначити множину можливих станів бронювання та допустимі переходи між ними. Використання такого опису забезпечує однозначне трактування життєвого циклу об’єкта бронювання і спрощує реалізацію бізнес-логіки на рівні застосунку [38].

Основні стани бронювання включають:

- pending – бронювання створене та очікує підтвердження постачальником послуг;
- confirmed – бронювання підтверджене та зарезервоване;
- cancelled – бронювання скасоване користувачем або постачальником;
- completed – послуга надана, бронювання завершене.

Переходи між станами зумовлюються подіями, що відбуваються в системі. Зокрема, після створення бронювання воно набуває стану pending; після підтвердження – confirmed; у разі скасування – cancelled; після фактичного надання послуги – completed. Така модель знижує ризик некоректних конфігурацій станів та підтримує контрольованість виконання бізнес-процесів.

Схематично життєвий цикл бронювання можна представити схемою зображеною на рис. 2.4.

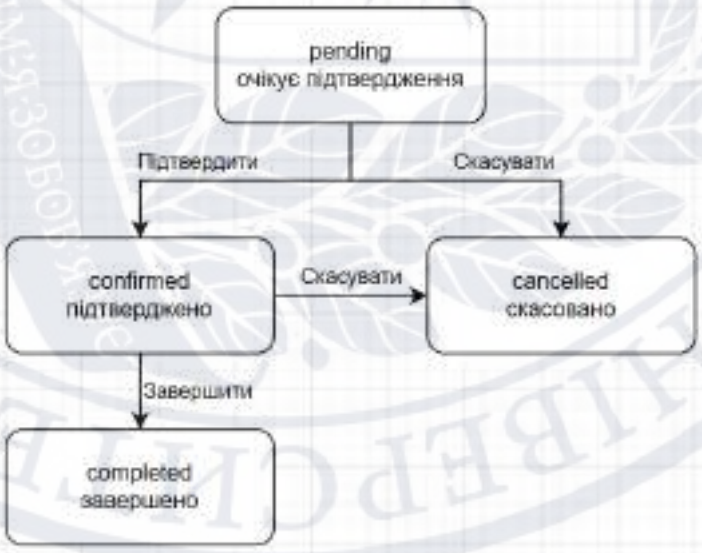


Рис. 2.4 – FSM-модель станів бронювання

Окремо слід розглянути механізм скасування бронювання як функціонально значущий елемент системи. Скасування може ініціювати як користувач, так і надавач послуг; після цього відповідний часовий інтервал повертається до переліку доступних для інших користувачів. У прикладних реалізаціях нерідко вводять додаткові обмеження, зокрема заборону скасування менш ніж за визначений час до початку послуги, що зазвичай сприяє раціональнішому використанню ресурсів.

З технічної перспективи реалізація системи бронювання передбачає використання серверної логіки для обробки запитів на створення, зміну та скасування бронювань. У середовищі розробки на базі Node.js із використанням фреймворку Express відповідна функціональність реалізується через REST API ендпоінти, які забезпечують взаємодію клієнтської частини із системою [4, 5, 32]. Водночас особливе значення має забезпечення атомарності операцій, оскільки це зменшує ризик ситуацій, коли паралельні запити можуть призводити до появи взаємовиключних бронювань.

Отже, система бронювання в межах запропонованої вебплатформи грунтується на поєднанні концепції тайм-слотів, процедур виявлення конфліктів і моделі скінченного автомата для керування станами бронювання. Така комбінація підходів підтримує узгодженість даних, підвищує надійність роботи та сприяє зручності використання, водночас формуючи підгрунтя для інтеграції з іншими підсистемами, зокрема з модулем повідомлень у режимі реального часу.

## 2.4 Проєктування системи повідомлень

Проєктування підсистеми повідомлень розглядається як один із визначальних компонентів веб-платформи онлайн-бронювання, оскільки саме вона забезпечує оперативне інформування користувачів про події в системі та підтримує комунікацію між клієнтами й надавачами послуг. На відміну від класичних веб-застосунків, у яких обмін даними переважно відбувається

синхронно через HTTP-запити, сучасні рішення потребують асинхронної взаємодії в режимі близькому до реального часу. Це дає змогу підвищити ефективність обробки подій і зробити взаємодію користувача із сервісом більш узгодженою. Відповідно, у межах даної роботи система повідомлень трактується не як периферійний модуль, а як інтегрована частина загальної архітектури платформи, що взаємодіє з підсистемою бронювання та реалізує event-driven модель обробки подій.

Функціональне призначення системи повідомлень полягає у передаванні інформації між компонентами платформи та кінцевими користувачами у відповідь на події, зокрема створення бронювання, його підтвердження, скасування або зміну статусу. При цьому істотною вимогою є зменшення затримок під час доставки даних, адже за умов конкуренції за обмежені ресурси (наприклад, часові слоти) навіть незначні затримки здатні спричиняти конфлікти або некоректне відображення актуального стану. Модель періодичного опитування сервера (polling) у такому випадку не забезпечує достатньої оперативності та створює додаткове навантаження на серверну інфраструктуру, що підтримує доцільність застосування технологій реального часу [1, 3].

У межах даної роботи реалізовано підсистему внутрішніх повідомлень (In-App Notifications), яка забезпечує інформування користувачів про події, що відбуваються в системі бронювання. На відміну від електронної пошти або зовнішніх push-сповіщень, повідомлення відображаються безпосередньо в інтерфейсі веб-платформи та є частиною її функціональності. Сповіщення формуються автоматично як реакція на ключові події бізнес-логіки, зокрема створення бронювання, його підтвердження, скасування або завершення. Такий підхід дозволяє підтримувати актуальність інформації для всіх учасників процесу бронювання та забезпечує оперативний зворотний зв'язок між системою і користувачем. Архітектурна реалізація підсистеми повідомлень грунтується на поєднанні REST API та технології WebSocket. REST API використовується для типових операцій, зокрема отримання історії повідомлень або збереження нових записів у базі даних, тоді як WebSocket підтримує двосторонній обмін даними у

режимі реального часу. На відміну від HTTP, де кожну взаємодію ініціює клієнт і вона завершується після надходження відповіді, WebSocket передбачає встановлення сталого з'єднання між клієнтом і сервером, що дозволяє передавати дані за потреби без повторного встановлення каналу зв'язку та супутніх накладних витрат [1, 4]. Такий підхід є придатним для чатів, систем сповіщень та інших інтерактивних механізмів, які потребують мінімальної затримки реакції [3].

З технічного погляду обмін повідомленнями доцільно описувати як послідовність подій. Наприклад, під час створення бронювання клієнт ініціює HTTP-запит через REST API; сервер зберігає дані в базі та формує відповідну подію. Далі ця подія передається через WebSocket на сторону провайдера, який без істотної затримки отримує повідомлення про нове бронювання [23]. Подібним чином, у разі підтвердження або скасування бронювання провайдером система створює подію, що доставляється клієнтові. У результаті формується асинхронна модель взаємодії, у межах якої зацікавлені сторони отримують актуальну інформацію без необхідності ручного оновлення сторінки. Ця послідовність у вигляді схеми зображена на рисунку 2.5.

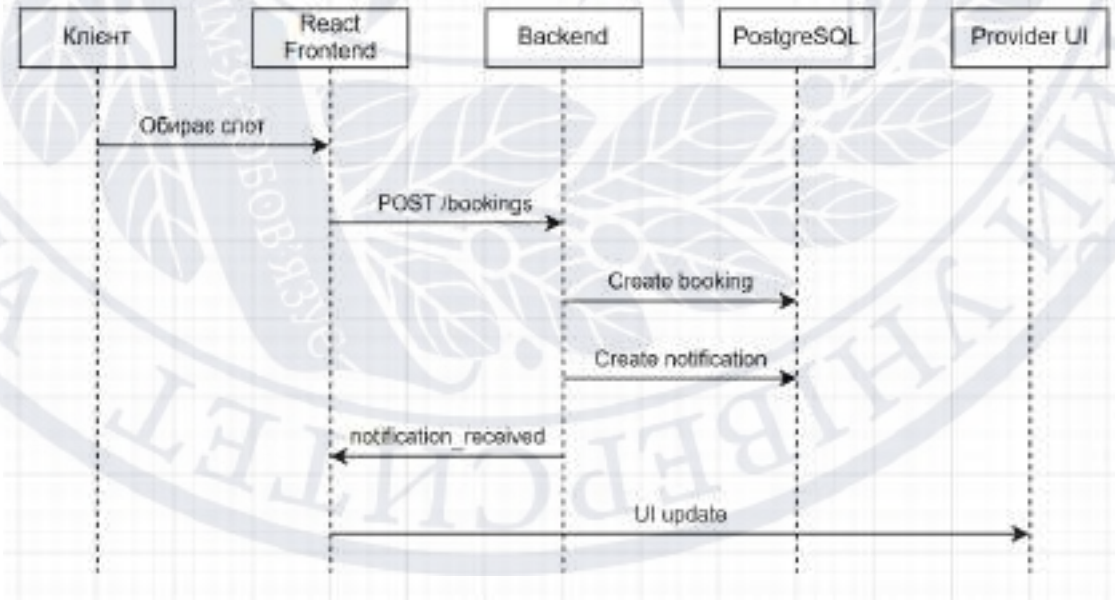


Рисунок 2.5 – Послідовність real-time обміну повідомленнями в системі бронювання

Ключовою характеристикою спроєктованого рішення є застосування event-driven підходу, за якого основні дії інтерпретуються як події, що запускають визначені реакції. Це сприяє декомпозиції системи на відносно автономні модулі та зменшує ступінь їх зв’язування, що зазвичай позитивно позначається на масштабованості й супроводжуваності програмного забезпечення. Так, подія "booking_created" може ініціювати паралельно кілька дій: створення запису в базі даних, формування системного повідомлення та надсилання push-сповіщення користувачеві. Така організація полегшує розширення функціональності без модифікації вже наявних компонентів.

Для забезпечення роботи підсистеми внутрішніх повідомлень у базі даних передбачено окрему сутність Notification. Кожне сповіщення містить інформацію про користувача-одержувача, тип події, текст повідомлення, дату створення та статус прочитання. Додатково реалізовано механізм відстеження непрочитаних повідомлень, що дозволяє відображати їх кількість у користувацькому інтерфейсі. Така структура забезпечує можливість зберігання історії сповіщень і підтримує реалізацію функцій перегляду та позначення повідомлень як прочитаних.

WebSocket-з’єднання може реалізовуватися із застосуванням спеціалізованих бібліотек, зокрема Socket.IO, яка за наявності підтримки використовує WebSocket, а за потреби переходить на резервні механізми на основі HTTP long polling [8]. Такий підхід підвищує надійність роботи в різних середовищах виконання. Додатково такі бібліотеки спрощують реалізацію механізмів адресної доставки повідомлень конкретним користувачам. Для цього можуть використовуватися логічні групи підключень (rooms), які дозволяють надсилати повідомлення лише тим користувачам, для яких вони призначені. Це особливо важливо для систем бронювання, де інформація про замовлення повинна бути доступною тільки відповідним учасникам процесу.

Інтеграція підсистеми внутрішніх повідомлень із підсистемою бронювання є одним із центральних елементів запропонованого проєкту. Усі основні події, пов'язані зі створенням і зміною стану бронювань, автоматично породжують

відповідні In-App сповіщення. Після створення бронювання система надсилає повідомлення провайдеру послуги, а після підтвердження або скасування бронювання формується повідомлення для клієнта. Завдяки використанню WebSocket такі сповіщення доставляються практично миттєво та відображаються в інтерфейсі без необхідності оновлення сторінки.

Отже, спроєктована система повідомлень виступає невід'ємним компонентом веб-платформи онлайн-бронювання та забезпечує взаємодію користувачів у режимі реального часу. Поєднання WebSocket із REST API формує гнучку й масштабовану архітектурну основу, узгоджену з типовими вимогами до сучасних веб-застосунків. Запропоноване рішення орієнтоване на зниження затримок передачі даних, підвищення якості взаємодії користувача із системою та створення передумов для подальшого функціонального розвитку платформи.

### 2.5 Проєктування ER-діаграми

Проєктування структури бази даних розглядається як один із визначальних етапів розроблення веб-платформи онлайн-бронювання, оскільки саме на цьому рівні фіксуються принципи зберігання даних, їх подальшої обробки та характер взаємозв'язків між ними. Для розв'язання завдань, сформульованих у межах цієї роботи, обрано реляційну модель даних, реалізація якої здійснюється із застосуванням СКБД PostgreSQL. Такий вибір зумовлено потребою підтримувати цілісність даних, коректно моделювати складні зв'язки між сутностями та задавати обмеження безпосередньо на рівні бази даних, зокрема з метою зниження ймовірності конфліктів бронювання [12, 14].

ER-діаграма (Entity-Relationship Diagram) використовується як формалізований опис структури бази даних, що забезпечує наочне відображення ключових сутностей предметної області, їх атрибутів і зв'язків. Для розроблюваної системи ER-діаграма фіксує логіку взаємодії користувачів, послуг, бронювань і повідомлень, завдяки чому формується узгоджене уявлення

про функціонування програмного комплексу [39]. ER-діаграму бази даних веб-платформи онлайн-бронювання наведено на рисунку 2.6.

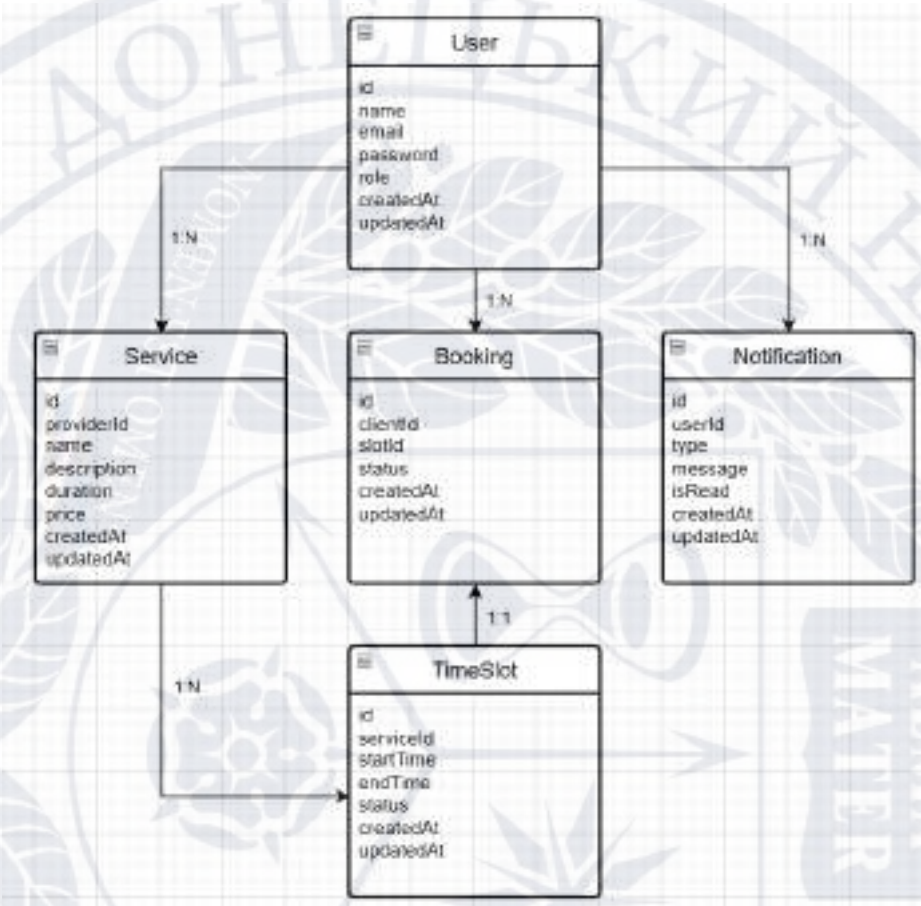


Рисунок 2.6 – ER-діаграма бази даних веб-платформи онлайн-бронювання

Базовою сутністю системи є користувач (User), під яким розуміється будь-який учасник платформи незалежно від призначеної йому ролі. У межах прийнятого підходу ролі (client, provider, admin) представлені в одній таблиці users, що відповідає ідеї нормалізації та спрощує адміністрування доступу. До атрибутів сутності належать унікальний ідентифікатор, персональні відомості, дані облікового запису для автентифікації, а також поле ролі, яке визначає права користувача відповідно до моделі RBAC [9]. Запропоноване рішення зменшує ризик дублювання даних і зберігає можливість подальшого розширення функціоналу.

Наступною сутністю виступає послуга (Service), яка описує конкретний сервіс, що надається провайдером. Кожна послуга пов’язується з користувачем-провайдером через зовнішній ключ; таким чином реалізується відношення типу «один до багатьох», коли один провайдер може пропонувати кілька послуг. Атрибутивний склад сутності охоплює назву, опис, тривалість і вартість, що забезпечує подання клієнтові релевантної інформації під час вибору.

Для організації механізму бронювання вводиться сутність розкладу (Schedule), яка задає базові параметри доступності провайдера. Вона містить відомості про робочі дні та часові інтервали, у межах яких можуть формуватися слоти, придатні для резервування. Спираючись на ці дані, система формує часові інтервали (time slots), які виконують роль проміжної логічної ланки між розкладом і бронюваннями. Хоча збереження тайм-слотів у базі даних не є обов’язковим, їх використання, як правило, спрощує пошук доступного часу та робить перевірку зайнятості ресурсу більш прозорою.

Центральною сутністю є бронювання (Booking), що відображає факт резервування послуги клієнтом. Вона одночасно пов’язана з кількома сутностями, зокрема з користувачем-клієнтом, провайдером і послугою. Зв’язування виконується через зовнішні ключі та дає змогу отримувати повний контекст для кожного запису бронювання. Набір атрибутів включає часові межі (start_time, end_time), статус і дату створення. Статус бронювання інтерпретується як реалізація кінцевого автомата станів (FSM), що дає змогу відстежувати життєвий цикл бронювання, зокрема стани «очікує підтвердження», «підтверджено», «скасовано» та «завершено».

Окремий акцент у проєктуванні зроблено на зниженні ризику конфліктів бронювання, які виникають у разі спроби резервувати один і той самий часовий інтервал кількома користувачами. Для цього застосовуються перевірки на рівні прикладної логіки (зокрема аналіз перетину часових інтервалів), а також інструменти забезпечення обмежень на рівні бази даних. Зокрема, у PostgreSQL може бути залучено індекс типу GiST разом з обмеженням EXCLUDE, що автоматично забороняє накладання часових діапазонів для одного провайдера

[14, 40]. Зазначена комбінація підвищує стійкість системи до помилок і зменшує ймовірність неконсистентних станів.

Підсистема обміну повідомленнями описується сутністю Notification, яка використовується для фіксації системних повідомлень, що генеруються у відповідь на події в системі, наприклад створення бронювання або зміну його статусу.

З погляду зв’язків між сутностями в системі визначено кілька базових типів відношень. Зв’язок між користувачем і послугами відповідає типу «один до багатьох», оскільки один провайдер може надавати кілька послуг. Подібне відношення використовується між користувачем і бронюваннями, адже один користувач здатен створювати багато записів бронювання. Сутність бронювання також співвідноситься з послугою, що дозволяє однозначно встановити, який сервіс було замовлено. Для повідомлень передбачено два зв’язки з користувачем: окремо для відправника та для отримувача, що дає змогу коректно моделювати двосторонню комунікацію [39].

У підсумку, на основі ER-діаграми сформовано логічно узгоджену структуру бази даних, яка підтримує необхідні функціональні можливості системи онлайн-бронювання. Використання реляційної моделі, зовнішніх ключів і обмежень цілісності забезпечує коректність даних, тоді як залучення спеціалізованих механізмів, зокрема GiST-індексів, сприяє підвищенню надійності та продуктивності системи. Запропонована структура зберігає потенціал масштабування та може бути розширена надалі без потреби радикальної зміни базової архітектури.

# ВИСНОВКИ ДО РОЗДІЛУ 2

Проєктування веб-платформи онлайн-бронювання послуг, здійснене у другому розділі, грунтувалося на вимогах, сформованих аналізом предметної області. На цьому етапі було визначено архітектуру системи, структуру взаємодії її компонентів та принципи реалізації ключових бізнес-процесів.

Розроблена клієнт-серверна архітектура платформи передбачає інтеграцію REST API для основних операцій та WebSocket-з'єднань для обміну даними у реальному часі. Цей підхід забезпечує поєднання надійності традиційних механізмів взаємодії з можливостями оперативної синхронізації інформації між користувачами.

У межах проєктування функціональної складової було сформовано алгоритм створення бронювання, що включає вибір послуги, перевірку доступності часового слоту, створення запису та обробку його станів. Для управління життєвим циклом бронювання запропоновано модель скінченного автомата станів, яка забезпечує коректність переходів та підвищує надійність бізнес-логіки системи.

Додатково було спроєктовано систему внутрішніх повідомлень для автоматичного інформування користувачів про події, зокрема створення, підтвердження або скасування бронювань. Особливу увагу приділено механізму передачі повідомлень у реальному часі за допомогою WebSocket, що підтримує актуальність даних без необхідності оновлення сторінки.

На основі визначених функціональних вимог було побудовано ER-діаграму бази даних та сформовано структуру ключових сутностей системи, які включають користувачів, послуги, часові слоти, бронювання та повідомлення. Розроблена модель даних забезпечує цілісність інформації, підтримує рольове розмежування доступу та слугує основою для реалізації механізмів бронювання і повідомлень.

Таким чином, у другому розділі було сформовано повний проєкт веб-платформи онлайн-бронювання послуг, який визначає архітектурні, функціональні та інформаційні основи для майбутньої програмної реалізації, що буде розглянута у наступному розділі.

# РОЗДІЛ 3. РЕАЛІЗАЦІЯ ВЕБ-ПЛАТФОРМИ ДЛЯ ОНЛАЙН-БРОНЮВАННЯ

## 3.1 Вибір технологій та середовища розробки

Для розроблення веб-платформи онлайн-бронювання застосовано стек, орієнтований на продуктивність, масштабованість і взаємодію в реальному часі. Систему спроєктовано за клієнт–серверною моделлю з виокремленням клієнта, серверного застосунку, реляційної БД та каналу миттєвого обміну повідомленнями. Серверну частину реалізовано на Node.js, який підтримує подієво-орієнтоване виконання з неблокуючим введенням-виведенням і придатний для високої кількості паралельних мережевих запитів, зокрема для WebSocket-з’єднань. Серверний застосунок побудовано з використанням Express.js як мінімалістичного фреймворку для маршрутизації, обробки HTTP-запитів і middleware, а також інтеграції бібліотек; у системі він слугує основою REST API для взаємодії клієнта і сервера.

Клієнтську частину реалізовано на React для побудови SPA з компонентною організацією інтерфейсу, що зменшує кількість повних перезавантажень сторінок і полегшує повторне використання коду. Обрана технологія узгоджується з REST API та WebSocket-підходом, необхідними для бронювань і повідомлень.

У клієнті передбачено сторінки авторизації, перегляду послуг, керування бронюваннями, обміну повідомленнями та адміністративних функцій; навігацію забезпечує React Router, а HTTP-взаємодію з сервером – Axios.

Для зберігання даних використано PostgreSQL як надійну об’єктно-реляційну СКБД із підтримкою транзакцій, обмежень цілісності та складних зв’язків між сутностями. У БД зберігаються користувачі, послуги, часові слоти, бронювання й повідомлення; застосовано зовнішні ключі, унікальність та індекси, а також транзакційні перевірки конфліктів інтервалів для зменшення ризику дублювань бронювань. Для доступу до БД використано Prisma ORM, яка

описує схему декларативно та генерує типізований клієнт, зменшуючи частку ручного SQL і підвищуючи коректність взаємодії з даними за рахунок перевірок типів.

Взаємодію в реальному часі реалізовано через Socket.IO поверх WebSocket, що забезпечує двосторонній обмін даними без періодичного опитування сервера. На відміну від HTTP-моделі запит–відповідь, постійне з'єднання дає змогу оперативно передавати події та зміни стану. У системі Socket.IO застосовано для оновлення статусів бронювань, повідомлень про створення або скасування записів і внутрішніх сповіщень, завдяки чому користувач отримує актуальні дані без ручного оновлення сторінки.

Автентифікацію та авторизацію реалізовано на основі JSON Web Token як стандарту передавання підписаних JSON-даних; після входу токен використовується для підтвердження ідентичності в наступних запитах, підтримуючи безстанний підхід, сумісний із REST API. Контроль доступу організовано за RBAC; перевірки виконуються на сервері через middleware та на клієнті шляхом обмеження доступу до сторінок і функцій.

Для викликів REST API на клієнті використано Axios із централізованим додаванням JWT через interceptors і уніфікованою обробкою відповідей, що забезпечує узгоджену взаємодію модулів із сервером. Отже, поєднання Node.js, Express.js, React, PostgreSQL, Prisma, Socket.IO, JWT та Axios забезпечує реалізацію бронювання, рольового доступу, контролю цілісності даних і обміну в реальному часі відповідно до визначених вимог.

### 3.2 Реалізація серверної частини системи

Серверну частину веб-платформи реалізовано на Node.js із використанням Express.js. Вона обробляє клієнтські запити, виконує бізнес-логіку, взаємодіє з PostgreSQL через Prisma ORM, забезпечує автентифікацію й авторизацію, а також підтримує бронювання та повідомлення. Архітектуру організовано за багаторівневим підходом із розмежуванням маршрутизації, контролерів,

сервісів, middleware і WebSocket-компонента. Структуру застосунку сформовано за модульним принципом: окремі директорії для routes, controllers, services, middleware, конфігурацій і WebSocket-модуля. На рисунку 3.1 подано структуру серверної частини після завершення проєктування та реалізації.

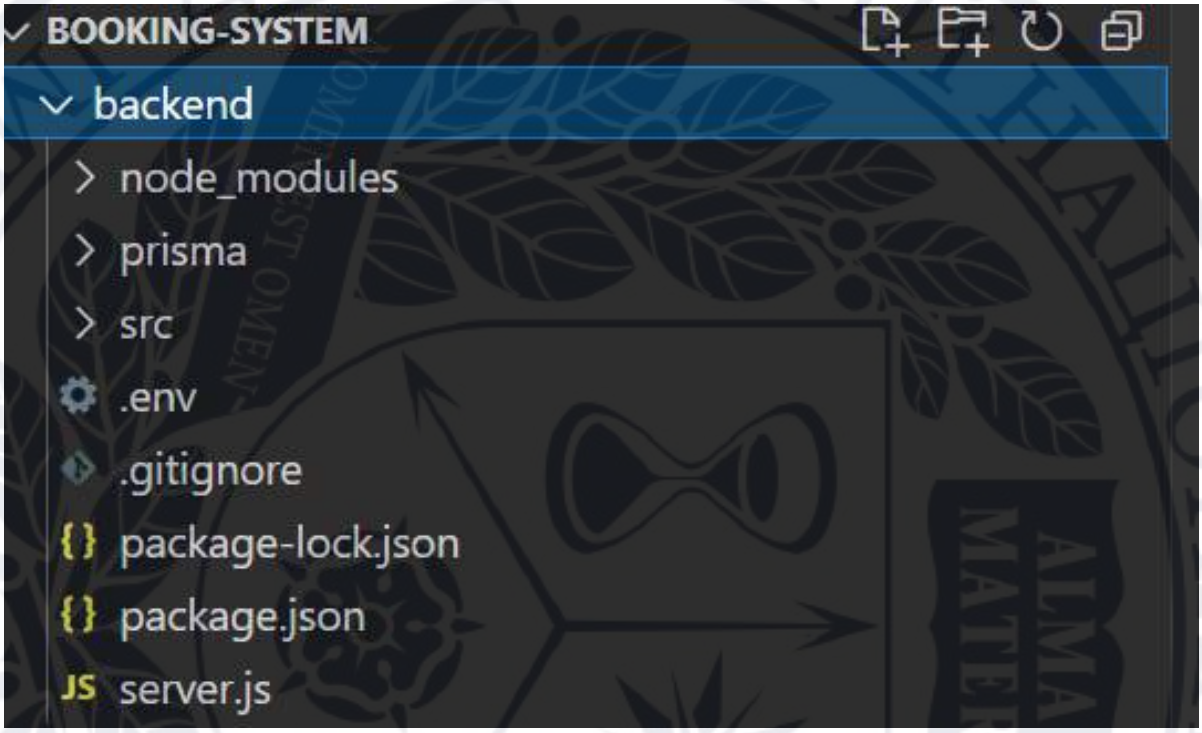


Рисунок 3.1 – Структура серверної частини веб-платформи

### 3.2.1 Реалізація REST API

Для взаємодії клієнта із сервером реалізовано REST API, що надає доступ до функцій системи через HTTP та відповідає стилю REST, зберігаючи розділення клієнтської і серверної частин та підтримуючи масштабування. API організовано у групи маршрутів за функціональними модулями. Маршрути автентифікації охоплюють реєстрацію, вхід і отримання даних поточного користувача: POST /auth/register, POST /auth/login, GET /auth/me.

Під час реєстрації перевіряється коректність даних і унікальність email; пароль хешується bcrypt. Після входу видається JWT для доступу до захищених ресурсів. На рисунку 3.2 видно приклад тестування реєстрації в Postman.


Рисунок 3.2 – Тестування REST API реєстрації користувача

Маршрути роботи з послугами провайдерів підтримують створення, читання, оновлення та видалення: GET /services, POST /services, PATCH /services/:id, DELETE /services/:id.

Створення послуги дозволене лише ролі провайдера; дані зберігаються в таблиці services та прив’язуються до виконавця. Окремо реалізовано маршрути для часових слотів: створення доступних інтервалів, перегляд власних слотів і їх блокування. Підсистема бронювання представлена такими маршрутами: POST /bookings, GET /bookings/my, GET /bookings/:id, PATCH /bookings/:id/confirm, PATCH /bookings/:id/cancel, PATCH /bookings/:id/complete.

Клієнт створює бронювання, провайдер змінює його стан через підтвердження, скасування або завершення. Маршрути системи повідомлень мають вигляд: GET /notifications, PATCH /notifications/:id/read, PATCH

/notifications/read-all, GET /notifications/unread-count. Вони забезпечують отримання повідомлень, позначення прочитання та підрахунок непрочитаних.

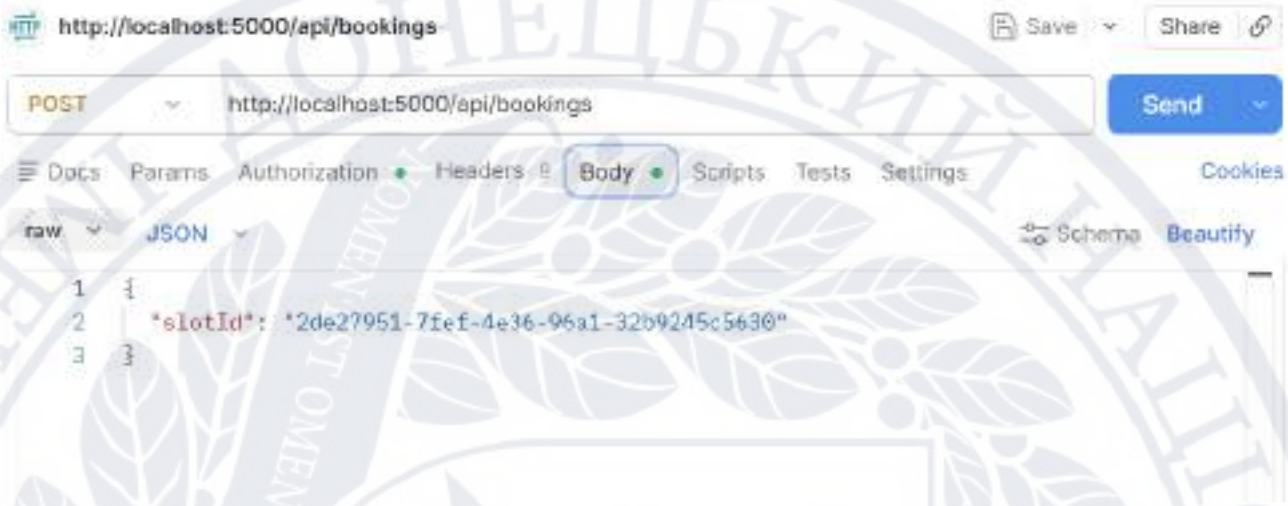


Рисунок 3.3 – Тестування REST API системи бронювання

На рисунку 3.3 наведено приклад тестування маршруту бронювання. Якщо роль відповідна і слот вільний то він буде заброньований.

### 3.2.2 Реалізація RBAC-моделі

Вимогою системи є рольове керування доступом (RBAC), що обмежує функції відповідно до ролі користувача.

Передбачено три ролі:

- CLIENT;
- PROVIDER;
- ADMIN.

Роль зберігається в таблиці user (поле role), включається до JWT після авторизації та використовується під час перевірок доступу (рис. 3.4). Автентифікацію реалізовано на основі JWT: після входу сервер формує токен із ідентифікатором користувача та роллю, який передається в заголовку.

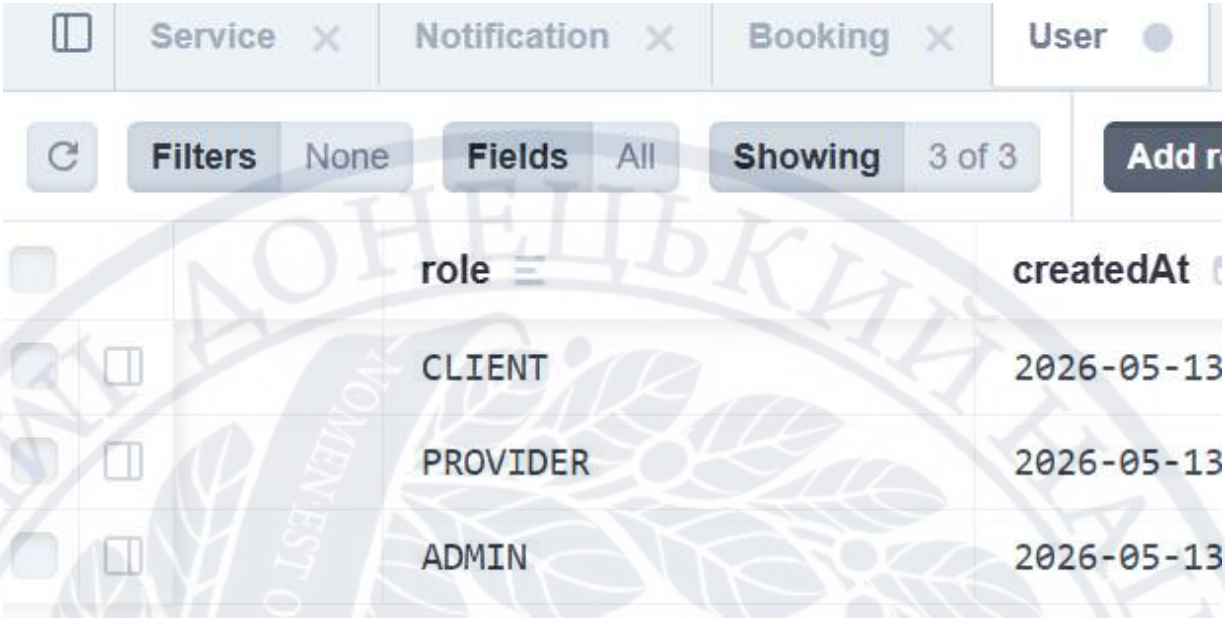


Рисунок 3.4 – Роль користувача в таблиці user

Middleware verifyToken перевіряє підпис токена, отримує дані користувача та додає їх до запиту. Middleware checkRole зіставляє роль користувача з переліком дозволених для маршруту, зокрема створення послуг і слотів обмежено роллю PROVIDER, а створення бронювання – роллю CLIENT. У підсумку RBAC забезпечує централізоване розмежування прав доступу.

### 3.2.3 Реалізація бізнес-логіки бронювання

Підсистема бронювання включає керування станами, перевірку конфліктів часових інтервалів та забезпечення цілісності даних за одночасного доступу. Життєвий цикл бронювання формалізовано як FSM із такими станами (рис. 3.5):
PENDING;
CONFIRMED;
CANCELLED;
COMPLETED.

Нове бронювання отримує PENDING; провайдер переводить його в CONFIRMED або CANCELLED; після надання послуги встановлюється COMPLETED. Дозволені переходи:

PENDING – CONFIRMED;

PENDING – CANCELLED;

CONFIRMED – COMPLETED;

CONFIRMED – CANCELLED.

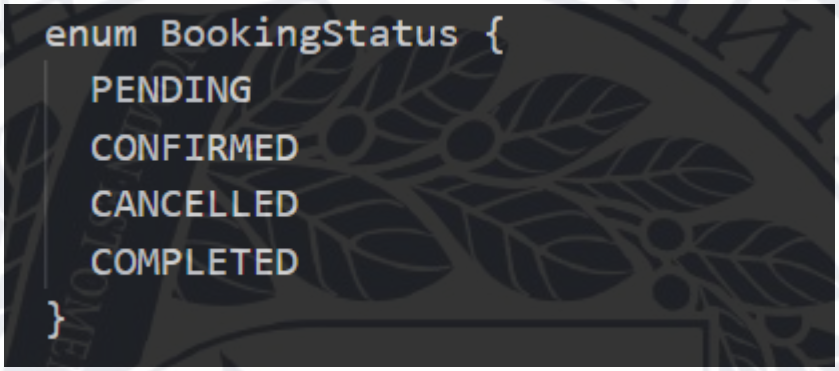


Рисунок 3.5 – FSM-модель станів бронювання

Ключовою є перевірка конфліктів: під час створення бронювання система контролює статус вибраного слота та його доступність, з огляду на можливі одночасні запити. Після створення бронювання слот переводиться в BOOKED, що блокує повторне використання. Для зменшення ризику race condition застосовано транзакції Prisma: створення запису в bookings і оновлення статусу слота виконуються атомарно, а в разі помилки зміни скасовуються. Такий підхід підтримує цілісність даних за високої конкуренції запитів. На рисунку 3.6 наведено фрагмент реалізації створення бронювання та перевірки слота.

```
const createBooking = async (clientId, data) => {
  const { slotId } = data;

  if (!slotId) {
    throw new Error("slotId is required");
  }

  const booking = await prisma.$transaction(async (tx) => {
    const slot = await tx.timeSlot.findUnique({
      where: {
        id: slotId,
      },
      include: {
        service: true,
      },
```

Рисунок 3.6 – Реалізація бізнес-логіки створення бронювання

У результаті створено серверний модуль, що реалізує REST API, механізми автентифікації та авторизації, RBAC і бізнес-логіку бронювання, забезпечуючи керованість і можливість подальшого розширення, зокрема для сценаріїв взаємодії в реальному часі, розглянутих у наступних підрозділах.

### 3.3 Реалізація бази даних

Одним із визначальних етапів розробки веб-платформи онлайн-бронювання стала реалізація бази даних для зберігання відомостей про користувачів, послуги, часові слоти, бронювання та повідомлення. Як СКБД обрано PostgreSQL з огляду на підтримку транзакцій, засоби забезпечення цілісності та роботу зі зв’язками між сутностями і налаштовано підключення серверної частини через Prisma ORM із параметрами в .env.

#### 3.3.1 Реалізація структури БД у PostgreSQL

Реалізацію виконано відповідно до попередньо сформованої ER-моделі. Структуру даних нормалізовано; використано зовнішні ключі, індекси та

обмеження цілісності для підвищення стабільності й продуктивності. Сформовано п’ять таблиць: user, services, time_slots, bookings, notifications. Таблиця user зберігає ім’я, email, пароль, роль і дату створення; поле role застосовано для RBAC. На рисунку 3.7 наведено структуру таблиці user у PostgreSQL.

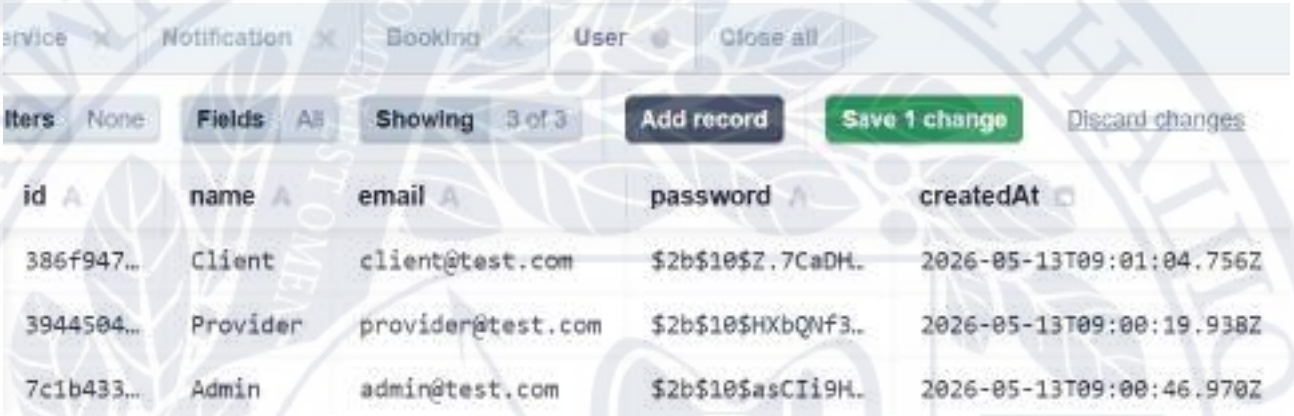


Рисунок 3.7 – Структура таблиці user у PostgreSQL

Таблиця services містить назву, опис, тривалість, вартість і посилання на провайдера; кожна послуга пов’язана з користувачем ролі PROVIDER. Таблиця time_slots зберігає початок/завершення, статус та посилання на послугу й провайдера. Передбачено стани: AVAILABLE, BOOKED, BLOCKED. Таблиця bookings фіксує посилання на клієнта, провайдера, послугу та слот; поле status відображає життєвий цикл бронювання (PENDING, CONFIRMED, CANCELLED, COMPLETED). Таблиця notifications реалізує внутрішні повідомлення (текст, тип, статус прочитання, дата створення); повідомлення генеруються під час зміни станів бронювання для real-time сповіщень. На рисунку 3.8 наведено основні таблиці бази даних у Prisma Studio.

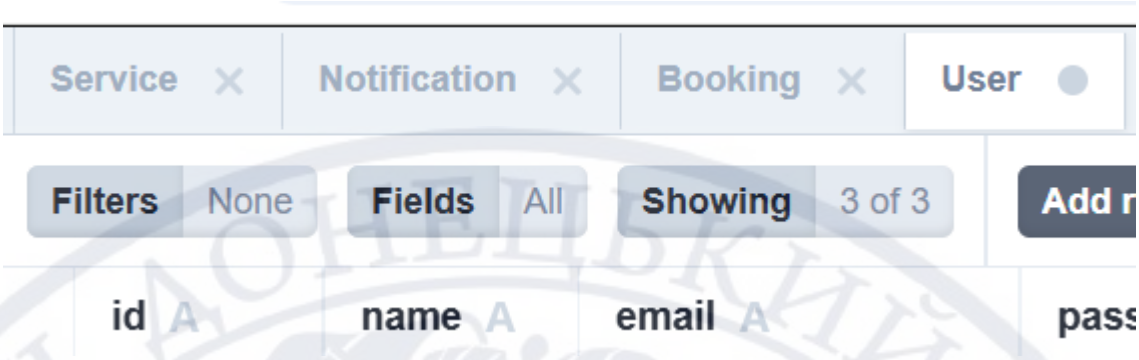


Рисунок 3.8 – Основні таблиці бази даних у Prisma Studio

### 3.3.2 Реалізація зв’язків та обмежень

Зв’язки між сутностями реалізовано через foreign keys для підтримання цілісності даних. Визначено такі відношення: користувач-послуги (1:N), провайдер-слоти (1:N), послуга-слоти (1:N), користувач-бронювання (1:N), слот-бронювання (1:1), користувач-повідомлення (1:N). Застосовано @relation для генерації зовнішніх ключів. Для email встановлено UNIQUE. Для критичних полів використано NOT NULL: email, password, role, startTime, endTime, status. Створено індекси для providerId, serviceId, status, startTime, endTime, що прискорює запити та пошук доступних слотів.

Стани слотів, бронювань і ролі користувачів задано enum-типами PostgreSQL для обмеження допустимих значень. Для запобігання подвійного бронювання застосовано перевірки доступності слоту та транзакції; також розглянуто exclusion constraints і GiST-індекси для уникнення перетинів часових інтервалів.

Для взаємодії з PostgreSQL і керування схемою використано Prisma ORM. Створено schema.prisma як центральний опис джерела даних, генератора Prisma Client, моделей, зв’язків та enum. Наведений рисунок 3.9 показує фрагмент schema.prisma з описом ключових сутностей.

```
model User {
  id        String   @id @default(uuid())
  name      String
  email     String   @unique
  password  String
  avatar    String?
  role      Role     @default(CLIENT)
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt

  // як client
  bookings      Booking[]      @relation("ClientBookings")
```

Рисунок 3.9 – Опис моделей бази даних у файлі schema.prisma

Окремі ENUM-типи охоплюють ролі, статуси бронювань і слотів, а також типи повідомлень, забезпечуючи контроль значень на рівні схеми. Міграції створено через Prisma Migrate, що автоматизує формування SQL для таблиць, індексів і зв’язків у PostgreSQL. Для застосування міграцій використано команду npx prisma migrate dev. Після виконання міграцій схема синхронізується зі структурою БД. На рисунку 3.10 представлено створення та застосування міграцій у Prisma.

```
PS C:\Users\user\Desktop\booking-system\backend> npx prisma migrate dev --name update_notification_model
Environment variables loaded from .env
Prisma schema loaded from prisma\schema.prisma
Datasource "db": PostgreSQL database "booking_system", schema "public" at "localhost:5432"
```

Рисунок 3.10 – Виконання міграцій Prisma

Після цього згенеровано Prisma Client для типізованого доступу до PostgreSQL командою npx prisma generate. Prisma Client використано в сервісах серверної частини для CRUD-операцій, роботи зі слотами, бронюваннями та повідомленнями, що зменшує обсяг SQL і підсилює коректність завдяки

перевірці типів. У підсумку, поєднання PostgreSQL і Prisma ORM забезпечило узгоджену структуру зберігання даних із підтримкою зв’язків, контролем цілісності та ефективною взаємодією серверної частини з базою даних.

## 3.4 Реалізація клієнтської частини системи

Клієнтську частину веб-платформи реалізовано на основі React, що дало змогу побудувати SPA з динамічним інтерфейсом, оновленням без перезавантаження та компонентною декомпозицією. Вона відповідає за візуалізацію даних, взаємодію користувача, виконання HTTP-запитів до сервера та обробку подій у реальному часі через WebSocket-з’єднання.

Під час розробки пріоритетом була масштабованість, щоб спростити розширення функцій і подальший супровід.

### 3.4.1 Реалізація React-архітектури

Після ініціалізації застосунку структуру проєкту сформовано відповідно до модульних принципів, розподіливши компоненти за директоріями за їхнім призначенням. Клієнтська частина організована через такі елементи:

- pages;
- components;
- services;
- contexts.

Директорія pages містить сторінки, що відповідають маршрутам застосунку; навігацію між ними забезпечує React Router. У проєкті визначено сторінки: LoginPage, RegisterPage, DashboardPage, ProvidersPage, BookingPage, NotificationsPage, HomePage, MyBookingsPage, MyServicesPage, ServiceDetailsPage. На рисунку 3.11 наведено структуру директорії pages.

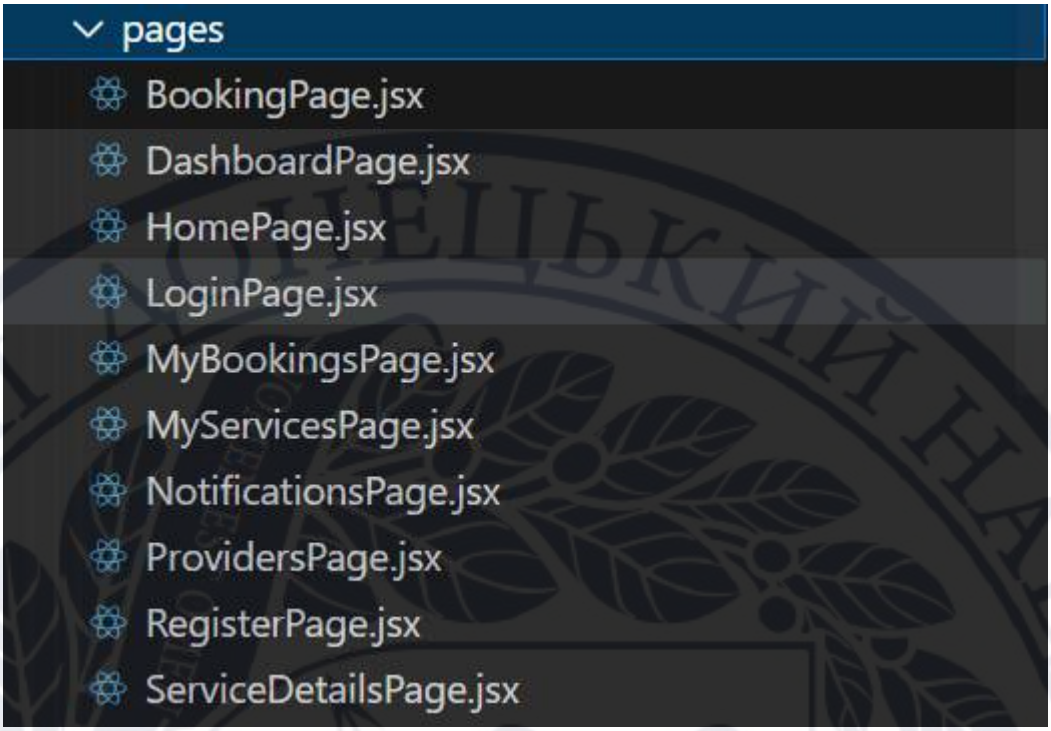


Рисунок 3.11 – Структура директорії pages

Для повторного використання елементів інтерфейсу створено директорію components з автономними UI-компонентами, придатними для застосування на різних сторінках. До них належать Navbar та NotificationBell.

Такий підхід зменшує дублювання коду й полегшує супровід. Взаємодію із сервером організовано через директорію services, де зосереджено модулі роботи з REST API. Ключові сервіси:

- authService;
- bookingService;
- serviceService;
- socketService;
- timeSlotService;
- notificationService.

Сервіси інкапсулюють логіку запитів та використовують Axios для обміну з сервером. Глобальний стан керується через Context API. У директорії contexts реалізовано контексти, що надають спільні дані компонентам. Зокрема створено

AuthContext, який зберігає відомості про користувача та роль. На рисунку 3.12 наведено загальну структуру клієнтської частини веб-платформи.

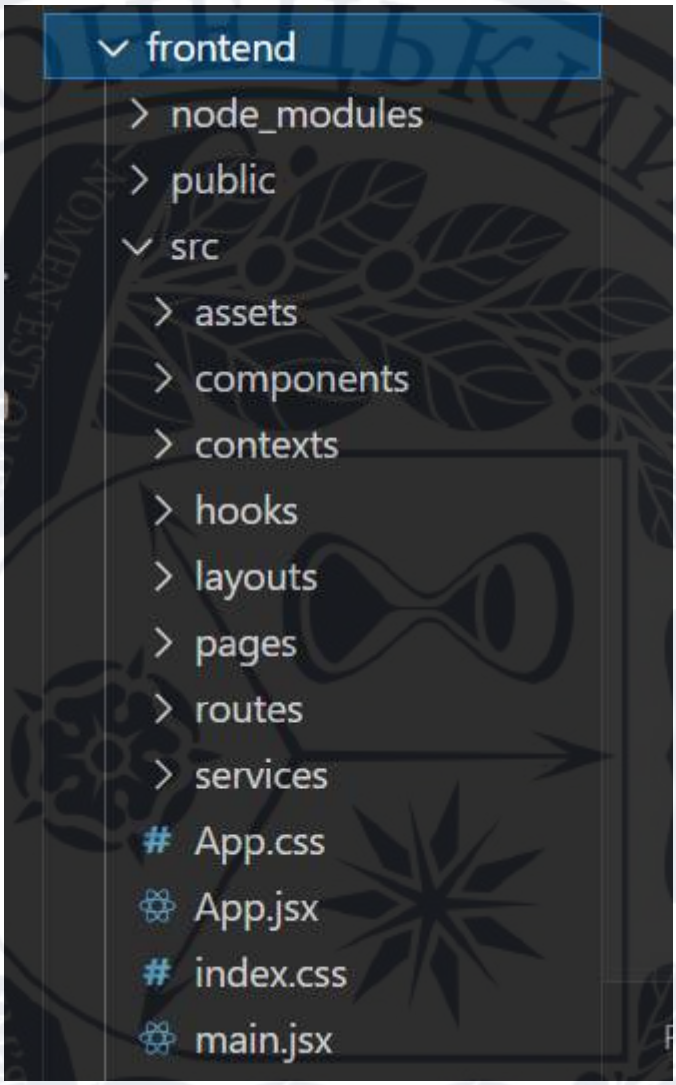


Рисунок 3.12 – Архітектура React-застосунку

### 3.4.2 Реалізація інтерфейсу користувача

Після формування архітектури реалізовано інтерфейс, орієнтований на зрозумілу роботу користувачів із різними ролями. Було створено простий інтерфейс сторінок за допомогою CSS-фреймворку Tailwind CSS. Інтерфейс виконано у мінімалістичному SaaS-стилі зі світлою колірною схемою та єдиним акцентним кольором. Всі сторінки побудовано на основі єдиної системи компонентів: картки, форми, кнопки та навігаційна панель витримані в

узгодженому стилі. Насамперед створено сторінки автентифікації (вхід і реєстрація) з формами введення електронної пошти, пароля та інших даних (див. рис. 3.13). Після надсилання форми через REST API у разі успіху JWT-токен зберігається в браузері для подальших звернень до системи.

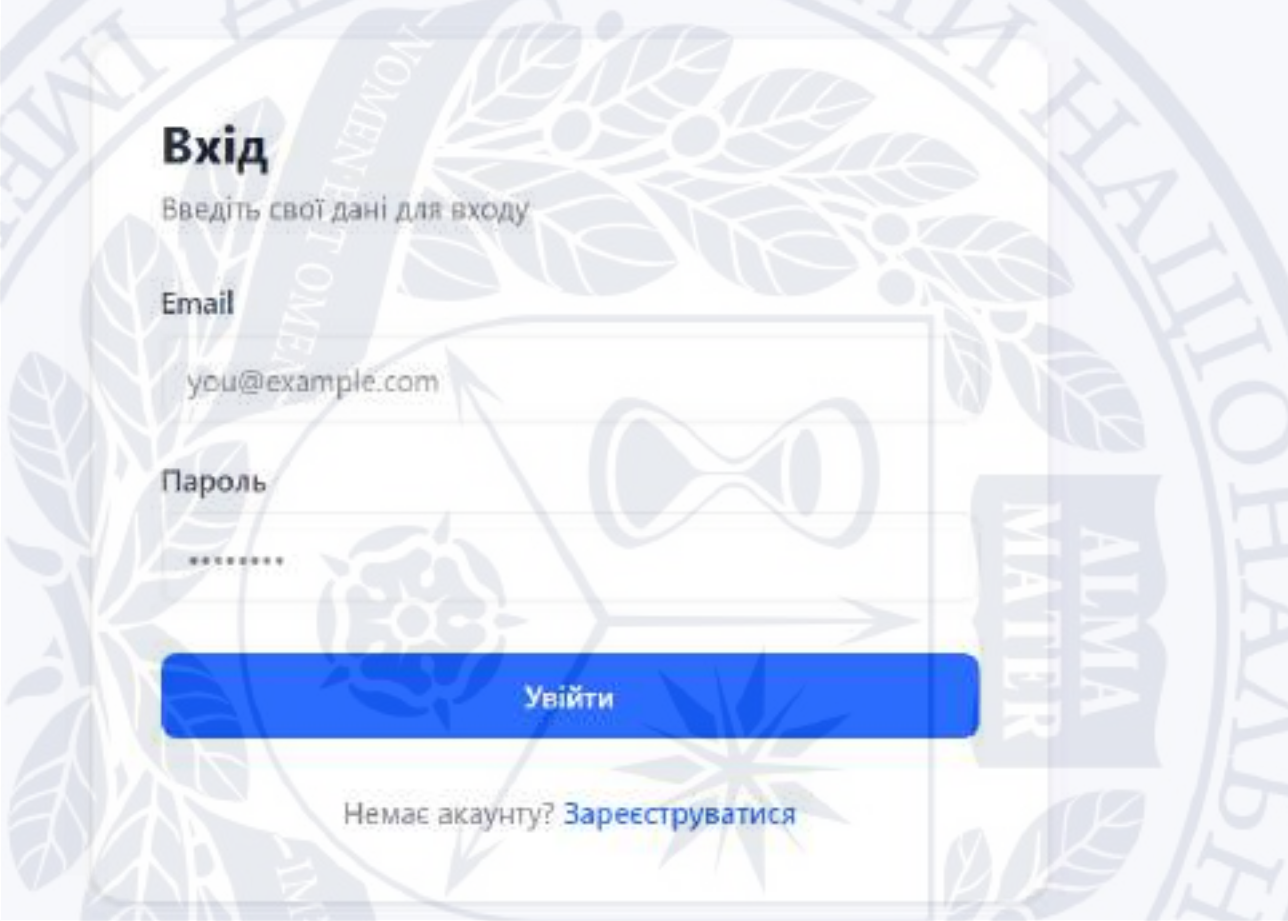


Рисунок 3.13 – Сторінка авторизації користувача

Ключовим модулем для провайдера є Provider Dashboard, що забезпечує керування послугами, слотами та бронюваннями, включно з переглядом і зміною записів (див. додаток А). На рисунку 3.14 наведено реалізацію інтерфейсу кабінету провайдера.

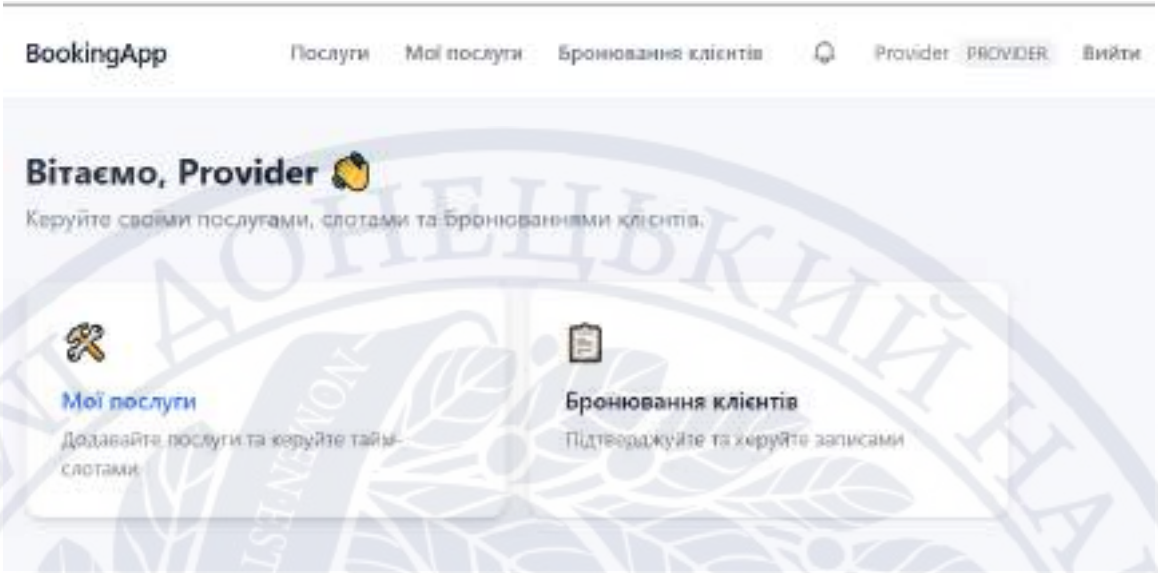


Рисунок 3.14 – Кабінет провайдера послуг

Для клієнтів реалізовано сторінку бронювання з послідовністю дій: вибір провайдера, послуги, слоту та підтвердження (див. додаток Б). Після створення бронювання дані відображаються в кабінеті користувача. На рисунку 3.15 наведено сторінку бронювання послуг.

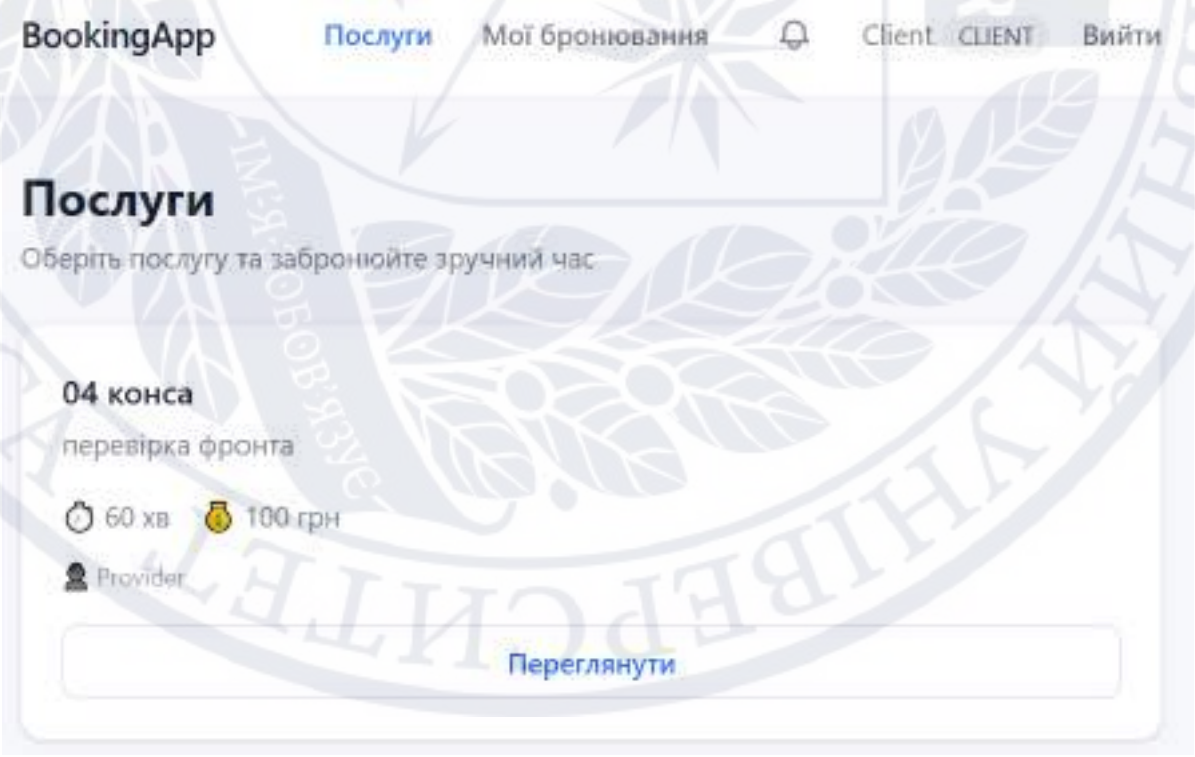


Рисунок 3.15 – Інтерфейс створення бронювання

Для адміністратора організовано сторінку зі своїм інтерфейсом, проте функціонування ще знаходяться в розробці (див. додаток В). Окремо реалізовано систему повідомлень на основі компонента NotificationBell, який показує кількість непрочитаних подій і відкриває перелік повідомлень із часом та типом. Передбачено оперативне оновлення лічильника, позначення як прочитаних і синхронізацію через WebSocket. На рисунку 3.16 наведено інтерфейс системи повідомлень.

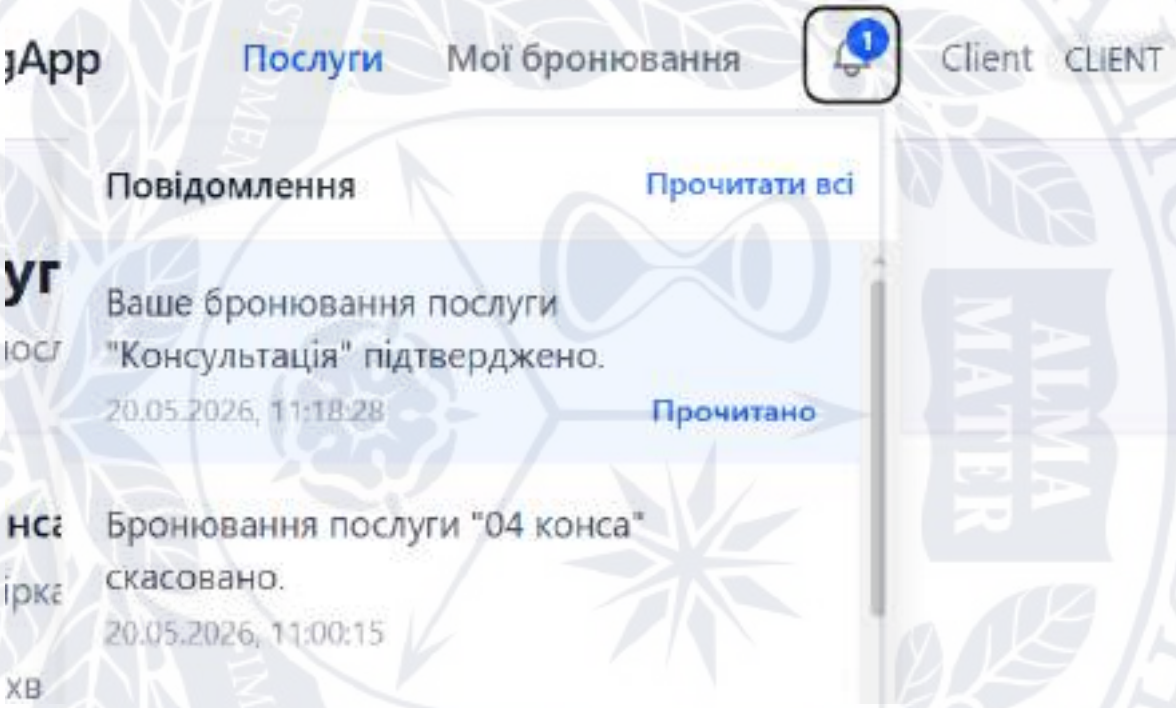


Рисунок 3.16 – Компонент повідомлень користувача

### 3.4.3 Реалізація frontend RBAC

Для розмежування доступу на клієнтській стороні впроваджено RBAC, узгоджений із серверною моделлю авторизації. Після входу роль фіксується в AuthContext і використовується для визначення доступного інтерфейсу. Основою стали Protected Routes: компонент маршрутизації перевіряє наявність JWT-токена та роль перед відкриттям сторінки; за відсутності авторизації виконується перенаправлення на сторінку входу. Додатково обмежено доступ до

окремих сторінок: керування послугами для PROVIDER, адміністративні функції для ADMIN, створення бронювання для CLIENT. На рисунку 3.17 наведено приклад реалізації Protected Route.

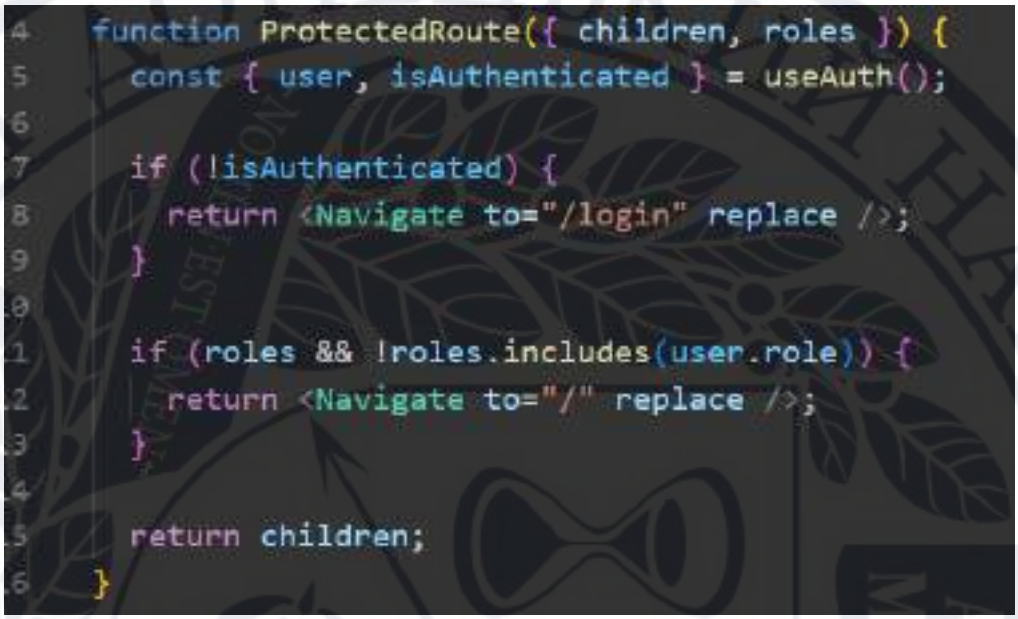


Рисунок 3.17 – Реалізація захищених маршрутів

Другий рівень RBAC – рольові інтерфейси: після входу користувачеві відображається лише функціональність його ролі. Також навігаційне меню формується динамічно залежно від ролі, що зменшує кількість недоступних елементів у інтерфейсі. На рисунках 3.18 – 3.19 наведено приклад адаптивного меню для різних ролей користувачів.

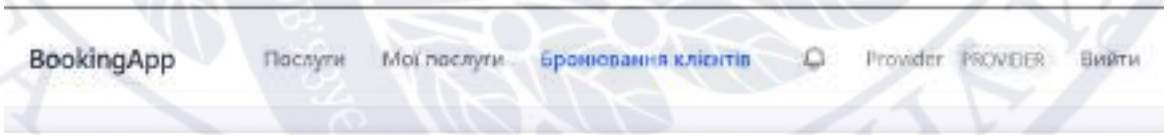


Рисунок 3.18 – Верхня панель для користувача з роллю PROVIDER


Рисунок 3.19 – Верхня панель для користувача з роллю CLIENT

У підсумку створено React-застосунок для системи бронювання з інтеграцією через REST API, підтримкою подій у реальному часі та рольовим керуванням доступом; обрана архітектура орієнтована на подальше розширення і супровід.

### 3.5 Реалізація real-time синхронізації

Однією з головних особливостей розробленої веб-платформи є підтримка взаємодії в режимі реального часу (real-time). На відміну від традиційного підходу, коли клієнтська частина періодично надсилає HTTP-запити для отримання оновлених даних, у розробленій системі використовується технологія WebSocket, яка дозволяє підтримувати постійне двостороннє з'єднання між клієнтом та сервером. Завдяки цьому користувачі миттєво отримують інформацію про зміни станів бронювань, оновлення доступних слотів та нові повідомлення без необхідності перезавантаження сторінки.

Для реалізації механізму синхронізації в режимі реального часу була використана бібліотека Socket.IO, яка забезпечує зручний інтерфейс для роботи з WebSocket та підтримує автоматичне відновлення з'єднання у випадку його втрати. Впровадження Socket.IO дозволило реалізувати подієво-орієнтовану архітектуру взаємодії між компонентами системи та забезпечити високу швидкість передачі даних між клієнтом і сервером.

#### 3.5.1 Реалізація WebSocket сервера

Після завершення REST API та базової бізнес-логіки створено окремий модуль WebSocket-сервера на Socket.IO, інтегрований з Express.js і запущений паралельно з обробкою HTTP-запитів. Під час старту застосунку ініціалізується екземпляр Socket.IO, прив’язаний до HTTP-сервера, після чого сервер очікує підключень клієнтів. На рисунку 3.20 наведено процес ініціалізації Socket.IO сервера.

```
const initSocket = (server) => {
  io = new Server(server, {
    cors: {
      origin: "http://localhost:5173",
      methods: ["GET", "POST", "PATCH", "DELETE"],
      credentials: true,
    },
  });
```

Рисунок 3.20 – Ініціалізація Socket.IO сервера

Після встановлення з'єднання клієнт передає JWT-токен для ідентифікації; сервер валідує токен і встановлює користувача, що дає змогу надсилати повідомлення адресно. Для персоналізованої доставки подій застосовано механізм rooms у Socket.IO: після успішного підключення користувач додається до власної кімнати.

Це дозволяє надсилати події конкретному користувачу без широкомовної розсилки всім підключеним клієнтам. На рисунку 3.21 наведено код використання rooms.

```
const emitToUser = (userId, event, payload) => {
  const socketIO = getIO();

  socketIO.to(`user:${userId}`).emit(event, payload);
};
```

Рисунок 3.21 – Використання rooms для персоналізованої доставки подій

Після налаштування rooms визначено набір WebSocket-подій, що відображають ключові бізнес-стани системи та синхронізують дані між клієнтом і сервером. Основними подіями стали booking_created, booking_confirmed,

booking_cancelled, slot_updated та notification_received. Кожна подія відповідає окремій бізнес-події всередині системи та використовується для узгодження станів між клієнтською і серверною частинами.

### 3.5.2 Реалізація event-driven взаємодії

Для автоматичної синхронізації застосовано подієво-орієнтований підхід, за яким зміни стану системи формують події та надсилаються відповідним користувачам. Подія booking_created генерується після успішного створення бронювання; після запису даних і формування системного повідомлення провайдер отримує сповіщення через WebSocket. Алгоритм роботи виглядає наступним чином:

- клієнт створює бронювання;
- сервер записує інформацію до бази даних;
- створюється системне повідомлення;
- генерується подія booking_created;
- провайдер отримує повідомлення через WebSocket.

Подію booking_confirmed використано для фіксації підтвердження провайдером: після зміни статусу сервер оновлює запис у bookings, формує повідомлення клієнту та надсилає booking_confirmed, що забезпечує негайне відображення нового статусу. Паралельно реалізовано notification_received для доставки системних повідомлень: повідомлення записується в notifications, визначається отримувач, після чого подія надсилається в кімнату користувача, без додаткових запитів до сервера. Подію slot_updated застосовано для узгодження доступних слотів; вона генерується при створенні або скасуванні бронювання, зміні статусу слоту чи додаванні нового слоту провайдером. Після отримання події клієнтська частина оновлює дані про доступність часу.

### 3.5.3 Реалізація real-time оновлення інтерфейсу

Після налаштування WebSocket-сервера та визначення подій Socket.IO інтегровано з клієнтською частиною. Для централізованого підключення створено SocketContext, який надає доступ до сокет-з'єднання компонентам React-застосунку. На рисунку 3.22 наведено підключення React-клієнта до Socket.IO сервера.

```
1  import { io } from "socket.io-client";
2
3  const SOCKET_URL = "http://localhost:5000";
4
5  const socket = io(SOCKET_URL, {
6    autoConnect: false,
7  });
8
9  export default socket;
```

Рисунок 3.22 – Підключення React до Socket.IO сервера

Автоматичне оновлення слотів реалізовано через обробку slot_updated: список доступного часу змінюється без повторного завантаження сторінки. Це зменшує ймовірність конфлікту вибору одного інтервалу кількома користувачами. Оновлення статусів бронювань реалізовано через події booking_confirmed і booking_cancelled: React оновлює локальний стан і відображає актуальний статус після надходження події. На рисунках 3.23 – 3.24 наведено початковий статус бронювання та приклад автоматичної зміни статусу після отримання WebSocket-події.

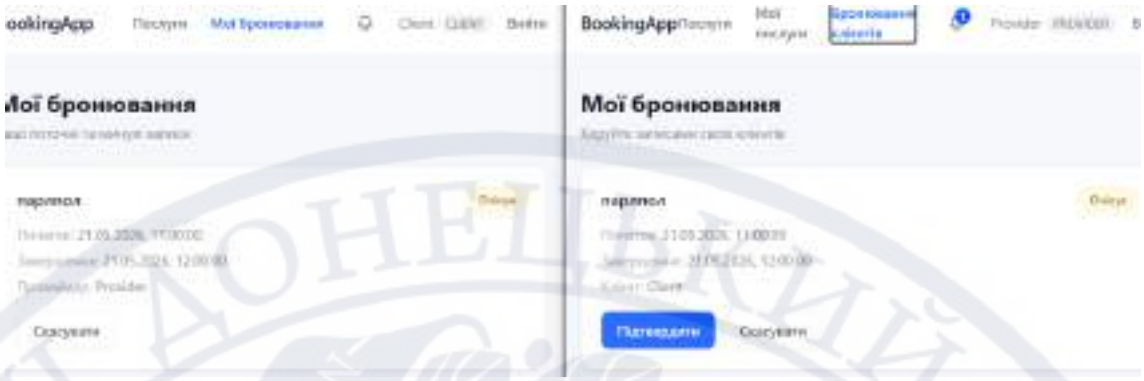


Рисунок 3.23 – Оновлення статусу бронювання в режимі реального часу

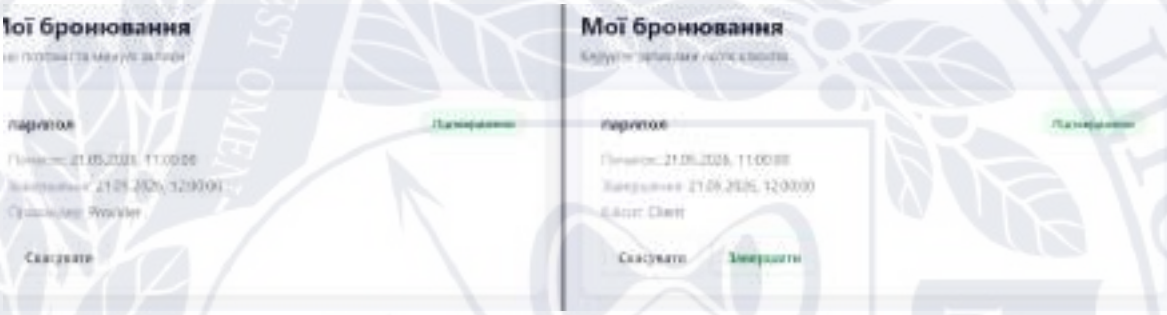


Рисунок 3.24 – Оновлення статусу бронювання в режимі реального часу

Система миттєвих повідомлень базується на notification_received: інтерфейс оновлює список, лічильник непрочитаного та відображає нове повідомлення відразу після появи події на сервері. На рисунку 3.25 показано, що після підтвердження бронювання автоматично оновлюється список повідомлень.

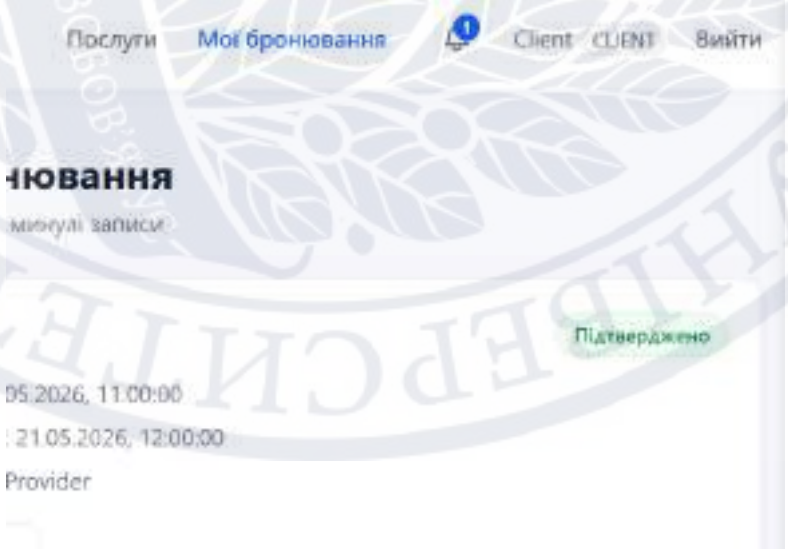


Рисунок 3.25 – Real-time оновлення системи повідомлень

У підсумку реалізовано механізм real-time синхронізації як обмін подіями між клієнтом і сервером. Поєднання Socket.IO, rooms і подієво-орієнтованої взаємодії забезпечує оперативне оновлення даних про бронювання, слоти та повідомлення, що визначає одну з функціонально важливих характеристик платформи.

## 3.6 Реалізація системи повідомлень

Веб-платформа містить підсистему повідомлень, що забезпечує оперативне інформування користувачів про події в процесі бронювання. Механізм автоматично сповіщає клієнтів і провайдерів про створення, підтвердження, скасування та завершення бронювань, підтримуючи актуальність інтерфейсу без ручного оновлення сторінки.

На відміну від підходу з періодичною перевіркою статусів, у системі застосовано доставку подій у реальному часі через Socket.IO, що зменшує затримки інформування та стабілізує узгодженість даних для учасників бронювання.

Базою підсистеми є таблиця notifications, спроєктована на етапі формування схеми даних. Запис повідомлення містить ідентифікатор отримувача, тип події, текст, час створення та ознаку прочитання.

### 3.6.1 Реалізація notifications

Для зберігання повідомлень визначено сутність Notification, яка акумулює всі сповіщення, сформовані під час виконання бізнес-операцій платформи. Модель включає атрибути id, userId, type, message, isRead та createdAt. Поле userId задає отримувача, тоді як type відображає категорію події. Для уніфікації обробки застосовано перелік типів: BOOKING_CREATED, BOOKING_CONFIRMED, BOOKING_CANCELLED, BOOKING_COMPLETED. Використання enum-типів стандартизує структуру повідомлень і спрощує їх

інтерпретацію на сервері та клієнті. На рисунку 3.26 наведено структуру моделі Notification у prisma Studio.


Рисунок 3.26 – Реалізація моделі Notification

Підтримку непрочитаних повідомлень реалізовано через поле isRead, яке під час створення має значення isRead = false. Після перегляду користувачем значення змінюється на isRead = true (рис. 3.27). Це дає змогу обчислювати кількість непрочитаних записів і відображати відповідний індикатор у інтерфейсі.

```
await prisma.notification.updateMany({
  where: {
    userId,
    isRead: false,
  },
  data: {
    isRead: true,
  },
```

Рисунок 3.27 – Логіка зміни статусу повідомлення

Для доступу до повідомлень реалізовано API-модуль, який підтримує такі операції:

- отримання списку повідомлень користувача;
- отримання кількості непрочитаних повідомлень;
- позначення повідомлення як прочитаного;
- позначення всіх повідомлень як прочитаних.

Фільтрація здійснюється за ідентифікатором поточного користувача, що обмежує доступ до чужих повідомлень. Автоматичне створення повідомлень забезпечується серверними тригерами бізнес-логіки, які формують записи в notifications за настання ключових подій. Основні тригери:

- створення бронювання;
- підтвердження бронювання;
- скасування бронювання;
- завершення бронювання.

У результаті зміни в системі доводяться до користувачів без додаткових дій з їхнього боку. На рисунку 3.28 наведено процес автоматичного створення повідомлення під час виконання бізнес-операції.

```
await notificationService.createNotification({
  userId: booking.providerId,
  type: "BOOKING_CREATED",
  message: `У вас нове бронювання послуги "${booking.service.name}".`,
});
```

Рисунок 3.28 – Автоматична генерація повідомлень

### 3.6.2 Інтеграція повідомлень із системою бронювання

Підсистема повідомлень реалізована як складова бізнес-логіки бронювання, а не як автономний модуль; її поведінка визначається станом бронювання та відповідними подіями. Для інтеграції застосовано подієво-

орієнтовану модель (Event-Driven Architecture), за якою кожна значуща бізнес-подія ініціює формування повідомлення. Під час створення бронювання після запису в bookings сервер виконує послідовність дій:

- створює бронювання;
- змінює статус відповідного слоту;
- створює повідомлення для провайдера;
- надсилає WebSocket-подію користувачу.

Під час підтвердження бронювання після встановлення статусу CONFIRMED система формує повідомлення для клієнта та передає його через WebSocket. Процес можна описати схемою, яка показана на рисунку 3.29.


Рисунок 3.29 – Процес обробки події підтвердження бронювання та доставки повідомлення клієнту

Це забезпечує швидке відображення зміни стану бронювання в інтерфейсі клієнта. Скасування бронювання обробляється аналогічно: незалежно від ініціатора система створює повідомлення для іншої сторони та узгоджує стан даних між компонентами платформи.

Доставка в реальному часі реалізована через WebSocket-з'єднання на основі Socket.IO. Після створення повідомлення сервер генерує подію notification_received. Подія надсилається конкретному користувачу через персональну кімнату (room), після чого клієнт автоматично оновлює перелік повідомлень і лічильник непрочитаних. Застосований механізм підтримує узгодженість між модулем бронювання, сервером та інтерфейсом: зміни стану бронювання відображаються без додаткових HTTP-запитів.

Отже, підсистема повідомлень є функціональною складовою платформи онлайн-бронювання, що автоматизує інформування користувачів і забезпечує доставку подій у реальному часі. Інтеграція з бронюванням у поєднанні з Socket.IO та подієво-орієнтованою взаємодією забезпечує оперативність сповіщень і синхронізацію компонентів системи.

### 3.7 Демонстрація роботи веб-платформи

У цьому підрозділі демонструється типовий сценарій взаємодії користувача із системою онлайн-бронювання. Наведений приклад охоплює повний цикл створення бронювання: від авторизації користувача до підтвердження бронювання та отримання відповідних повідомлень у режимі реального часу.

На першому етапі користувач проходить процедуру авторизації шляхом введення електронної пошти та пароля (див. Рис. 3.30). Після успішної перевірки облікових даних система надає доступ до персонального кабінету та функцій відповідно до ролі користувача.

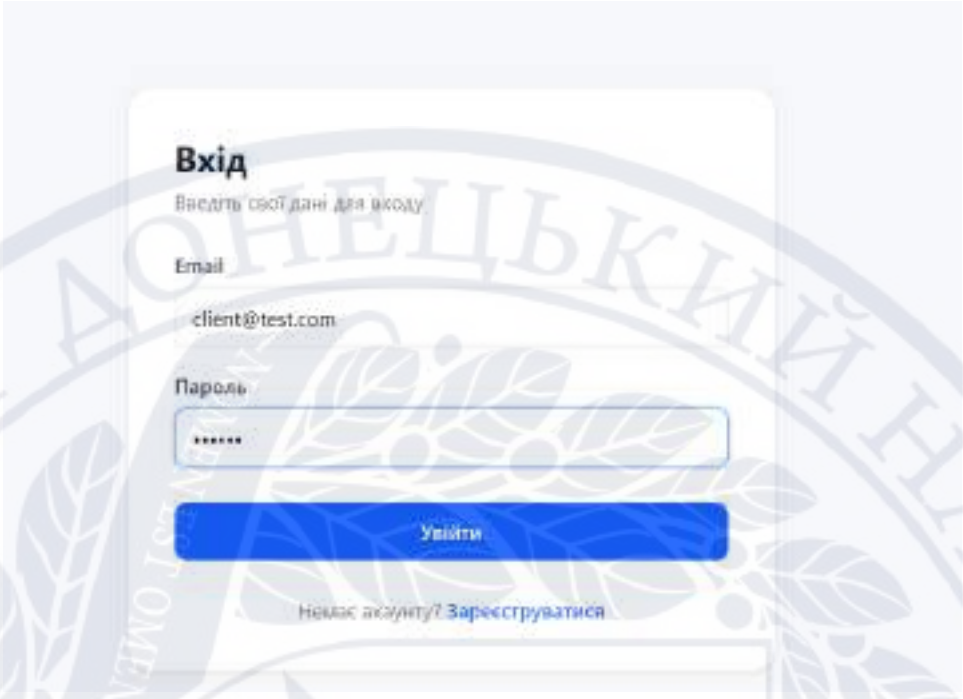


Рисунок 3.30 – Сторінка авторизації користувача

Після входу користувач переходить до сторінки провайдера послуг, де може переглянути доступні послуги та обрати необхідний тип бронювання. Інтерфейс показаний на рисунку 3.31.

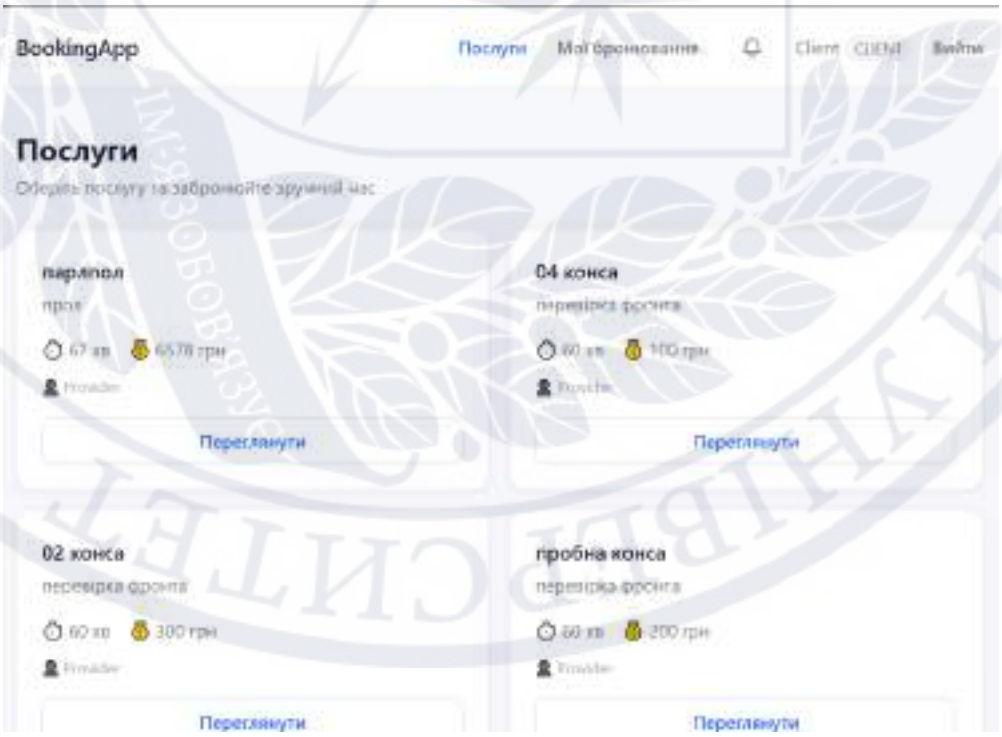


Рисунок 3.31 – Перегляд профілю провайдера та списку послуг

Після вибору послуги система відображає доступні часові інтервали. Користувач обирає бажаний слот і підтверджує створення бронювання. Бронювання слота показано на рисунку 3.32.

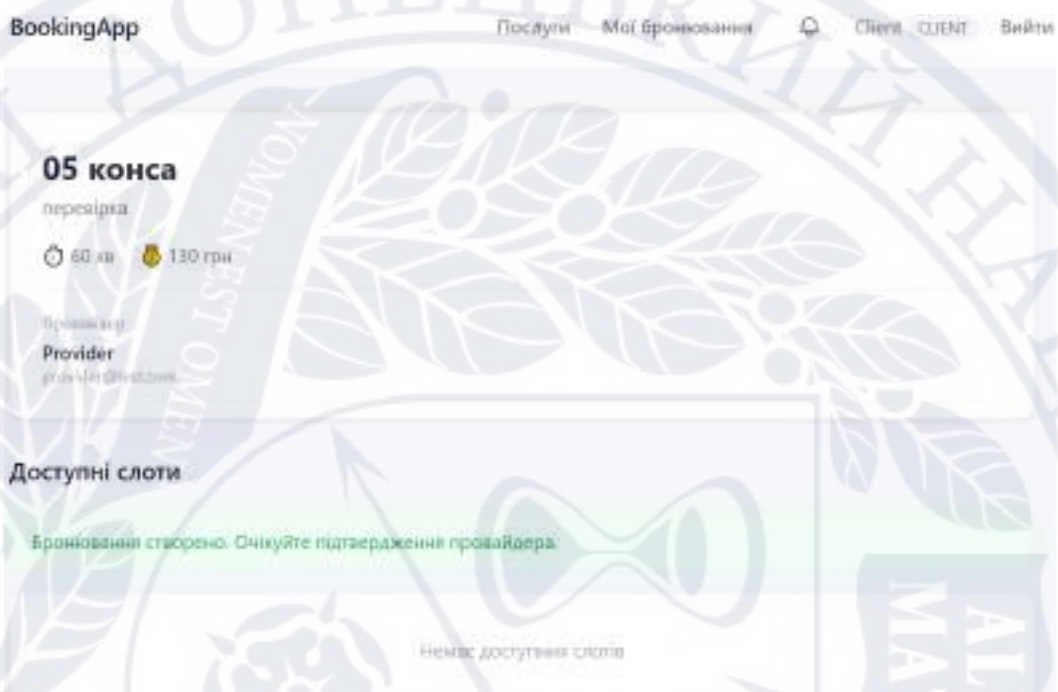


Рисунок 3.32 – Вибір і бронювання доступного часового слоту

Після успішного створення запису система формує нове бронювання зі статусом PENDING та зберігає його у базі даних. Бронювання відображається на сторінці провайдера, як показано на рисунку 3.33.

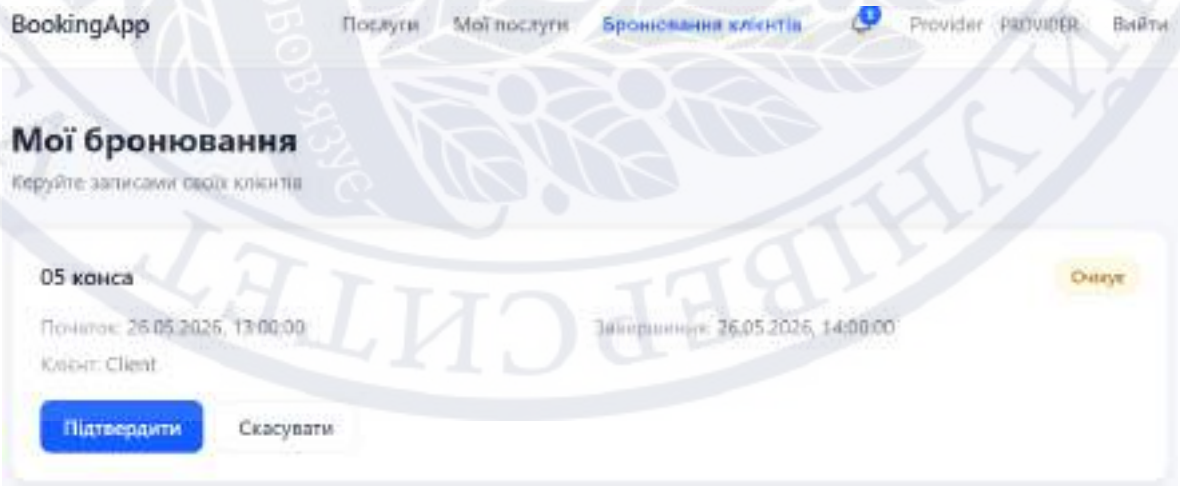


Рисунок 3.33 – Створене бронювання зі статусом PENDING

Одночасно провайдер отримує внутрішнє повідомлення про новий запит на бронювання (див. Рис. 3.34). Доставка повідомлення здійснюється автоматично через WebSocket без необхідності оновлення сторінки.

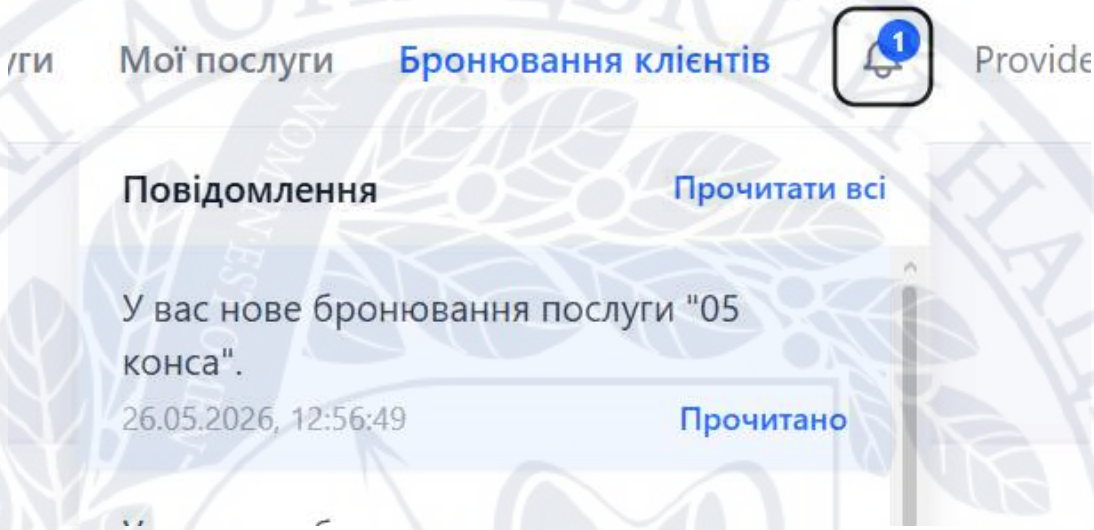


Рисунок 3.34 – Отримання повідомлення про нове бронювання

Після підтвердження бронювання провайдером статус запису змінюється на CONFIRMED. Відповідна зміна миттєво відображається в інтерфейсі клієнта, що підтверджує коректну роботу механізму синхронізації в режимі реального часу. Оновлений статус на сторінці клієнта показано на рисунку 3.35.


Рисунок 3.35 – Оновлення статусу бронювання після підтвердження

Після надання послуги, провайдер позначає бронювання як виконане і його статус змінюється на COMPLETED. Це так само відображається в інтерфейсі клієнта, який до того ж отримує повідомлення про завершення.

Таким чином, наведений сценарій демонструє взаємодію всіх основних компонентів системи: механізму авторизації, підсистеми бронювання, системи повідомлень та технології real-time синхронізації.

### 3.8 Перевірка працездатності системи

Після завершення реалізації веб-платформи онлайн-бронювання було проведено перевірку працездатності її основних підсистем. Метою перевірки було підтвердження коректної роботи реалізованого функціоналу, зокрема механізмів автентифікації та авторизації користувачів, системи бронювання, зміни станів бронювань, внутрішніх повідомлень та синхронізації даних у режимі реального часу. Перевірка виконувалася шляхом послідовного відтворення типових сценаріїв використання системи, визначених під час проєктування.

На початковому етапі було досліджено можливість створення нового облікового запису, здійснення входу до системи за допомогою електронної пошти та пароля, процедура отримання JWT-токена, а також доступність захищених ресурсів системи. На рисунку 3.36 можна побачити JWT-токен, який означає успішну авторизацію.

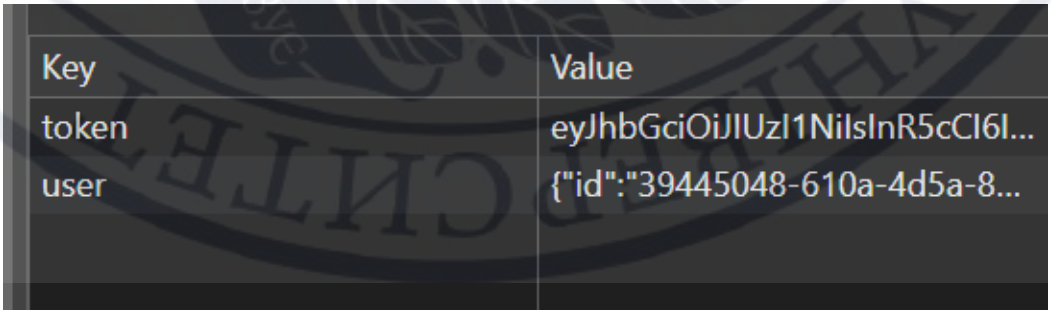


Рисунок 3.36 – Збереження JWT-токена після успішної авторизації

Також на прикладі акаунту провайдера, ми можемо побачити, що він не може мати доступ до функцій іншої ролі, адже frontend просто ховає непотрібні кнопки, як на рисунку 3.37.

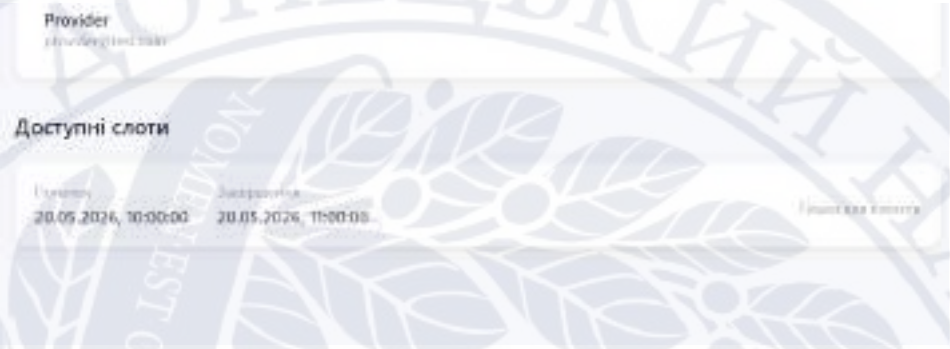


Рисунок 3.37 – Обмеження доступу до функцій бронювання для користувача ролі Provider

Наступний етап передбачав перевірку основної бізнес-логіки системи бронювання. Було проаналізовано функціональність створення бронювань, зміни їхнього статусу згідно з FSM-моделлю, а також здійснення дозволених переходів між станами. В цьому випадку frontend так само не дає можливості порушити логіку переходів між станами. На рисунку 3.38 наведено записи таблиці bookings, що демонструють використання різних станів життєвого циклу бронювання відповідно до FSM-моделі.

| | | |
|---|---|---|
| .. | 9ea157a3-1a24-47b9-acec… | CANCELLED |
| .. | 12ba05e4-6b99-4c98-a608… | CONFIRMED |
| .. | 864221c9-40f8-48c9-a078… | PENDING |
| .. | 980f3923-0a0e-45ac-81a7… | COMPLETED |
| .. | 2de27951-7fef-4e36-96a1… | CONFIRMED |
| .. | 9a38766c-514b-464c-844e… | CONFIRMED |
| .. | 52de8a16-030c-4169-8922… | PENDING |
| .. | 07acfdfe-674b-45e2-94ad… | CANCELLED |

Рисунок 3.38 – Приклади станів бронювань у базі даних

Додатково було проведено перевірку механізму запобігання конфліктам бронювання. Із цією метою було здійснено спробу створити два бронювання для одного і того ж часового слоту. Результати показали, що перше бронювання було успішно сформовано, тоді як другий запит не можливо було навіть створити бо статус слоту моментально змінювався і кнопка пропадала. Цей результат підтвердив коректність реалізації перевірки доступності часових слотів та функціонування механізмів захисту від подвійного бронювання.

Завершальний етап включав перевірку підсистеми внутрішніх повідомлень та механізму синхронізації даних у режимі реального часу. Верифікація проводилася шляхом одночасного функціонування системи під роллю клієнта та провайдера послуг у двох незалежних браузерних сесіях. При створенні нового бронювання клієнтом система автоматично генерувала внутрішнє повідомлення для провайдера, що можна побачити на рисунку 3.39. Після підтвердження бронювання провайдером, відповідне сповіщення надсилалося клієнту. Усі повідомлення відображалися без необхідності перезавантаження сторінки завдяки використанню технології WebSocket та бібліотеки Socket.IO.

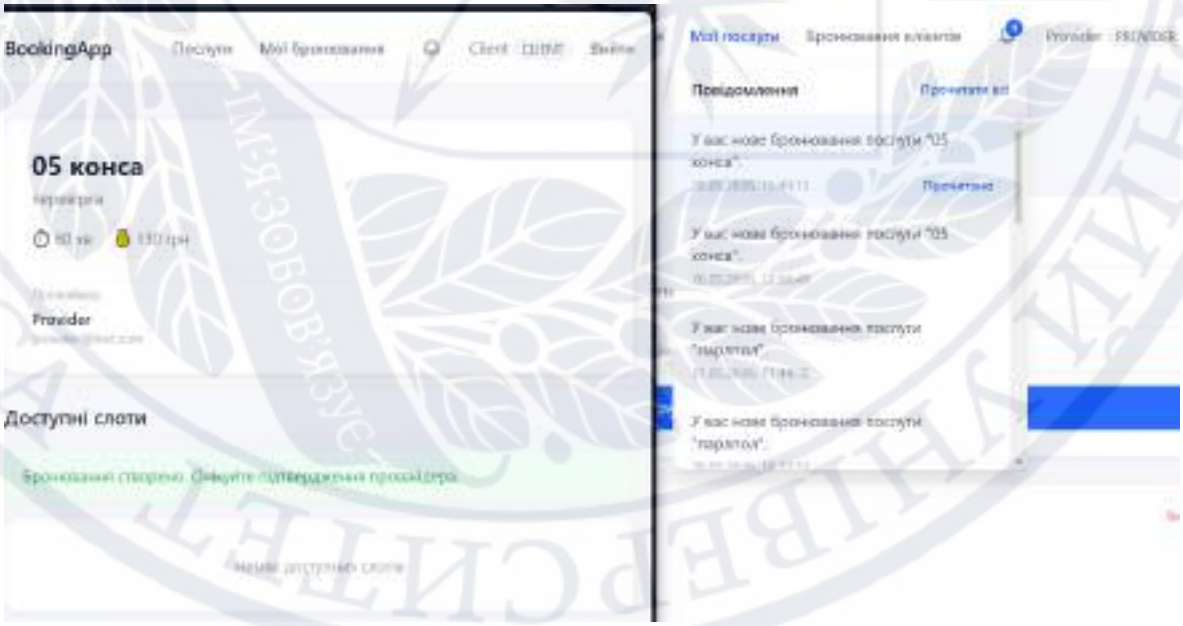


Рисунок 3.39 – Автоматичне створення повідомлення після бронювання

Перевірка усіх інших повідомлень також була проведена та дала позитивний результат.

Крім того, було перевірено механізм синхронізації часових слотів між різними користувачами. Після створення або зміни статусу бронювання відповідні зміни автоматично відображалися в інтерфейсах усіх зацікавлених користувачів, що підтвердило працездатність реалізованої системи обміну подіями в режимі реального часу. На рисунку 3.40 видно, що після бронювання слоту одним клієнтом у іншого він зникає.

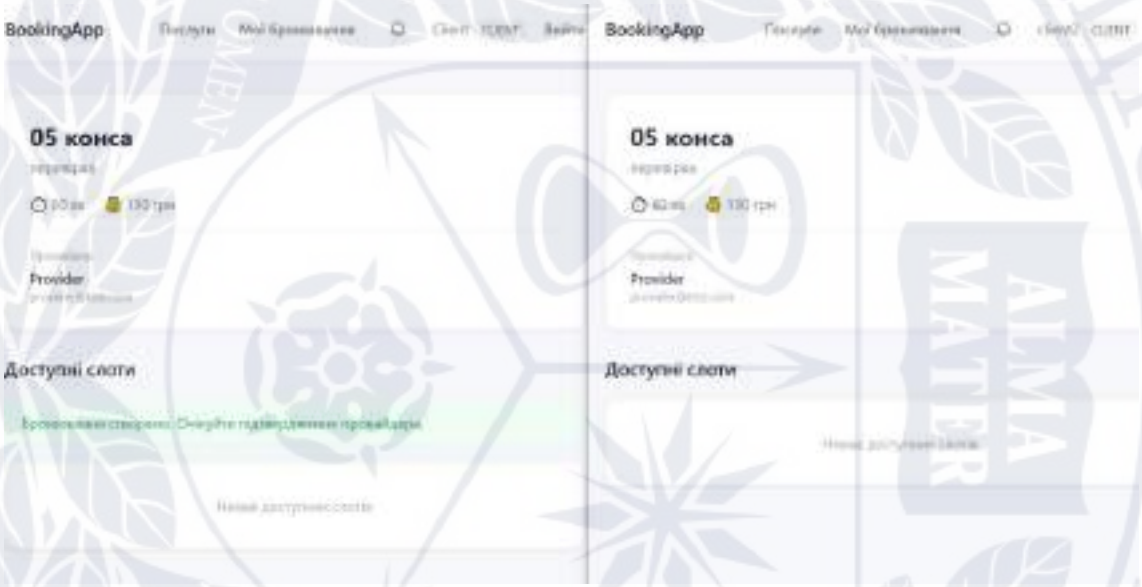


Рисунок 3.40 – Синхронізація статусу бронювання в режимі реального часу

Таким чином, проведена перевірка працездатності підтвердила коректне функціонування всіх основних компонентів веб-платформи. Було встановлено, що механізми авторизації та рольового доступу працюють відповідно до визначених вимог, система бронювання забезпечує коректне створення та обробку записів, а підсистема повідомлень і технологія WebSocket гарантують своєчасне оновлення інформації між користувачами. Отримані результати свідчать про працездатність розробленої веб-платформи та її готовність до використання відповідно до поставлених функціональних вимог.

# ВИСНОВКИ ДО РОЗДІЛУ 3

У третьому розділі представлено програмну реалізацію веб-платформи для онлайн-бронювання послуг. Ця реалізація грунтується на архітектурних рішеннях та вимогах, визначених на попередніх етапах дослідження. Для побудови системи було використано стек веб-технологій, що включає React для клієнтської частини, Node.js та Express.js для серверної логіки, PostgreSQL для зберігання даних і Prisma ORM для взаємодії з базою даних.

Під час реалізації було розроблено структуру бази даних, яка інтегрує сутності користувачів, послуг, часових слотів, бронювань та повідомлень. Для забезпечення цілісності даних використано зовнішні ключі, обмеження та індекси. Механізм міграцій було налаштовано за допомогою Prisma ORM, що дозволило забезпечити достовірне зберігання інформації та підтримку зв’язків між ключовими компонентами системи.

На серверній стороні розроблено REST API, що підтримує операції авторизації, управління послугами, бронюваннями та повідомленнями. Безпека системи забезпечена за допомогою JWT-аутентифікації та моделі рольового доступу (RBAC), яка диференціює права доступу для клієнтів, провайдерів послуг та адміністраторів. Крім того, реалізовано бізнес-логіку бронювання, що включає контроль допустимих переходів між станами та перевірку конфліктів часових слотів.

Клієнтська частина веб-платформи забезпечує взаємодію користувачів із системою за допомогою веб-інтерфейсу. Вона включає сторінки для авторизації, перегляду послуг, створення бронювань, управління часовими слотами та перегляду повідомлень. Додатково реалізовано механізми захисту маршрутів і динамічного відображення функціоналу згідно з роллю користувача.

Особливу увагу приділено реалізації системи повідомлень та механізму синхронізації даних у режимі реального часу. Використання технології WebSocket та бібліотеки Socket.IO дозволило забезпечити негайне оновлення статусів бронювань, повідомлень та інформації про доступність часових слотів

без необхідності перезавантаження веб-сторінок. Цей підхід сприяє підвищенню ефективності взаємодії користувачів та гарантує актуальність інформації в системі.

Проведена перевірка працездатності підтвердила коректне функціонування основних підсистем веб-платформи. Було успішно протестовано механізми авторизації, рольового доступу, бронювання, внутрішніх повідомлень та синхронізації даних у режимі реального часу. Отримані результати демонструють відповідність реалізації встановленим вимогам та підтверджують готовність веб-платформи до подальшого впровадження.

# ВИСНОВКИ

У ході виконання дипломної роботи було розглянуто проблему організації онлайн-бронювання послуг та особливості побудови сучасних веб-платформ, призначених для автоматизації процесів взаємодії між клієнтами і надавачами послуг. Проведений аналіз предметної області показав, що існуючі рішення успішно реалізують базові механізми бронювання, проте часто мають обмеження щодо оперативності оновлення даних, синхронізації інформації між користувачами та підтримки взаємодії в режимі реального часу.

У першому розділі було досліджено особливості систем онлайн-бронювання, проаналізовано сучасні платформи Calendly, SimplyBook.me, Booksy, Setmore та Acuity Scheduling, визначено їх функціональні можливості та характерні недоліки. На основі проведеного аналізу сформовано функціональні та нефункціональні вимоги до майбутньої системи, визначено цільову аудиторію та основні сценарії використання веб-платформи.

У другому розділі виконано проєктування програмної системи. Було розроблено загальну архітектуру веб-платформи на основі клієнт-серверного підходу, спроєктовано механізм бронювання, життєвий цикл бронювання на основі моделі скінченного автомата станів, систему внутрішніх повідомлень та структуру бази даних. Окрему увагу приділено реалізації рольового керування доступом та організації обміну даними в режимі реального часу за допомогою технології WebSocket.

У третьому розділі здійснено програмну реалізацію веб-платформи із використанням сучасних веб-технологій. Клієнтську частину розроблено на базі React, серверну частину реалізовано із застосуванням Node.js та Express, а для зберігання даних використано PostgreSQL і Prisma ORM. Реалізовано механізми авторизації на основі JWT-токенів, систему бронювання з перевіркою конфліктів часових слотів, модель рольового доступу RBAC, підсистему внутрішніх повідомлень та синхронізацію даних у режимі реального часу за допомогою Socket.IO.

Для перевірки працездатності системи було виконано тестування основних функціональних можливостей. Результати перевірки підтвердили коректну роботу механізмів авторизації, рольового доступу, створення та обробки бронювань, системи повідомлень і технології синхронізації в режимі реального часу. Також було підтверджено працездатність механізму запобігання подвійним бронюванням та коректність реалізації життєвого циклу бронювання.

Таким чином, поставлена мета роботи була досягнута. Розроблено веб-платформу для онлайн-бронювання послуг із реалізацією рольового доступу, системи внутрішніх повідомлень та підтримкою обміну даними в режимі реального часу. Створене програмне рішення забезпечує автоматизацію процесів бронювання, підвищує оперативність інформування користувачів і може слугувати основою для подальшого розвитку та впровадження додаткових функціональних можливостей.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. WebSocket architecture: 8 best practices for building real-time apps. *Ably Realtime*. [Електронний ресурс]. URL: https://ably.com/topic/websocket-architecture-best-practices (дата звернення: 28.03.2026).
2. Shivani M. System Design Concept: Web Sockets. *Medium*. [Електронний ресурс]. URL: https://medium.com/@shivanimutke2501/day-51-system-design-concept-web-sockets-5fc63cc4ca07 (дата звернення: 28.03.2026).
3. WebSockets API. *MDN Web Docs*. [Електронний ресурс]. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (дата звернення: 28.03.2026).
4. Getting Started with WebSockets in Node.js. *Node.js*. [Електронний ресурс]. URL: https://nodejs.org/learn/getting-started/websocket (дата звернення: 29.03.2026).
5. Express – Fast, unopinionated, minimalist web framework for Node.js. [Електронний ресурс]. URL: https://expressjs.com/ (дата звернення: 29.03.2026).
6. Django Channels Documentation. [Електронний ресурс]. URL: https://channels.readthedocs.io/en/latest/ (дата звернення: 01.04.2026).
7. WebSockets in FastAPI. *FastAPI*. [Електронний ресурс]. URL: https://fastapi.tiangolo.com/advanced/websockets/ (дата звернення: 01.04.2026).
8. Socket.io: The fastest and most reliable real-time engine. [Електронний ресурс]. URL: https://socket.io/ (дата звернення: 01.04.2026).
9. What is Role-Based Access Control (RBAC)? *IBM*. [Електронний ресурс]. URL: https://www.ibm.com/think/topics/rbac (дата звернення: 01.04.2026).
10. Choudhary J. Building Role-Based Access Control (RBAC) in Node.js and Express.js. *Medium*. [Електронний ресурс]. URL: https://medium.com/@jayantchoudhary271/building-role-based-access-

control-rbac-in-node-js-and-express-js-bc870ec32bdb (дата звернення: 01.04.2026).

11. Mitra T. The Timeslottr Algorithm for efficient booking. [Електронний ресурс]. URL: https://www.tilomitra.com/blog/timeslottr-algorithm (дата звернення: 02.04.2026).
12. Hotel reservation schema design with PostgreSQL. *Dev.to*. [Електронний ресурс]. URL: https://dev.to/chandra179/hotel-reservation-schema-design-postgresql-3i9j (дата звернення: 03.04.2026).
13. Оптимізація структур даних для систем бронювання. *DOU*. [Електронний ресурс]. URL: https://dou.ua/forums/topic/40868/ (дата звернення: 04.04.2026).
14. Bala S. Hotel booking schema design comparison. *Dev.to*. [Електронний ресурс]. URL: https://dev.to/sumedhbala/hotel-booking-schema-design-comparison-g3h (дата звернення: 04.04.2026).
15. Calendly. [Електронний ресурс]. URL: https://calendly.com/ (дата звернення: 04.04.2026).
16. SimplyBook. [Електронний ресурс]. URL: https://simplybook.me/uk/ (дата звернення: 05.04.2026).
17. Booksy. [Електронний ресурс]. URL: https://booksy.com/uk-us/ (дата звернення: 07.04.2026).
18. Setmore. [Електронний ресурс]. URL: https://www.setmore.com/ (дата звернення: 08.04.2026).
19. AcuityScheduling. [Електронний ресурс]. URL: https://acuityscheduling.com/ (дата звернення: 10.04.2026).
20. What is an online booking system? *TechTarget*. [Електронний ресурс]. URL: https://www.techtarget.com/searchsoftwarequality/definition/online-booking-system (дата звернення: 14.04.2026).
21. Optimistic Concurrency Control Pattern. *Microsoft Learn*. [Електронний ресурс]. URL: https://learn.microsoft.com/en-

us/azure/architecture/patterns/optimistic-concurrency-pattern (дата звернення: 18.04.2026).

22. What is client-server architecture? *Red Hat*. [Електронний ресурс]. URL: https://www.redhat.com/en/topics/api/what-is-client-server-architecture (дата звернення: 20.04.2026).
23. What is REST? *RESTful API Design*. [Електронний ресурс]. URL: https://restfulapi.net/ (дата звернення: 22.04.2026).
24. Online booking system definition and features. *TechTarget*. [Електронний ресурс]. URL: https://www.techtarget.com/searchcustomerexperience/definition/online-booking-system (дата звернення: 25.04.2026).
25. Software Advice: What is an online booking system? [Електронний ресурс]. URL: https://www.softwareadvice.com/resources/what-is-an-online-booking-system/ (дата звернення: 28.04.2026).
26. WebSockets vs HTTP Streaming vs SSE. *Ably Blog*. [Електронний ресурс]. URL: https://ably.com/blog/websockets-vs-http-streaming-vs-sse (дата звернення: 30.04.2026).
27. Real-time web apps with WebSockets. *Pusher Blog*. [Електронний ресурс]. URL: https://pusher.com/blog/websockets-realtime-web-apps/ (дата звернення: 01.05.2026).
28. Role-based access control (RBAC) explained. *TechTarget*. [Електронний ресурс]. URL: https://www.techtarget.com/searchsecurity/definition/role-based-access-control-RBAC (дата звернення: 04.05.2026).
29. Designing a perfect booking system. *Smashing Magazine*. [Електронний ресурс]. URL: https://www.smashingmagazine.com/2020/04/designing-perfect-booking-system/ (дата звернення: 05.05.2026).
30. Authentication and Authorization Flow. *Auth0 Docs*. [Електронний ресурс]. URL: https://auth0.com/docs/get-started/authentication-and-authorization-flow (дата звернення: 05.05.2026).

31. OWASP Top Ten Project. [Електронний ресурс]. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 05.05.2026).
32. Express.js Documentation. [Електронний ресурс]. URL: https://expressjs.com/ (дата звернення: 05.05.2026).
33. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT) (RFC 7519). [Електронний ресурс]. URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 07.05.2026).
34. Sandhu R., Coyne E., Feinstein H., Youman C. Role-Based Access Control Models. *IEEE Computer*. 1996. Vol. 29, No. 2. P. 38–47.
35. Ferraiolo D., Kuhn R. Role-Based Access Controls. *15th National Computer Security Conference*. 1992. P. 554–563.
36. IETF. The WebSocket Protocol (RFC 6455). [Електронний ресурс]. URL: https://datatracker.ietf.org/doc/html/rfc6455 (дата звернення: 08.05.2026).
37. PostgreSQL Documentation. Exclusion Constraints. [Електронний ресурс]. URL: https://www.postgresql.org/docs/current/ddl-constraints.html#DDL-CONSTRAINTS-EXCLUSION (дата звернення: 08.05.2026).
38. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston : Addison-Wesley, 1994. 395 p.
39. Connolly T., Begg C. Database Systems: A Practical Approach to Design, Implementation, and Management. 6th ed. Boston : Pearson, 2015. 1440 p.
40. PostgreSQL Global Development Group. PostgreSQL Documentation – Range Types and Exclusion Constraints. [Електронний ресурс]. 2024. URL: https://www.postgresql.org/docs/current/rangetypes.html (дата звернення: 10.05.2026).

# ДОДАТКИ

# ДОДАТОК А

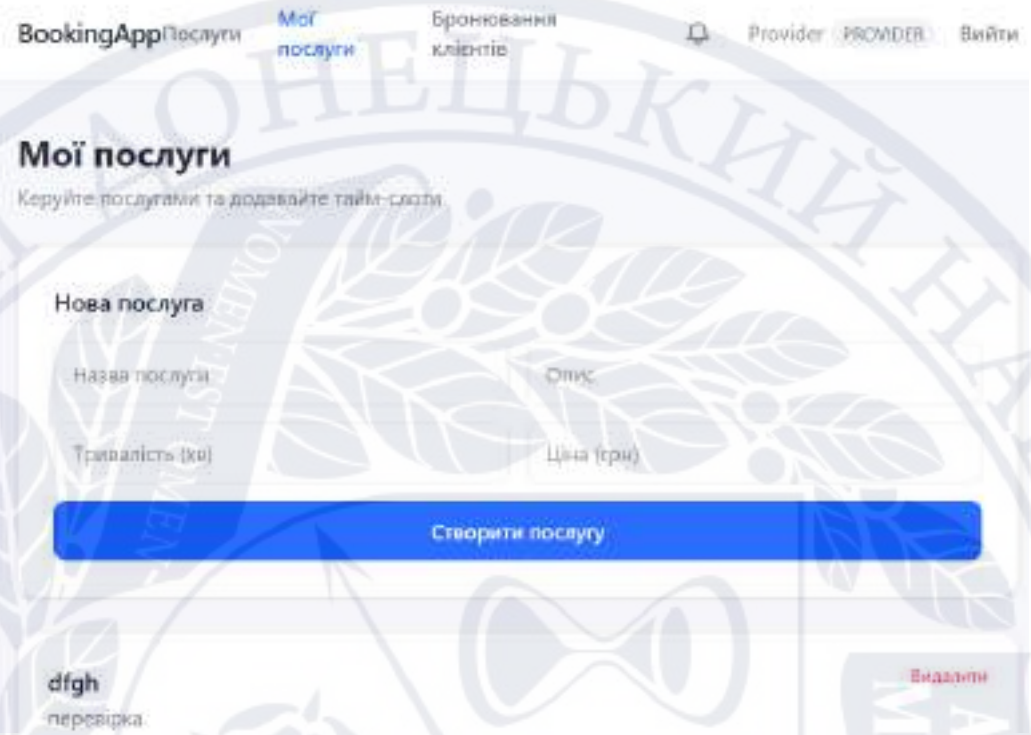


Рис. А.1 – Сторінка створення і керування послугами надавача послуг

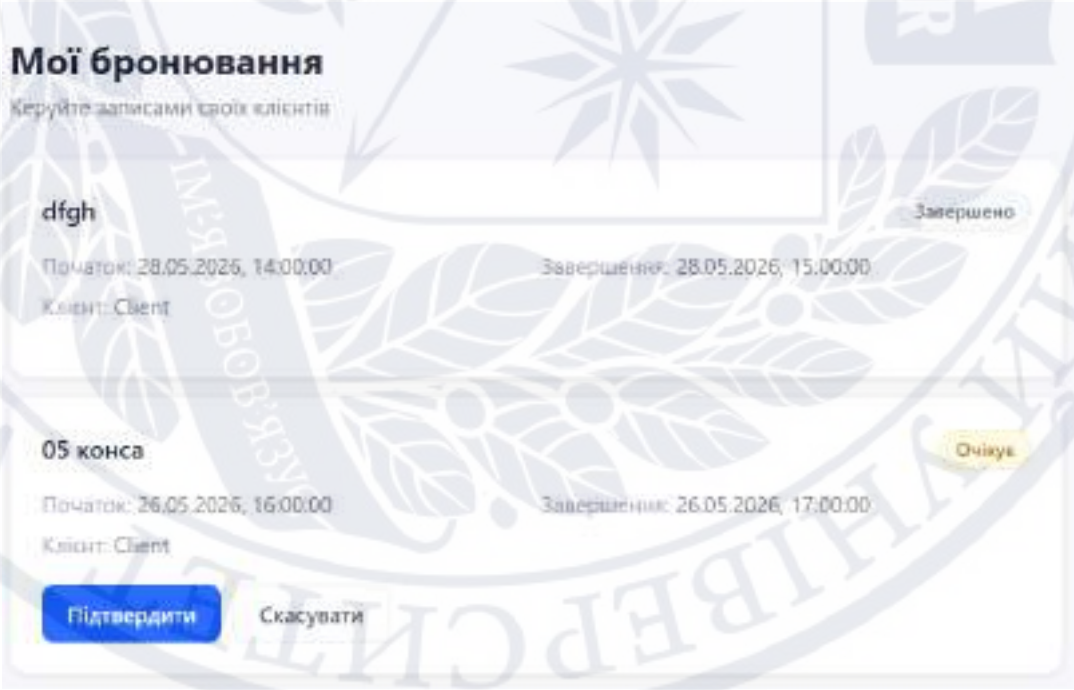


Рис. А.2 – Сторінка керування бронюваннями клієнтів надавачем послуг

# ДОДАТОК Б

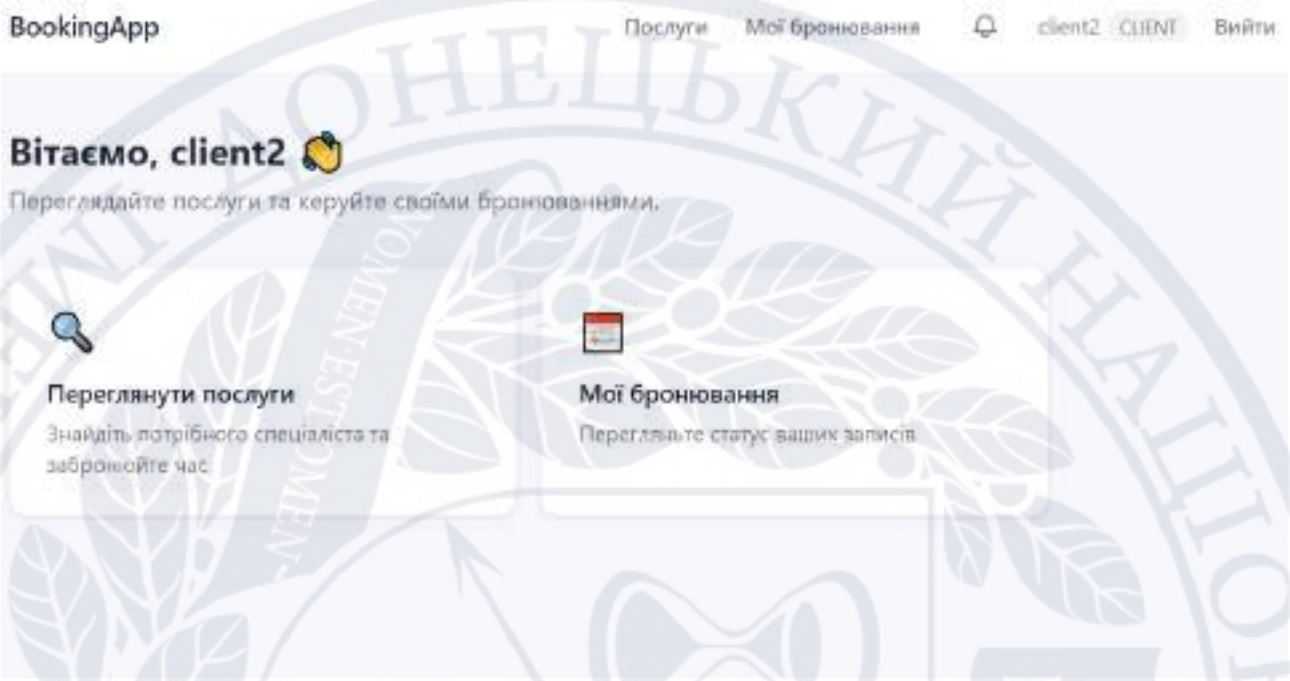


Рис. Б.1 – Головна сторінка клієнта після авторизації

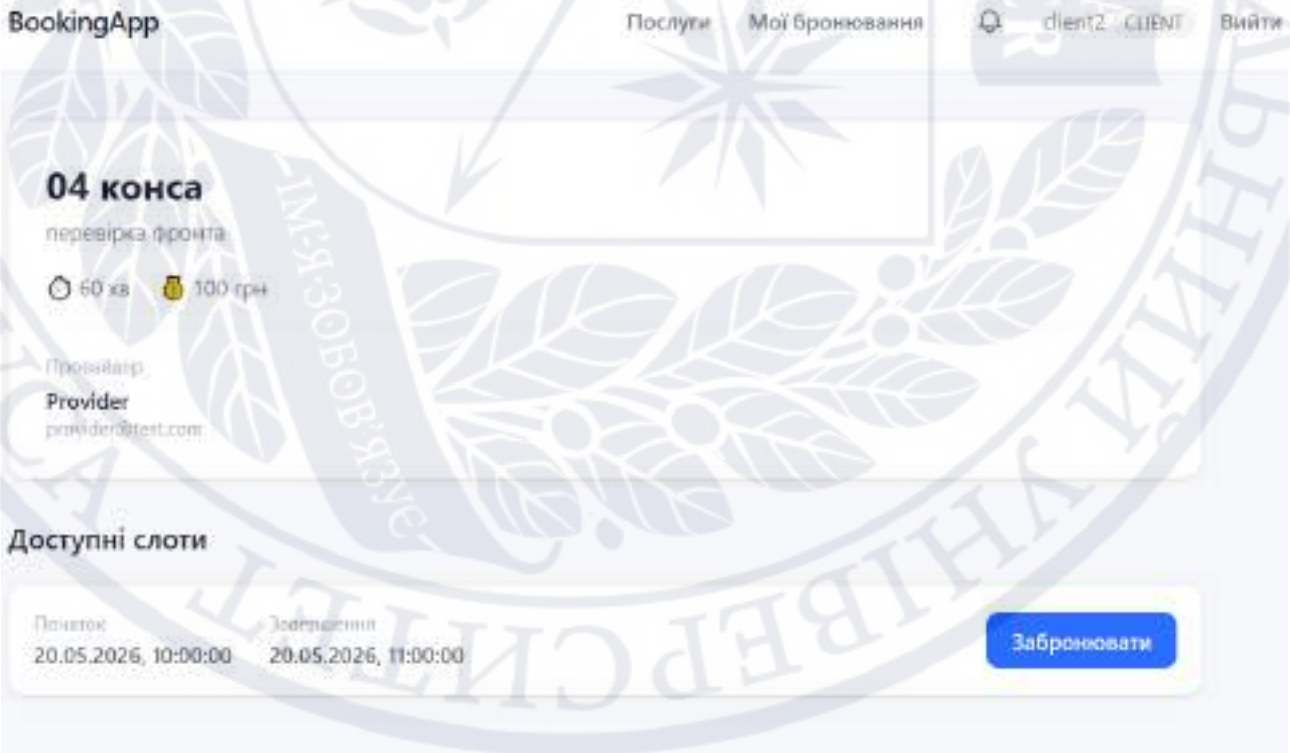


Рис. Б.2 – Перегляд клієнтом інформації про послугу та доступних часових слотів

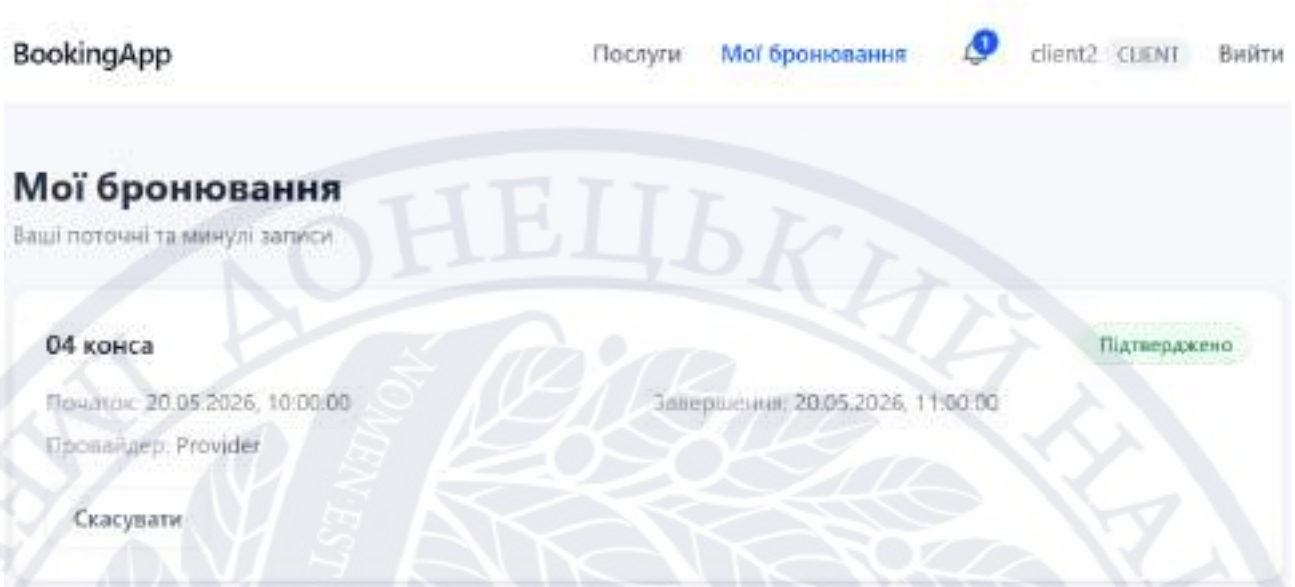


Рис. Б.3 – Сторінка перегляду своїх бронювань клієнтом

# ДОДАТОК В

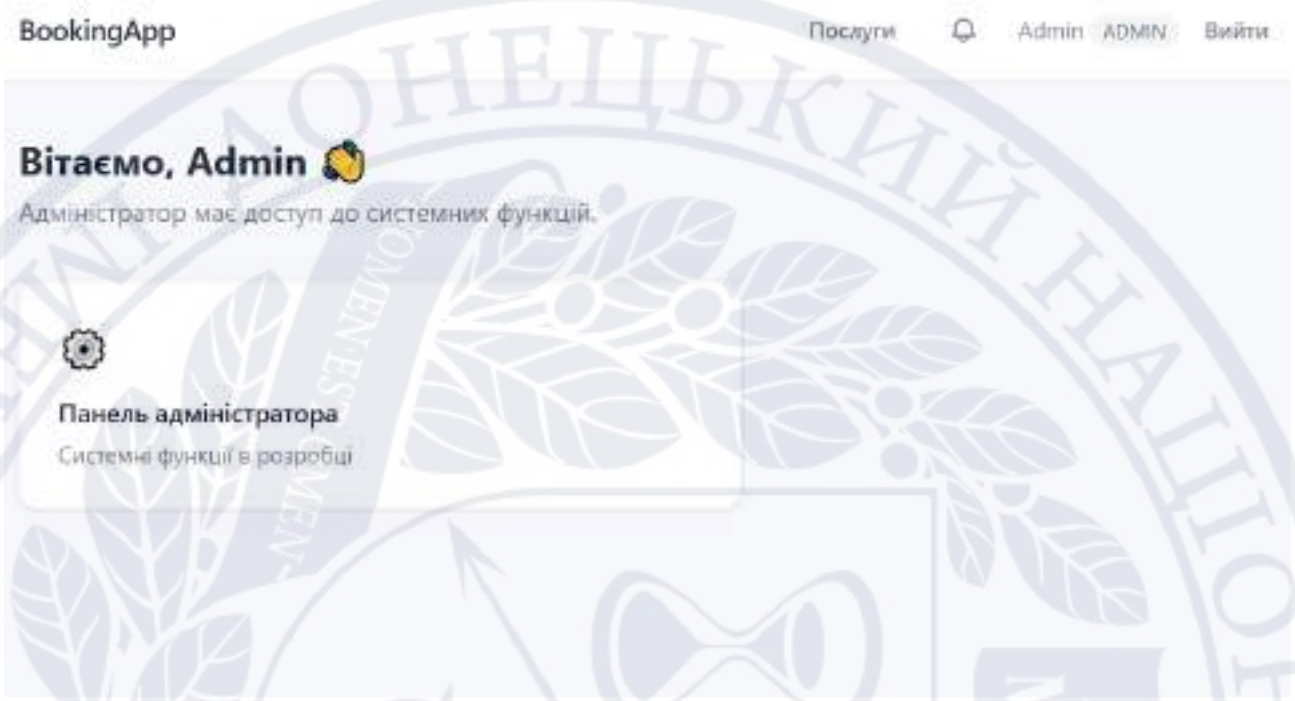


Рис. В.1 – Головна сторінка адміністратора системи