МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інженерії

програмного забезпечення

_____ Євген ДАВИДЕНКО

«___» _____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

ВЕБЗАСТОСУНОК ПОШУКУ КОВОРКІНГІВ

Спеціальність 121 Інженерія програмного забезпечення

Освітня програма «Інженерія програмного забезпечення»

Здобувач _____ **Михайло КІЩУК**

«__» _____ 2026 р.

Керівник роботи

PhD,

доцентка _____ **Катерина АНТІПОВА**

«__» _____ 2026 р.

Миколаїв – 2026

Завдання на виконання кваліфікаційної роботи

Чорноморський національний університет імені Петра Могили

Факультет	Комп'ютерних наук
Кафедра	Інженерії програмного забезпечення
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступінь	Бакалавр
Спеціальність	121 Інженерія програмного забезпечення
Освітня програма	Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри інженерії
програмного забезпечення
_____ Євген ДАВИДЕНКО
«___» _____ 2025 р.

ЗАВДАННЯ

на кваліфікаційну бакалаврську роботу здобувача

Кіщука Михайла

1. Тема кваліфікаційної роботи Вебзастосунок пошуку коворкінгів затверджена наказом ректора ЧНУ ім. Петра Могили №349 від «26» грудня 2025 р.
2. Строк представлення кваліфікаційної роботи «___» _____ 2026 р.
3. Очікуваним результатом роботи є вебзастосунок для пошуку та бронювання коворкінгів, який забезпечує клієнтам зручний інструмент для пошуку, фільтрації та бронювання робочих просторів, а власникам коворкінгів – модуль управління своїми просторами та бронюваннями з розмежуванням доступу за ролями користувачів.
4. Перелік питань, що підлягають розробці: аналіз предметної галузі та огляд існуючих аналогів систем управління коворкінгами; обґрунтування вибору

технологічного стеку та програмних засобів; проєктування клієнт-серверної архітектури системи та розробка структури реляційної бази даних; реалізація серверної частини з REST API, механізмом JWT-авторизації та рольовою моделлю доступу; розробка клієнтської частини з модулями пошуку, фільтрації, інтерактивної карти, бронювання та системи відгуків і рейтингів; реалізація адміністративного модуля модерації коворкінгів та управління користувачами; проведення тестування розроблених модулів та оцінка ефективності готової системи.

5. Перелік графічних матеріалів: Презентація

6. Консультанти:

Консультант	**Кафедра (організація)**	**Частина роботи**

Дата видачі завдання « 27 » грудня 2025 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: **Вебзастосунок пошуку ковокрінгів**

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження технічного завдання	27.12.2025	27.12.2025	*Виконано*
2.	Огляд літератури та аналіз аналогів	28.12.2025	02.02.2026	*Виконано*
3.	Складання календарного плану КБР	03.02.2026	04.02.2026	*Виконано*
4.	Визначення ролей, сценаріїв, вимог	05.02.2026	18.02.2026	*Виконано*
5.	Проєктування БД та вибір стеку	19.02.2026	10.03.2026	*Виконано*
6.	Розробка бекенду	11.03.2026	28.03.2026	*Виконано*
7.	Розробка фронтенду	29.03.2026	19.04.2026	*Виконано*
8.	Отримання відгуку керівника КБР	20.04.2026	21.04.2026	*Виконано*
9.	Оформлення КБР та презентації	22.04.2026	26.05.2026	*Виконано*
10.	Попередній захист	27.05.2026	27.05.2026	*Виконано*
11.	Завершення оформлення КБР та презентації	12.06.2026	12.06.2026	*Виконано*
12.	Рецензування	13.06.2026	18.06.2026	*Виконано*
13.	Захист кваліфікаційної роботи			

Здобувач _____ **Михайло КІЩУК**

«__» _____ 2026 р.

Керівник роботи

PhD,

доцентка _____ **Катерина АНТІПОВА**

«__» _____ 2026 р.

АНОТАЦІЯ

до кваліфікаційної бакалаврської роботи

«Вебзастосунок пошуку коворкінгів»

Здобувач 409 гр.: Кіщук Михайло

Керівник: PhD, доцентка Антіпова Катерина

Актуальність роботи зумовлена стрімким зростанням попиту на гнучкі формати організації робочого простору серед віддалених працівників і фрілансерів, а також відсутністю зручних вітчизняних україномовних платформ для агрегації, пошуку та бронювання коворкінгів.

Метою роботи є проєктування та розробка вебзастосунку з клієнт-серверною архітектурою для спрощення взаємодії між клієнтами, власниками коворкінг-просторів та адміністраторами платформи.

Об’єктом роботи є процес інформаційної взаємодії між орендодавцями коворкінг-просторів та потенційними клієнтами в умовах цифровізації ринку комерційної нерухомості та дистанційної зайнятості.

Предметом роботи є програмні засоби проєктування і розробки вебзастосунків на основі клієнт-серверної архітектури з REST API та рольовою моделлю доступу.

Кваліфікаційна робота складається із вступу, 4 розділів, висновків та переліку джерел посилання.

У вступі обґрунтовано актуальність обраної теми, визначено мету, основні завдання, об’єкт і предмет роботи, а також розкрито практичне значення одержаних результатів.

У першому розділі проведено аналіз предметної галузі та розглянуто існуючі програмні рішення для пошуку та управління.

У другому розділі обрано й обґрунтовано вибір технологічного стеку, сформовано специфікацію вимог до програмного забезпечення.

У третьому розділі описано архітектуру системи з використанням UML-моделювання , програмну реалізацію клієнтської та серверної частин, включаючи

систему сповіщень та аналітику для власників , а також наведено детальний опис інтерфейсу користувача.

У четвертому розділі наведено специфікацію основних класів і методів , описано процес та результати модульного й API-тестування за допомогою інструментів Vitest і Swagger , а також розроблено детальне керівництво користувача зі сценаріями роботи для кожної ролі.

У висновках узагальнено результати виконаної роботи, підтверджено виконання поставлених завдань та визначено шляхи подальшого розширення функціоналу створеного вебзастосунку.

Кваліфікаційна робота викладена на 69 сторінках машинописного тексту, складається із вступу, 4 розділів, загальних висновків, переліку джерел посилання з 18 найменувань та 2 додатки. Праця містить 14 таблиць та 28 рисунків.

Ключові слова: *вебзастосунок, коворкінг, система бронювання, ASP.NET Core, React, MySQL, RESTful API, клієнт-серверна архітектура, авторизація, рейтингова система.*

ABSTRACT

to the bachelor's qualification work

"Web application for finding coworking spaces"

Student of group 409: Kishchuk Mykhailo

Supervisor: PhD, Associate Professor Antipova Kateryna

The relevance of the work is due to the rapid growth of demand for flexible formats for organizing workspaces among remote workers and freelancers, as well as the lack of convenient domestic Ukrainian-language platforms for aggregation, search and booking of coworking spaces.

The purpose of the work is to design and develop a web application with client-server architecture to simplify interaction between clients, coworking space owners and platform administrators.

The object of the work is the process of information interaction between coworking space landlords and potential clients in the context of digitalization of the commercial real estate market and remote employment.

The subject of the work is software tools for designing and developing web applications based on client-server architecture with REST API and role-based access model.

The qualification work consists of an introduction, 4 chapters, conclusions, and a list of references.

The introduction substantiates the relevance of the selected topic, defines the goal, main tasks, object and subject of the work, and also reveals the practical significance of the results obtained.

The first section analyzes the subject area and considers existing software solutions for search and management.

The second section selects and justifies the choice of the technology stack, and formulates the software requirements specification.

The third section describes the system architecture using UML modeling, the software implementation of the client and server parts, including the notification system and analytics for owners, and a detailed description of the user interface.

The fourth section provides a specification of the main classes and methods, describes the process and results of unit and API testing using the Vitest and Swagger tools, and also develops a detailed user manual with work scenarios for each role.

The conclusions summarize the results of the work performed, confirm the fulfillment of the tasks set, and identify ways to further expand the functionality of the created web application.

The qualification work is presented on 69 pages of typewritten text, consists of an introduction, 4 chapters, general conclusions, a list of references with 18 titles and 2 appendices. The work contains 14 tables and 28 figures.

Keywords: *ASP.NET Core, authorization, booking system, client-server architecture, coworking, MySQL, rating system, React, RESTful API, web application.*

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БД – база даних

СКБД – система керування базами даних

КБР – кваліфікаційна бакалаврська робота

ПЗ – програмне забезпечення

API – інтерфейс програмування застосунків (Application Programming Interface)

JSON – об’єктна нотація JavaScript (JavaScript Object Notation)

JWT – вебтокен JSON (JSON Web Token)

ORM – об’єктно-реляційне відображення (Object-Relational Mapping)

REST – передача стану представлення (Representational State Transfer)

SaaS – програмне забезпечення як послуга (Software as a Service)

SPA – односторінковий застосунок (Single-Page Application)

ВСТУП

Сучасний ринок праці демонструє стрімке зростання кількості фрілансерів, віддалених працівників і малих підприємців, які потребують якісного робочого простору поза домом. За даними галузевих досліджень, ринок коворкінгів продовжує активно розвиватись, однак процес пошуку підходящого простору залишається розрізненим і незручним: інформація розміщена на різних платформах, відсутня єдина система порівняння, фільтрації та бронювання. Особливо гостро ця проблема стоїть в Україні, де відсутні україномовні рішення подібного класу, адаптовані до потреб вітчизняного ринку.

Розробка вебзастосунку, що об’єднує можливості каталогу, пошуку з фільтрацією, інтерактивної карти, системи бронювання та відгуків в єдиній україномовній платформі, є актуальною науково-практичною задачею.

Мета роботи – проєктування та розробка вебзастосунку пошуку і бронювання коворкінгів для спрощення взаємодії між клієнтами, власниками коворкінг-просторів і адміністраторами платформи.

Для досягнення визначеної мети необхідно вирішити завдання:

– провести аналіз предметної області та існуючих аналогів систем пошуку і управління коворкінгами;

– визначити функціональні вимоги до системи та ролі користувачів;

– обрати технічний стек та архітектурний підхід;

– спроєктувати структуру бази даних;

– обрати інструменти розробки та архітектурний підхід;

– реалізувати механізм автентифікації та авторизації користувачів на основі JWT-токенів з підтримкою рольової моделі;

– розробити модуль пошуку та фільтрації коворкінгів за критеріями;

– реалізувати модуль бронювання з автоматичним розрахунком вартості та перевіркою конфліктів;

– розробити систему рейтингів і відгуків з автоматичним перерахунком середнього балу коворкінгу;

– реалізувати адміністративний модуль модерації нових коворкінгів та управління користувачами;

– виконати тестування роботи системи.

Об’єкт роботи – процес інформаційної взаємодії між орендодавцями коворкінг-просторів та потенційними клієнтами в умовах цифровізації ринку комерційної нерухомості та дистанційної зайнятості.

Предмет роботи – програмні засоби проєктування і розробки вебзастосунків на основі клієнт-серверної архітектури з REST API та рольовою моделлю доступу.

Обґрунтування необхідності розробки

Існуючі рішення у сфері управління та пошуку коворкінгів переважно орієнтовані на великі мережі просторів та управління внутрішніми бізнес-процесами. Дослідження сучасних вебплатформ показало, що більшість із них мають такі недоліки:

– відсутність української мови інтерфейсу;

– висока вартість підписки, що є бар’єром для малих операторів коворкінгів;

– відсутність геолокації коворкінгів на інтерактивній карті;

– відсутня вбудована прозора система відгуків і рейтингів для клієнтів.

Аналіз тенденцій ринку свідчить про стрімке зростання попиту на гібридні формати роботи та гнучкі робочі простори. Світовими тенденціями у вирішенні поставлених завдань є перехід до хмарних SaaS-рішень, автоматизація процесів бронювання, впровадження смарт-доступу та створення орієнтованих на спільноту маркетплейсів.

Сфера застосування результатів

Розроблений вебзастосунок може бути використаний у:

– єдиних національних або регіональних агрегаторах та маркетплейсах для пошуку та порівняння гнучких робочих просторів;

– діяльності власників малого та середнього бізнесу для управління бронюваннями, номерним фондом та перегляду статистики;

– повсякденній діяльності цільової аудиторії кінцевих споживачів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Успішна розробка програмного продукту вимагає глибокого розуміння середовища, в якому він буде функціонувати. Ринок комерційної нерухомості та організації робочих просторів перебуває у стані активної цифровізації, де ключову роль відіграють спеціалізовані платформи, що об’єднують орендодавців та клієнтів. Детальне вивчення цього простору, його структурних особливостей та наявних програмних аналогів дозволяє чітко окреслити проблематику та сформувати обґрунтований базис для створення нової, більш ефективної системи пошуку і бронювання.

1.1 Соціально-економічні передумови та тенденції розвитку ринку коворкінгів

Сучасна трансформація глобального ринку праці характеризується відходом від традиційних офісних моделей на користь гнучкості та мобільності. Стрімкий розвиток цифрових технологій дозволив значній частині фахівців виконувати професійні обов’язки незалежно від стаціонарного робочого місця. Це призвело до виникнення явища коворкінгу – спільного робочого середовища, яке інтегрує переваги професійної офісної інфраструктури з автономністю дистанційної зайнятості. Коворкінг-простори сьогодні розглядаються не просто як місця для оренди столів, а як складні екосистеми, що сприяють підвищенню продуктивності та стимулюють підприємницьку активність [1].

В українському контексті попит на такі простори зростає під впливом розширення ІТ-сектору, збільшення кількості фрілансерів та необхідності бізнесу адаптуватися до нестабільних умов. Для багатьох підприємців та незалежних контрагентів коворкінг стає інструментом оптимізації витрат, оскільки дозволяє уникати довгострокових зобов’язань за договорами оренди нерухомості. Проте, попри кількісне зростання пропозицій на ринку, потенційні користувачі стикаються з проблемою неефективного пошуку та порівняння доступних варіантів. Процес вибору часто ускладнений розрізненістю даних: інформація про

ціни, наявність вільних місць, технічне оснащення та реальні відгуки розпорошена по соціальних мережах або індивідуальних сайтах окремих операторів.

Розвиток економіки спільного споживання диктує нові вимоги до сервісів, що обслуговують цю галузь. Коворкінги стають ключовим трендом не лише для самозайнятих осіб, а й для великих корпорацій, що впроваджують гібридні формати роботи [2].

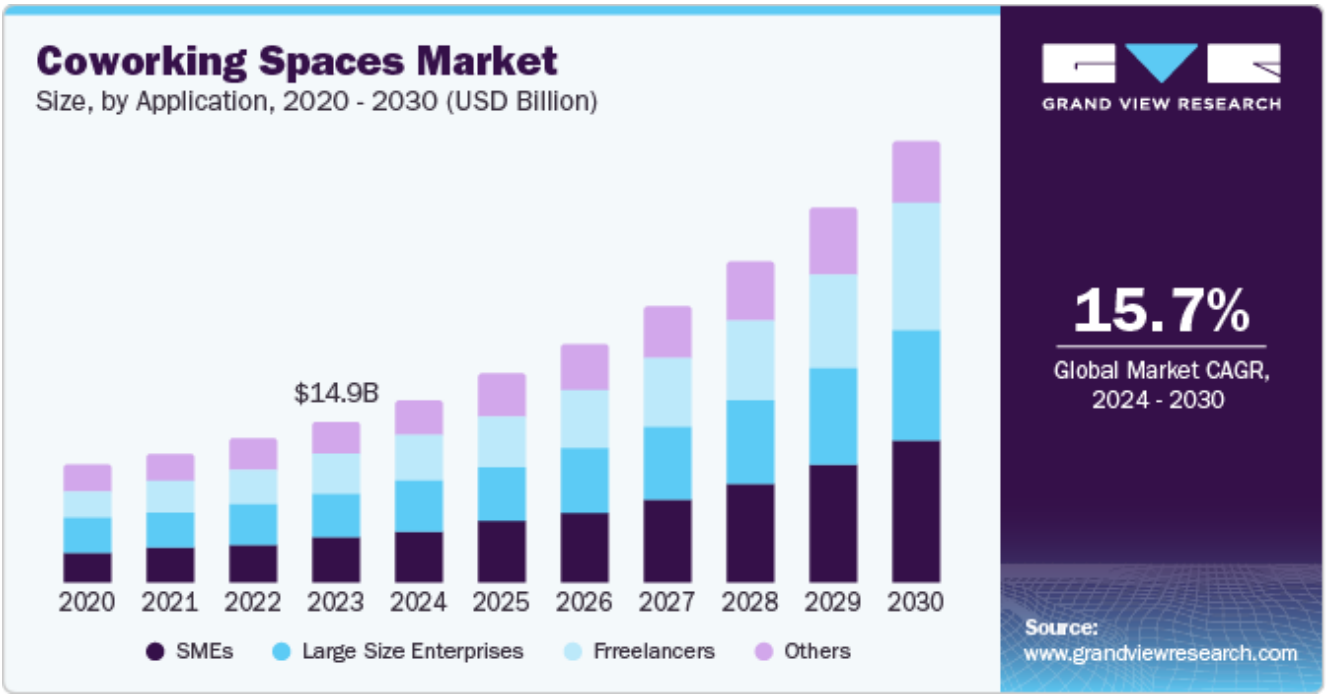

Рисунок 1.1 – Динаміка та прогноз обсягу глобального ринку коворкінгів за категоріями користувачів (2020–2030 рр.), млрд дол. США

Світові аналітичні дослідження підтверджують стрімке масштабування цієї галузі. Згідно з даними компанії Grand View Research [3], очікується, що сукупний середньорічний темп зростання глобального ринку коворкінгів у період з 2024 по 2030 рік становитиме 15,7% (рис. 1.1). Така позитивна динаміка свідчить про глибоку структурну перебудову підходів до організації робочого процесу в усьому світі.

Варто зазначити, що структура попиту на ці послуги є диверсифікованою: ринок чітко сегментується між малим та середнім бізнесом, великими підприємствами, а також фрілансерами. Ця статистика наочно ілюструє описану раніше тенденцію та підтверджує, що сучасні коворкінги здатні задовольняти

інфраструктурні потреби як індивідуальних спеціалістів, так і масштабних корпоративних команд. Відповідно, виникає потреба в автоматизації процесів взаємодії між постачальниками послуг та споживачами. Створення єдиного цифрового середовища для моніторингу ринку коворкінгів дозволило б нівелювати інформаційну асиметрію, спростити процедуру бронювання та забезпечити прозорість ціноутворення.

Технологічна реалізація такої платформи потребує застосування сучасних засобів вебпрограмування, здатних забезпечити високу швидкість обробки даних та зручність користувацького інтерфейсу. Використання компонентно-орієнтованих бібліотек для фронтенд-частини та надійних серверних рішень дозволяє побудувати масштабовану систему, що відповідатиме специфічним вимогам локального ринку та запитам сучасної цифрової аудиторії. Таким чином, детальне вивчення механізмів функціонування коворкінгів та методів їх цифровізації є необхідним кроком для розробки ефективного інструментарію управління гнучкими робочими просторами.

1.2 Аналіз предметної галузі та структурно-функціональні особливості об’єкта дослідження

Коворкінг як явище виник на початку 2000-х років і спочатку був поширений переважно серед технологічних стартапів та фрілансерів у великих містах. Перший офіційно задокументований коворкінг було відкрито у Сан-Франциско в 2005 році. Надалі модель стала глобальним трендом, охопивши як мегаполіси, так і невеликі міста [2]. Дослідники визначають коворкінг як форму організації праці, що поєднує фізичний простір, спільноту та набір послуг, пропонуючи альтернативу домашньому офісу та традиційній оренді [4].

З точки зору інформаційних систем, ринок коворкінгів є двостороннім: з одного боку – власники просторів, які прагнуть ефективно заповнити місця та отримати дохід; з іншого – клієнти, що шукають доступне, зручне та добре обладнане робоче місце. Відповідно, платформа-посередник має задовольняти потреби обох сторін, забезпечуючи прозорість, зручність та надійність транзакцій.

Структурно об’єкт дослідження включає такі ключові сутності:

– коворкінг – основна одиниця каталогу, містить назву, адресу, опис, перелік зручностей, ціну за годину, кількість місць, фото та геокоординати для відображення на карті;

– організація – юридична або фізична особа, що є власником одного або кількох коворкінгів і несе відповідальність за достовірність розміщеної інформації;

– користувач – суб’єкт системи з визначеною роллю: гість, клієнт, власник або адміністратор;

– бронювання – факт резервування конкретного місця в коворкінгу на визначений проміжок часу з розрахованою вартістю;

– відгук – оцінка від 1 до 5 зірок та текстовий коментар, що клієнт може залишити після підтвердженого бронювання.

Функціональна модель системи передбачає чотири рівні доступу. Гість може переглядати каталог, читати відгуки та використовувати пошук і фільтри – без необхідності реєстрації. Клієнт після авторизації отримує можливість бронювати місця, керувати власними бронюваннями, залишати відгуки та зберігати улюблені простори. Власник коворкінгу може додавати та редагувати свої простори, підтверджувати або скасовувати бронювання клієнтів, а також переглядати статистику доходів. Адміністратор відповідає за модерацію нових коворкінгів і забезпечення цілісності даних платформи.

Важливою особливістю предметної галузі є необхідність перевірки доступності місць у реальному часі. Оскільки кілька клієнтів можуть намагатись забронювати той самий час одночасно, система повинна мати механізм перевірки конфліктів бронювань на рівні бази даних. Крім того, у коворкінгу може бути кілька місць, тому перевірка має враховувати кількість паралельних бронювань відносно загальної місткості приміщення [5].

Геолокаційна складова є невід’ємною частиною сучасних платформ пошуку нерухомості та послуг. Для вебзастосунку пошуку коворкінгів відображення об’єктів на інтерактивній карті суттєво покращує досвід користувача, дозволяючи обрати простір з урахуванням зручності розташування відносно будинку, офісу або

транспортних вузлів [6]. Бібліотека Leaflet у поєднанні з даними OpenStreetMap надає відкрите та безкоштовне рішення для реалізації цього функціоналу [7].

Система відгуків і рейтингів є критично важливим інструментом довіри на двосторонніх платформах. Можливість залишати відгук лише після підтвердженого бронювання запобігає маніпуляціям і підвищує достовірність оцінок, що позитивно впливає як на вибір клієнтів, так і на мотивацію власників підтримувати якість послуг [1].

Ефективне функціонування такої двосторонньої платформи вимагає тісної взаємодії усіх визначених структурних сутностей. Зокрема, сутність «бронювання» виступає ключовою транзакційною ланкою між «користувачем» зі статусом клієнта та конкретним «коворкінгом», який належить відповідній «організації». Саме під час оформлення цієї транзакції система повинна задіювати описаний вище механізм перевірки конфліктів, що гарантує відповідність кількості паралельних резервувань реальній фізичній місткості приміщення.

Зі свого боку, накопичення транзакційних даних формує базис для роботи підсистеми оцінювання. Оскільки сутність «відгук» жорстко прив’язана до факту підтвердженого бронювання, платформа акумулює виключно верифікований користувацький досвід. У комплексі з геолокаційним пошуком це створює прозоре інформаційне середовище, яке повністю нівелює наявну на ринку проблему розрізненості даних та забезпечує користувачів необхідними інструментами для порівняння просторів.

1.3 Огляд та аналіз сучасного стану програмних рішень у предметній галузі

Ринок програмного забезпечення для управління коворкінгами налічує десятки рішень, однак переважна більшість з них розрахована на управління власним простором, а не на агрегацію та пошук з боку клієнта. Аналіз та порівняння трьох поширених і функціонально близьких до предмета розробки систем наведено у таблицях 1.1–1.7.

Таблиця 1.1 – Характеристика та аналіз системи OfficeRnD

Назва	OfficeRnD
Виробник	OfficeRnD Ltd.
Архітектура	3-tier web application, Cloud-based SaaS
Мова реалізації	TypeScript, React, Node.js, PostgreSQL
Основні функції	Автоматизація управління членами, бронюваннями та простором. Виставлення рахунків та повторювані платежі. Аналітика зайнятості та доходів. Інтеграція з платіжними системами та системами контролю доступу. CRM для управління клієнтами.
Переваги	Гнучка кастомізація під потреби оператора; масштабованість для мульти-локацій; сильні інтеграції з зовнішніми сервісами; високий рівень безпеки (SOC 2, ISO 27001); присвячений онбординг для нових клієнтів.
Недоліки	Висока вартість підписки для малих операторів; потребує значного часу на навчання; повна залежність від інтернет-з’єднання; відсутня українська мова інтерфейсу; не є платформою-агрегатором для кінцевих клієнтів.
Джерело інформації	https://www.officernd.com/

OfficeRnD (рис. 1.2) є одним із лідерів ринку B2B-рішень для операторів коворкінгів. Система добре масштабується для великих мереж, проте для малого оператора її вартість та складність можуть бути невиправданими. З точки зору потреб кінцевого клієнта – людини, яка шукає де попрацювати, система не надає публічного каталогу з пошуком і фільтрацією.

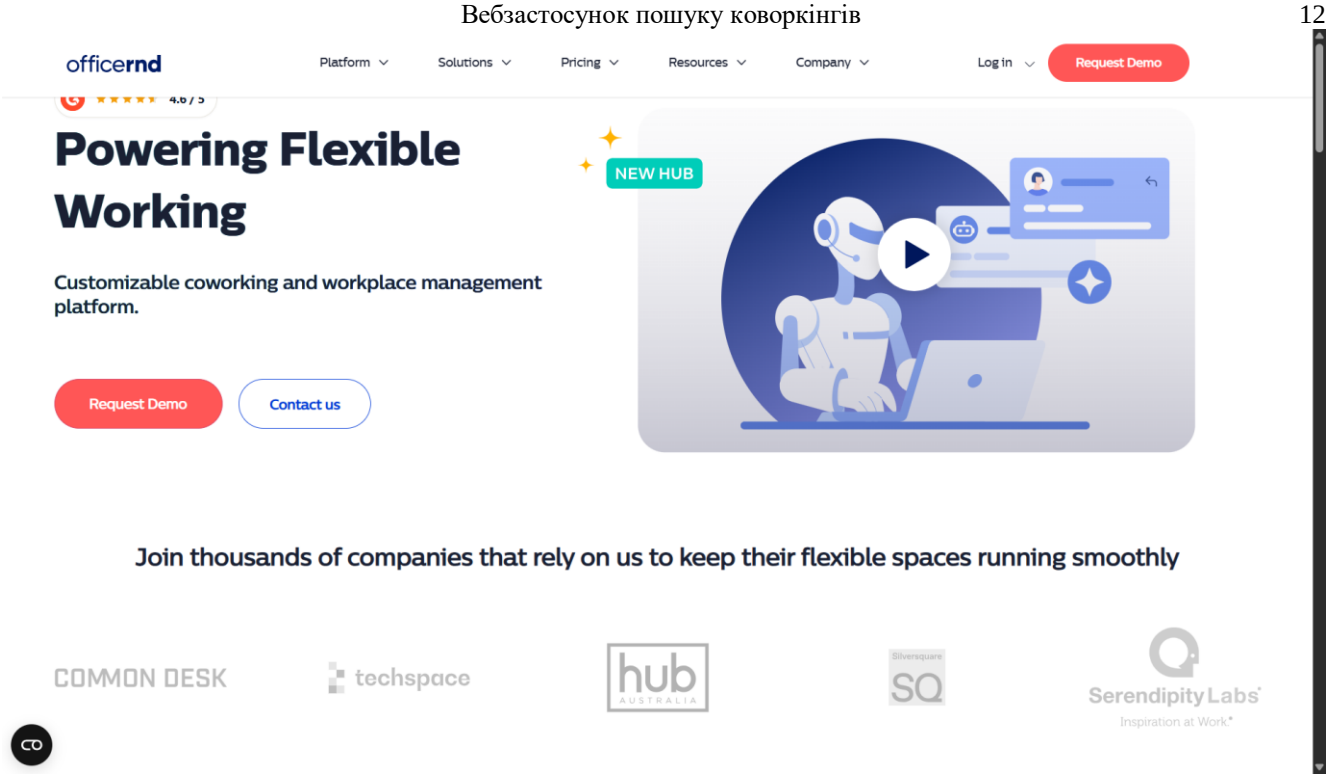

Рисунок 1.2 – Вебсайт компанії OfficeRnD

З метою поглибленого аналізу диференціальних характеристик системи OfficeRnD та проєкту, що перебуває на стадії розроблення, здійснено їх компаративний аналіз за ключовими критеріями, які мають визначальне значення для кінцевого користувача в Україні (табл. 1.2).

Таблиця 1.2 – Порівняння системи OfficeRnD та проєктованого вебзастосунку

Критерій	**OfficeRnD**	**Проєктований вебзастосунок**
Цільова аудиторія	Великі оператори	Клієнти та власники
Мова інтерфейсу	Англійська	Українська
Публічний каталог	Немає	Є
Пошук за картою	Немає	Є
Система відгуків	Обмежена	Є

Виходячи з порівняння, незважаючи на потужний функціонал OfficeRnD для бізнесу, проєктований вебзастосунок виграє в аспектах доступності для звичайного

користувача, наявності україномовного інтерфейсу та інструментів агрегованого пошуку.

Таблиця 1.3 – Характеристика та аналіз системи Nexudus

Назва	Nexudus
Виробник	Nexudus Ltd.
Архітектура	3-tier web application, Cloud-based SaaS
Мова реалізації	JavaScript, HTML/CSS, Bootstrap
Основні функції	Автоматизоване бронювання з регулярними платежами. Розумне бронювання та контроль доступу до просторів. Автоматизація робочих процесів. Централізоване адміністрування для мульти-локацій. Понад 60 інтеграцій із зовнішніми інструментами.
Переваги	Висока масштабованість для мереж коворкінгів; потужна автоматизація бізнес-процесів; якісна підтримка клієнтів; суттєва економія часу на операційних задачах.
Недоліки	Обмежена кастомізація без використання API; виключно англійська мова інтерфейсу; можливі додаткові витрати на інтеграції; відсутня вбудована система відгуків і рейтингів для кінцевих клієнтів.
Джерело інформації	https://www.nexudus.com/

Nexudus (рис. 1.3) є зрілим продуктом з широкою екосистемою інтеграцій, проте його архітектура орієнтована виключно на оператора коворкінгу. Відсутність системи рейтингів та публічного профілю простору унеможливлює використання Nexudus як платформи-агрегатора, що є принциповою відмінністю від вимог до проєктованого застосунку.

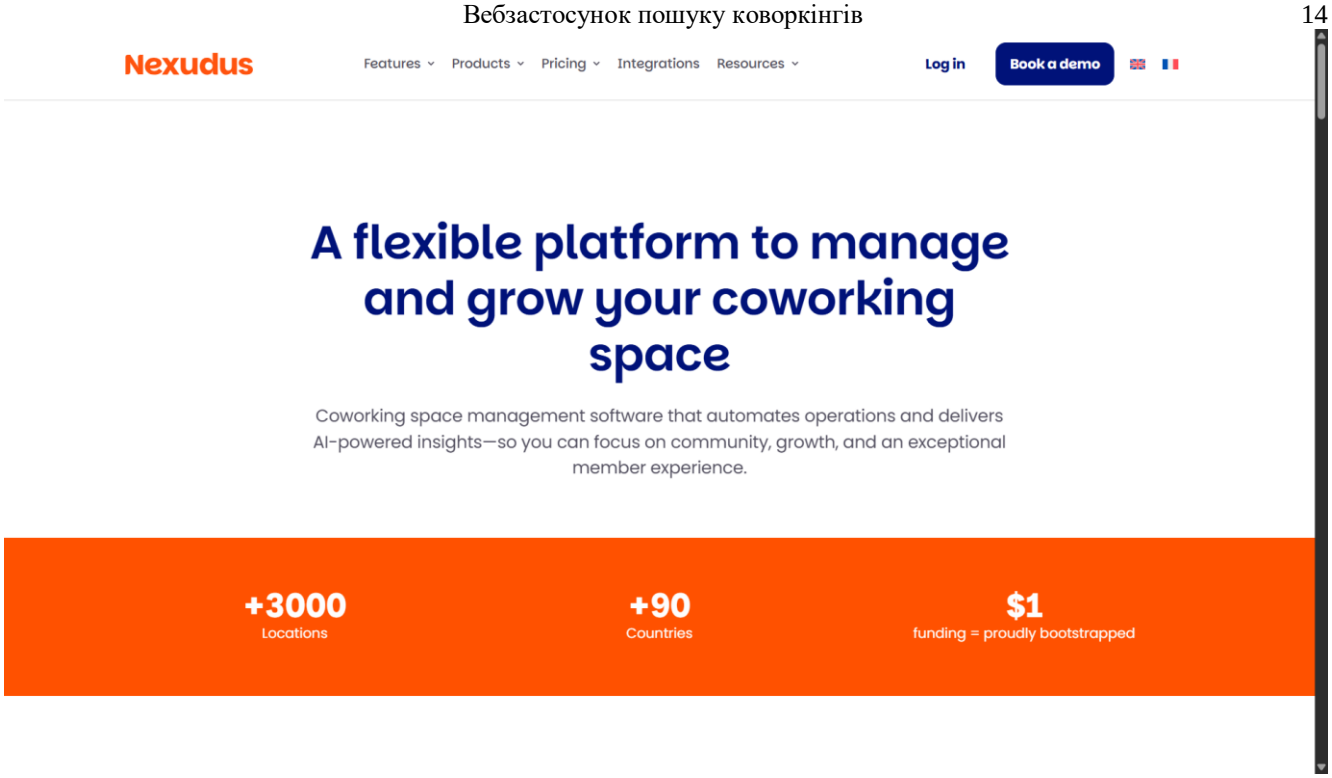

Рисунок 1.3 – Вебсайт компанії Nexudus

З метою поглибленого аналізу диференціальних характеристик системи Nexudus та проєкту, що перебуває на стадії розроблення, здійснено їх компаративний аналіз за ключовими критеріями, які мають визначальне значення для кінцевого користувача в Україні (табл. 1.4).

Таблиця 1.4 – Порівняння системи Nexudus та проєктованого вебзастосунку

Критерій	**Nexudus**	**Проєктований вебзастосунок**
Фокус системи	Автоматизація оператора	Маркетплейс коворкінгів
Перегляд без реєстрації	Немає	Є
Система рейтингів простору	Відсутня	Є
Порівняння об’єктів	Немає	До 3 просторів
Безкоштовний доступ (клієнт)	Немає	Є

Виходячи з порівняння, Nexudus є вузькоспеціалізованим інструментом адміністрування, тоді як проєктований вебзастосунок пропонує

клієнтоорієнтований підхід із відкритою системою відгуків та зручним порівнянням об’єктів без обов’язкової реєстрації.

Таблиця 1.5 – Характеристика та аналіз системи Coworks

Назва	Coworks
Виробник	Coworks, Inc.
Архітектура	3-tier web application, Cloud-based SaaS, мобільний додаток
Мова реалізації	JavaScript, React, Node.js
Основні функції	Інтуїтивне управління простором. Мобільний додаток для кінцевих користувачів. Бронювання day pass та конференц-залів. Портал для членів спільноти. Базова аналітика зайнятості.
Переваги	Простота та легкість використання; орієнтованість на спільноту; гнучка масштабованість; висока цінність за ціну; економія часу для операторів.
Недоліки	Обмежені розширені функції для великих мереж; фокус на простоті може обмежувати кастомізацію; відсутня геолокація коворкінгів на карті; немає системи рейтингів для клієнтів; відсутня підтримка української мови.
Джерело інформації	https://www.coworks.com/

Coworks (рис. 1.4) наближається до концепції клієнтоорієнтованої платформи завдяки мобільному додатку, однак також залишається переважно інструментом управління для оператора. Відсутність картографічного відображення просторів та системи незалежних рейтингів суттєво обмежує її корисність для клієнта, що обирає між кількома пропозиціями.

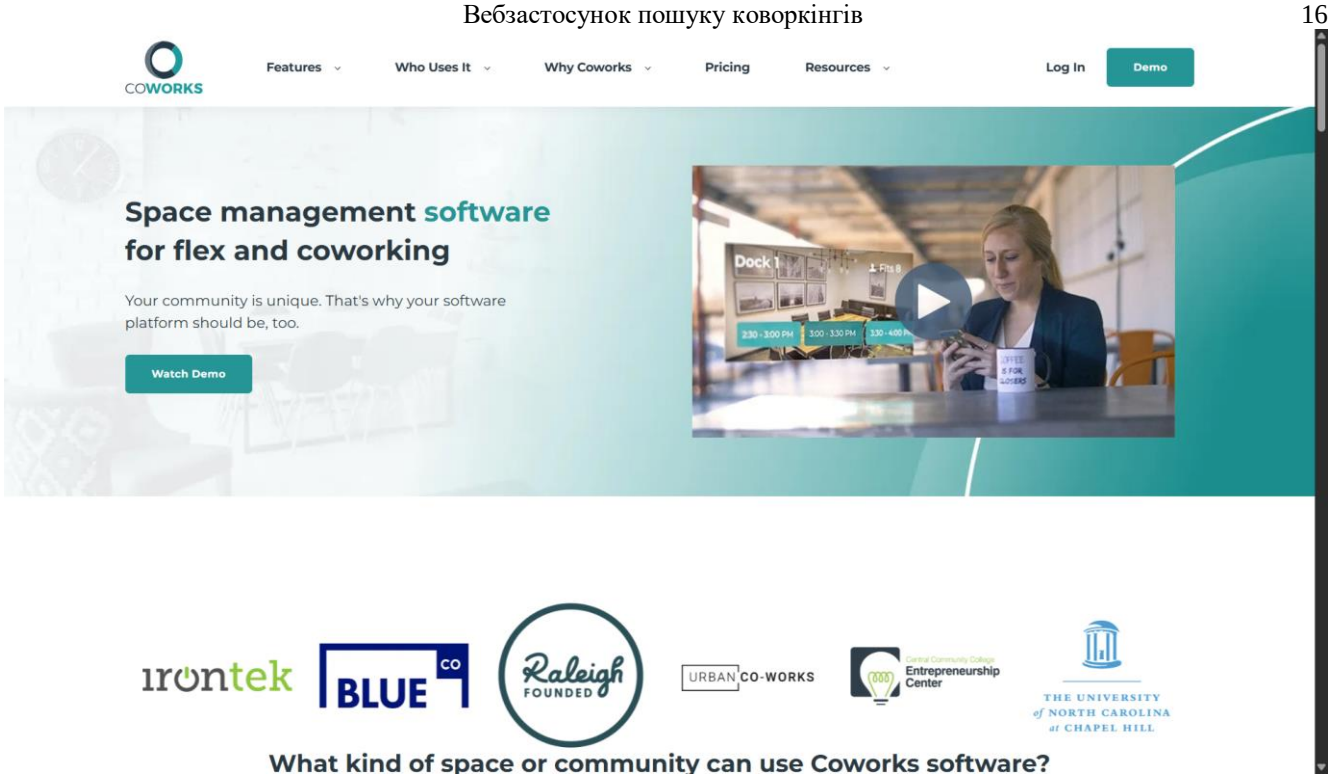

Рисунок 1.4 – Вебсайт компанії Coworks

З метою поглибленого аналізу диференціальних характеристик системи Coworks та проєкту, що перебуває на стадії розроблення, здійснено їх компаративний аналіз за ключовими критеріями, які мають визначальне значення для кінцевого користувача в Україні (табл. 1.6).

Таблиця 1.6 – Порівняння системи Coworks та проєктованого вебзастосунку

Критерій	**Coworks**	**Проєктований вебзастосунок**
Пошук за геолокацією	Немає	Є
Система відгуків	Відсутня	Є
Локалізація інтерфейсу	Англійська	Українська
QR-код верифікації	Немає	Є
Збереження в "Улюблені"	Обмежено	Є

Хоча Coworks пропонує мобільний додаток і зручний інтерфейс для ком’юніті, він не вирішує проблему пошуку простору за географічним розташуванням і не має української локалізації. Проєктований вебзастосунок

компенсує ці недоліки, надаючи повноцінний маркетплейс із картою та прозорою системою рейтингів.

Проведений покроковий аналіз підтверджує, що існуючі на ринку популярні рішення орієнтовані насамперед на внутрішні потреби операторів коворкінгів, залишаючи поза увагою зручність для кінцевого українського користувача та інструменти відкритої агрегації просторів.

З метою узагальнення виявлених характеристик аналогів та комплексного зіставлення їх із функціоналом проєктованого рішення сформовано загальну порівняльну таблицю 1.7.

Таблиця 1.7 – Порівняльний аналіз програмних рішень у предметній галузі

Критерій	**OfficeRnD**	**Nexudus**	**Coworks**	**Проєктований вебзастосунок**
Мова інтерфейсу	Англійська	Англійська	Англійська	Українська
Пошук за картою	Немає	Немає	Немає	Є
Система відгуків і рейтингів	Обмежена	Відсутня	Відсутня	Є
Порівняння коворкінгів	Немає	Немає	Немає	До 3 просторів
Перегляд без реєстрації	Немає	Немає	Немає	Є
Аналітика для власника	Є	Є	Обмежена	Є
Безкоштовний доступ	Немає	Немає	Немає	Є
Сповіщення in-app	Є	Є	Є	Є
QR-код бронювання	Немає	Немає	Немає	Є

Аналіз таблиці 1.7 виявляє ряд системних прогалин у наявних рішеннях. Жодне з розглянутих рішень не надає публічного україномовного каталогу коворкінгів. Відсутні картографічне відображення та незалежна система рейтингів є спільним недоліком усіх аналогів. Можливість перегляду та порівняння просторів без реєстрації є очікуваною функцією сучасних маркетплейсів, однак жоден із аналогів її не надає. Саме ці прогалини є обґрунтуванням для розробки нового рішення – вебзастосунку пошуку коворкінгів [1, 2, 4].

Висновки до розділу 1

У результаті виконання першого розділу проведено комплексний аналіз предметної області, який підтвердив стрімку трансформацію глобального та вітчизняного ринків праці у бік гнучких офісних моделей. Виявлено, що зростання попиту на коворкінги супроводжується проблемою інформаційної асиметрії, оскільки дані про доступні локації залишаються розрізненими та неструктурованими. Визначено ключові сутності системи, а також необхідність реалізації механізмів перевірки конфліктів бронювань у реальному часі, інтерактивної карти та верифікованих відгуків. Особливу увагу приділено геолокаційній складовій та системі верифікованих відгуків як фундаментальним інструментам побудови довіри між користувачами та власниками просторів. Шляхом порівняльного аналізу існуючих програмних рішень, таких як OfficeRnD, Nexudus та Coworks, встановлено, що ринок перенасичений інструментами для адміністрування бізнесу, проте практично не має зручних клієнтських агрегаторів з українською локалізацією та відкритим каталогом. Таким чином, результати аналізу повністю обґрунтовують доцільність розробки нового вебзастосунку, який би поєднував функції маркетплейса, системи бронювання та інтерактивної карти, задовольняючи специфічні потреби українського сегмента ринку коворкінгів.

2 МОДЕЛЮВАННЯ ОБ'ЄКТУ ТА ПРЕДМЕТУ РОБОТИ

Перехід від теоретичного дослідження до практичної реалізації вимагає чіткого структурування майбутньої інформаційної системи та визначення її технологічного базису. Побудова надійної клієнт-серверної архітектури безпосередньо залежить від виваженого вибору сучасних інструментів розробки, здатних забезпечити високу продуктивність, масштабованість та безпеку застосунку. Створення архітектурних моделей та формування вичерпної специфікації вимог є необхідним фундаментом, який передує написанню програмного коду та гарантує відповідність готового продукту поставленим завданням.

2.1 Аналіз сучасного стану інструментарію, моделей та методів

Розробка сучасного вебзастосунку для пошуку та бронювання коворкінгів потребує застосування широкого спектру технологій, інструментів і методів. Для обґрунтованого вибору технологічного стеку та архітектурних рішень необхідно провести аналіз актуального стану інструментарію у таких напрямках: фронтенд-розробка, бекенд-розробка, управління базами даних, геоінформаційні технології та забезпечення безпеки вебзастосунків

2.1.1 Технології фронтенд-розробки

Сучасна фронтенд-розробка базується на компонентно-орієнтованих підходах, що забезпечують повторне використання коду та полегшують підтримку складних інтерфейсів. Серед найпоширеніших фреймворків виділяють React, Angular та Vue.js.

Дослідження Emmanni P. S., опубліковане у «International Journal of Science and Research», проводить порівняльний аналіз Angular, React та Vue.js у контексті розробки SPA-застосунків. Автор встановлює, що React демонструє найвищу гнучкість у великих командах, відзначається розвиненою екосистемою та найширшою підтримкою спільноти, а також надає можливість вибудовувати

архітектуру на власний розсуд [8]. При цьому Angular пропонує більш жорстку та передбачувану структуру проєкту, що може бути перевагою для ентерпрайз-застосунків, але ускладнює вхід для нових розробників.

Świątkowski A. та Ścibior K. у «Journal of Computer Sciences Institute» порівнюють React, Next.js та Gatsby для створення SPA-застосунків. Дослідники доходять висновку, що React займає провідні позиції завдяки Virtual DOM, що мінімізує реальні операції з DOM і підвищує продуктивність, та однонаправленому потоку даних, що спрощує відлагодження [9]. Dubaj S. та Pańczyk B. у тому самому виданні порівнюють React та Svelte, підкреслюючи, що React забезпечує стабільність та довгострокову підтримку за рахунок потужної корпоративної підтримки з боку Meta [10].

Важливим аспектом є використання TypeScript разом із React. Rippon C. у навчальному посібнику «Learn React with TypeScript» зазначає, що статична типізація TypeScript суттєво знижує кількість помилок на етапі розробки, покращує автодоповнення в IDE та полегшує рефакторинг у великих кодових базах [11]. Для управління глобальним станом застосунку серед сучасних рішень виділяється Zustand – легковажна бібліотека, що не потребує шаблонного коду на відміну від Redux

2.1.2 Технології бекенд-розробки

Для серверної частини вебзастосунків широко застосовується платформа ASP.NET Core. Valiveti S. S. S. у роботі «Evolution of ASP.NET to ASP.NET Core», опублікованій IEEE, аналізує еволюцію платформи та встановлює, що ASP.NET Core 9 є кросплатформовим, модульним і продуктивним рішенням, яке суттєво перевершує попередні версії за швидкодією та зручністю розгортання завдяки вбудованому контейнеру IoC та middleware-архітектурі [12].

Malavasi A. у книзі «Modern Full-Stack Web Development with ASP.NET Core» детально розкриває підхід до побудови RESTful API на ASP.NET Core 9, зокрема реалізацію системи ролей через JWT Bearer та Entity Framework Core для ORM-взаємодії з базою даних [1, 13]. Автор підкреслює, що контролери в ASP.NET Core

дозволяють чітко розмежовувати відповідальність між компонентами, дотримуючись принципу Single Responsibility.

Proud N. у монографії «Minimal APIs in ASP.NET 9» досліджує підхід мінімальних API як альтернативу контролерам для нескладних сервісів, проте для застосунків зі складною бізнес-логікою, кількома рівнями авторизації та розгалуженою маршрутизацією рекомендує класичний підхід на основі контролерів [14]. Classon I. у «Migrating ASP.NET Microservices to ASP.NET Core 8» підкреслює переваги ASP.NET Core у контексті безпеки та продуктивності [15].

2.1.3 Технології управління базами даних

Вибір СКБД є критичним для продуктивності та масштабованості вебзастосунків. Shethiya A. S. у «Journal of Selected Topics in Academic Research» досліджує стратегії балансування навантаження та шардингу в SQL Server для великомасштабних вебзастосунків, доводячи, що реляційні СКБД зберігають конкурентоспроможність навіть при значних обсягах даних завдяки ефективній індексації та транзакційній цілісності [16].

Ilić M. та співавтори проводять порівняльний аналіз продуктивності Microsoft SQL Server та Oracle, встановлюючи, що MySQL є оптимальним вибором для вебзастосунків середнього масштабу завдяки відкритому коду, широкій документації та зрілій екосистемі інструментів [17]. Do T.-T.-T. та співавтори порівнюють графові та реляційні бази даних, підкреслюючи, що реляційні СКБД залишаються незамінними для систем із чіткою структурою зв’язків між сутностями, яка є характерною для застосунку бронювання коворкінгів [5].

2.1.4 Геоінформаційні технології

Відображення об’єктів на інтерактивній карті є невід’ємною частиною сучасних маркетплейсів нерухомості та послуг. Ahmad M. та співавтори у «Pioneering Approaches in Data Management» аналізують застосування даних OpenStreetMap для геомаркетингових аналітичних завдань, підкреслюючи, що OSM надає якісні картографічні дані для більшості регіонів світу безкоштовно та

без ліцензійних обмежень [6]. Ahmad M. у «Data-Driven Intelligent Business Sustainability» додатково розкриває, що поєднання Leaflet.js з даними OSM дозволяє реалізувати повноцінний картографічний функціонал без залежності від платних API, таких як Google Maps [7].

2.1.5 Предметна галузь коворкінгів

Для розуміння вимог до застосунку необхідно врахувати специфіку галузі. Kraus S. та співавтори у «Journal of Innovation & Knowledge» здійснюють систематичний огляд досліджень у сфері коворкінгів та мейкерспейсів, констатуючи стрімке зростання кількості таких просторів у світі та підкреслюючи важливість цифрових платформ для їх просування [2]. Berbegal-Mirabent J. у журналі «Sustainability» аналізує тренди у розвитку коворкінгів та виявляє, що сучасні клієнти очікують від платформ прозорості цін, незалежних відгуків та зручного онлайн-бронювання [1]. Girija S. та співавтори у «Property Management» досліджують фактори, що впливають на намір користуватися коворкінгами на ринках, що розвиваються, підтверджуючи, що функціональність мобільного та вебінтерфейсу є одним із ключових чинників вибору платформи [4].

2.2 Моделювання предметної області

Вибір технологічного стеку є стратегічним рішенням, що визначає довгостроковий потенціал масштабування системи, зручність її розробки та підтримки. На основі аналізу сучасного стану інструментарію, проведеного у підрозділі 2.1, та функціональних вимог, сформульованих у підрозділі 1.3, для проєкту обрано технологічний стек, що охоплює три основні шари: фронтенд, бекенд та базу даних. Нижче обґрунтовується вибір кожного компонента.

2.2.1 Фронтенд-шар

Основу фронтенд-шару складає бібліотека React 18 у поєднанні зі статичною типізацією TypeScript. React побудовано на компонентній моделі, що дозволяє розбивати складний інтерфейс на ізольовані, повторно використовувані блоки.

Концепція Virtual DOM забезпечує мінімальну кількість реальних операцій з деревом документа, що підвищує продуктивність рендерингу навіть при динамічних змінах стану [9]. Однонаправлений потік даних спрощує налагодження та тестування компонентів. TypeScript додає статичну типізацію, яка дозволяє відловлювати помилки ще на етапі написання коду, суттєво прискорює розробку завдяки автодоповненню в IDE та полегшує рефакторинг у великих кодових базах [11].

Для збірки та локального середовища розробки обрано Vite. На відміну від застарілих рішень на кшталт Create React App, Vite використовує нативні ES-модулі браузера для Hot Module Replacement (HMR), що забезпечує миттєве відображення змін без повного перезавантаження сторінки. Це суттєво прискорює ітеративну розробку складних інтерфейсів [11].

Для стилізації компонентів застосовано TailwindCSS 4 з підходом utility-first. На відміну від традиційних CSS-фреймворків із готовими компонентами, Tailwind надає набір атомарних класів, що дозволяє будувати довільні дизайни безпосередньо в JSX-розмітці, не перемикаючись між файлами. Це прискорює розробку та унеможливлює конфлікти між стилями різних компонентів [8].

Для управління глобальним станом застосунку (авторизаційний токен, тема інтерфейсу, список порівнюваних коворкінгів) обрано Zustand. Ця бібліотека мінімалістична і не потребує шаблонного коду (boilerplate), характерного для Redux: стан визначається у вигляді звичайного JavaScript-об’єкта з методами-мутаторами. Для взаємодії з API використовується Axios, що надає зручний інтерфейс для налаштування базової URL, інтерцепторів запитів та обробки помилок авторизації. Додаткові бібліотеки фронтенду та їх призначення наведено в таблиці 2.1.

Таблиця 2.1 – Бібліотеки фронтенду та їх призначення

Бібліотека	**Призначення**
React	Компонентна UI-бібліотека
TypeScript	Статична типізація JavaScript

Кінець таблиці 2.1

Бібліотека	Призначення
Vite	Збірка проєкту та HMR
TailwindCSS	Utility-first стилізація компонентів
Zustand	Глобальний стан застосунку
Axios	HTTP-клієнт для взаємодії з API
React Router DOM	Клієнтська навігація між сторінками
React Leaflet	Інтерактивна карта з маркерами
Recharts	Графіки статистики
React Hook Form	Керування формами та валідація
React DatePicker	Вибір дати та часу при бронюванні
QRCode.react	Генерація QR-кодів для бронювань
date-fns	Маніпуляції з датами та форматування
lucide-react	Набір SVG-іконок для UI

Таким чином, комплексне застосування обраного стеку фронтенд-технологій та допоміжних бібліотек дозволяє побудувати високопродуктивний, масштабований та стійкий до навантажень клієнтський застосунок. Це забезпечує реалізацію інтерактивного інтерфейсу користувача, що відповідає сучасним вимогам до систем пошуку та бронювання коворкінгів, а також гарантує швидку та безперебійну взаємодію клієнтської частини з сервером.

2.2.2 Бекенд-шар

Для серверної частини обрано платформу ASP.NET Core 9 та мову програмування C#. ASP.NET Core є кросплатформовим фреймворком, орієнтованим на продуктивність та модульність. Порівняно з попередніми версіями, ASP.NET Core 9 демонструє суттєво вищу пропускну здатність завдяки вбудованому пайплайну обробки запитів на основі Middleware-архітектури [12]. Кожен запит проходить через ланцюжок middleware-компонентів: авторизація,

обробка CORS, rate limiting, обробка виключень, що забезпечує розмежування відповідальності та гнучкість конфігурації.

Принциповим для проєкту є вбудований контейнер впровадження залежностей. Завдяки DI-контейнеру сервіси реєструються при запуску застосунку і надаються компонентам за необхідності. Це забезпечує слабку зв'язність між компонентами, спрощує їх тестування та підтримку.

Для взаємодії з базою даних застосовано Entity Framework Core 9 як ORM. EF Core генерує параметризовані SQL-запити, що автоматично захищає від SQL-ін'єкцій. Механізм міграцій дозволяє версіонувати схему бази даних та відтворювати її стан у будь-якому середовищі розгортання командою `dotnet ef database update`. ValueConverter EF Core використовується для прозорого шифрування та дешифрування персональних даних при зчитуванні та записі, не вимагаючи змін у коді бізнес-логіки. Бекенд-залежності та їх призначення наведено в таблиці 2.2.

Таблиця 2.2 – Пакети бекенду та їх призначення

Пакет NuGet	**Призначення**
Microsoft.EntityFrameworkCore	ORM для роботи з базою даних
Pomelo.EntityFrameworkCore.MySql	Провайдер EF Core для MySQL Server 8
Microsoft.EntityFrameworkCore.Tools	Інструменти для генерації міграцій
Microsoft.AspNetCore.Authentication.JwtBearer	Middleware для перевірки JWT-токенів
BCrypt.Net-Next	Хешування паролів алгоритмом BCrypt
Swashbuckle.AspNetCore	Генерація Swagger UI для документації API
AspNetCoreRateLimit	Обмеження частоти запитів з однієї IP
System.Text.Json (вбудований)	Серіалізація/десеріалізація JSON

Використання зазначених пакетів розширення у поєднанні з базовими архітектурними перевагами ASP.NET Core формує надійний та захищений бекенд-шар вебзастосунку. Такий набір серверних інструментів повністю задовольняє потреби у безпечній автентифікації користувачів, ефективній трансляції запитів до бази даних та контролі навантаження, закладаючи міцний фундамент для стабільної роботи платформи.

2.2.3 Рівень бази даних

Для зберігання даних обрано реляційну СКБД MySQL Server 8.0. Реляційна модель є природною для предметної галузі системи бронювання коворкінгів: сутності пов’язані чіткими відношеннями «один до багатьох» та «один до одного», які ефективно моделюються зовнішніми ключами реляційних таблиць [5]. Транзакційна підтримка гарантує цілісність даних при одночасному оформленні бронювань кількома клієнтами, що є критичною вимогою для систем резервування.

MySQL Server 8 є зрілою СКБД з відкритим вихідним кодом, широкою документацією та активною спільнотою. Він добре поєднується з Entity Framework Core через провайдер Pomelo, підтримує JSON-поля для зберігання довільних даних, а також забезпечує ефективну індексацію для пришвидшення пошукових запитів за містом, рейтингом та ціною [17].

2.2.4 Взаємодія технологій

Взаємодія трьох шарів здійснюється через REST API за протоколом HTTPS. Фронтенд надсилає запити, які бекенд валідує через JWT Bearer middleware. Після виконання бізнес-логіки Entity Framework Core транслює операції в SQL-запити до MySQL Server. Результат повертається клієнту у форматі JSON із camelCase-нотацією. Обґрунтування обраної архітектури спирається на сучасні наукові дослідження, зокрема на ґрунтовний аналіз особливостей формування front-end технологій у хмарних середовищах на основі клієнт-серверної архітектури, проведений С. Пасічником та Н. Кунанець [18]. У своїй статті автори запропонували узагальнену часову модель латентності користувацького запиту, де

загальна затримка розкладається на час мережевої передачі, час серверного опрацювання, час доступу до бази даних та витрати на серіалізацію і рендеринг у браузері.

Дослідники доводять, що для інформаційних систем, які опрацьовують значні масиви даних та вимагають швидкого відгуку інтерфейсу, критичним є зменшення сумарної затримки на кожному з цих етапів. Згідно з їхніми висновками, використання архітектурного стилю REST залишається високоефективним для стабільних ресурсних сервісів, управління якими складає основу систем бронювання. Застосування REST API дозволяє ефективно використовувати механізми умовного кешування, що суттєво зменшує навантаження на мережевий рівень та бекенд.

Крім того, у статті наголошується на важливості оптимізації фази рендерингу та UX-оптимізації на стороні клієнта за рахунок компонентних підходів і часткового відмальовування інтерфейсу. Таким чином, запропонований у роботі підхід до побудови клієнт-серверної архітектури, де клієнтський застосунок взаємодіє з серверною частиною через RESTful API із застосуванням кешування та оптимізованих запитів до реляційної бази даних, повністю відповідає сучасним науковим рекомендаціям щодо мінімізації параметрів часової моделі латентності та забезпечення цільових показників рівня сервісу.

2.3 Специфікація вимог до програмного забезпечення

1) ***Призначення та межі проєкту:***

1.1) призначення системи: проєктований вебзастосунок призначений для автоматизації пошуку, порівняння та бронювання коворкінг-просторів в Україні. Система обслуговує дві групи користувачів одночасно: клієнтів, яким надається зручний україномовний інструмент для підбору робочого місця за параметрами та безпечного бронювання в обраному часовому інтервалі, та власників коворкінгів, яким надається особистий кабінет для управління об’єктами, підтвердження бронювань та аналізу статистики;

1.2) погодження, що ухвалені в програмній документації: специфікацію складено відповідно до ISO/IEC/IEEE 29148:2025; ухвалено технологічний стек React 18 + TypeScript + Vite, ASP.NET Core 9, MySQL Server 8. Передбачено підтримку виключно української мови інтерфейсу, адаптивний дизайн та підтримку світлої та темної теми. Персональні дані шифруються засобами AES-256 через EF Core ValueConverter;

1.3) межі проєкту ПЗ: проєкт охоплює розробку вебклієнта та серверної інфраструктури для бронювання місць і керування коворкінгами. Не охоплює інтеграцію з реальними платіжними системами, нативний мобільний додаток, систему управління персоналом коворкінгів, офлайн-режим роботи.

2) **Загальний опис:**

2.1) сфера застосування: платформа обслуговує B2C-ринок оренди коворкінг-просторів України, поєднуючи власників просторів та клієнтів – фрілансерів, підприємців і представників малого бізнесу, що потребують гнучкого робочого простору;

2.2) характеристики користувачів: гість – неавторизований відвідувач, очікує можливості переглядати каталог, читати відгуки та шукати коворкінги без реєстрації; клієнт – зареєстрований користувач, шукає робоче місце, очікує зручного пошуку, бронювання з підтвердженням, відгуків та особистого кабінету; власник – підприємець або організація, що надає коворкінг в оренду, має середній технічний рівень, очікує панелі управління об’єктами, бронюваннями та статистикою; адміністратор – системний адміністратор платформи, має високий технічний рівень, відповідає за модерацію, управління користувачами та журнал аудиту;

2.3) загальна структура і склад системи: клієнтська частина – React 18 + TypeScript + Vite, серверна частина – ASP.NET Core 9 RESTful API, база даних – MySQL Server 8;

2.4) загальні обмеження: система функціонує виключно за наявності стабільного HTTPS-з’єднання зі швидкістю не менше 1 Мбіт/с; підтримувані браузери: Chrome 90+, Firefox 88+, Edge 90+, Safari 14+; бронювання

доступне лише у межах робочих годин: 08:00–23:00; один клієнт не може мати два паралельних перетинаючихся бронювання в одному коворкінгу; власник на базовому тарифному плані може розмістити не більше 2 активних коворкінгів.

3) **Функції системи:**

3.1) Пошук та перегляд коворкінгів:

3.1.1) опис функції: дозволяє здійснювати пошук коворкінгів за назвою або містом, застосовувати фільтри, сортувати результати, переглядати об’єкти на інтерактивній карті з маркерами та переглядати детальну сторінку обраного коворкінгу;

3.1.2) вхідна і вихідна інформація: вхідна – рядок пошуку, параметри фільтрів, параметр сортування; вихідна список коворкінгів із назвою, містом, фото, рейтингом, ціною за годину та кількістю місць;

3.1.3) функціональні вимоги: пошук здійснюється за частковим збігом у назві та місті; результати завантажуються порціями через нескінченний скрол; час відповіді на пошуковий запит – не більше 2 с.

3.2) Бронювання місця:

3.2.1) опис функції: дозволяє авторизованому клієнту обрати коворкінг, задати дату початку та час завершення бронювання. Система перевіряє наявність вільних місць, відсутність конфліктів бронювань та належність часового інтервалу до допустимого діапазону. Час завершення вибирається для того самого дня, що і початок;

3.2.2) вхідна і вихідна інформація: вхідна – ідентифікатор коворкінгу, datetime початку та datetime завершення; вихідна – об’єкт бронювання зі статусом «очікує підтвердження», розрахована вартість, сповіщення клієнту та власнику;

3.2.3) функціональні вимоги: перевірка умови на наявність вільних місць; заборона дублювання бронювань одного клієнта в тому самому часовому інтервалі; блокування кнопки «Забронювати» при недоступності.

3.3) Управління відгуками:

3.3.1) опис функції: дозволяє клієнту, що має підтверджене бронювання у коворкінгу, залишити один відгук із числовою оцінкою 1–5 та текстовим коментарем. Клієнт може редагувати або видаляти свій відгук. Рейтинг коворкінгу автоматично перераховується після кожної зміни;

3.3.2) вхідна і вихідна інформація: вхідна – числова оцінка, текстовий коментар, ідентифікатор коворкінгу; вихідна – збережений відгук із ім’ям автора та датою, оновлений середній рейтинг коворкінгу;

3.3.3) функціональні вимоги: один відгук від клієнта для кожного коворкінгу; відгук доступний лише після підтвердженого бронювання; автор може відредагувати або видалити власний відгук, адміністратор – видаляти будь-який.

3.4) Управління організацією та коворкінгами:

3.4.1) опис функції: дозволяє власнику створити організацію з довільними контактними даними у вигляді JSON-рядка, додавати та редагувати коворкінги, переглядати й підтверджувати бронювання клієнтів, а також скасовувати їх;

3.4.2) вхідна і вихідна інформація: вхідна – назва, адреса, опис, URL логотипу організації, JSON-об’єкт із контактами, параметри коворкінгу; вихідна – збережений об’єкт зі статусом «на модерації»;

3.4.3) функціональні вимоги: реєстрація організації є обов’язковою умовою перед додаванням коворкінгів; базовий тарифний план обмежує власника 2 активними коворкінгами; новий коворкінг публікується у каталозі лише після затвердження адміністратором.

3.5) Адміністрування платформи:

3.5.1) опис функції: дозволяє адміністратору затверджувати або відхиляти нові коворкінги після їх додавання власником, переглядати

всі бронювання платформи з можливістю підтвердження та скасування, управляти обліковими записами користувачів, переглядати журнал аудиту всіх дій у системі;

3.5.2) вхідна і вихідна інформація: вхідна – параметри пошуку в журналі, рішення модератора; вихідна – відфільтрований список записів журналу, оновлений статус коворкінгу або бронювання;

3.5.3) функціональні вимоги: кожна дія (вхід, реєстрація, бронювання, підтвердження, скасування) записується в таблицю AuditLogs часової мітки та email користувача; журнал доступний виключно адміністратору та підтримує пошук і пагінацію.

4) **Вимоги до інформаційного забезпечення:**

4.1) джерела і зміст вхідної інформації (даних): вхідна інформація надходить з форм введення даних користувачами (реєстрація, авторизація, бронювання, відгуки, дані організацій та коворкінгів); параметрів URL-рядка запиту (фільтри, сортування, пагінація); геокоординат, що вводяться власниками у формах додавання коворкінгів;

4.2) нормативно-довідкова інформація (класифікатори, довідники тощо): використовуються внутрішні довідники що містять перелік типів зручностей коворкінгу, ролі користувачів, статуси бронювань та тарифні плани організацій;

4.3) вимоги до способів організації, збереження та ведення інформації: Дані зберігаються у реляційній СКБД MySQL Server 8.0. Персональні дані користувачів шифруються алгоритмом AES-256 до запису та розшифровуються автоматично при читанні. Контактна інформація організацій зберігається у вигляді JSON-рядка у полі ContactInfo, що забезпечує гнучкість без модифікації схеми.

5) **Вимоги до технічного забезпечення:** мережа: швидке з’єднання з підтримкою HTTPS; клієнтські пристрої: сучасні браузери на ПК або смартфонах.

6) **Вимоги до програмного забезпечення:**

6.1) архітектура програмної системи: триланкова клієнт-серверна архітектура: рівень представлення, рівень бізнес-логіки та рівень даних. Взаємодія між рівнями відбувається виключно через HTTPS;

6.2) системне програмне забезпечення: Windows Server 2022 або Ubuntu 24.04 LTS і новіші, Nginx.

6.3) мережне програмне забезпечення: TLS 1.3, підтримка HTTP/2;

6.4) програмне забезпечення ведення інформаційної бази: MySQL Server 8.0, Entity Framework Core 9 з провайдером Pomelo.EntityFrameworkCore.MySql;

6.5) мова і технологія розробки ПЗ: TypeScript 5, React 18, Vite 5, TailwindCSS 4, Zustand, Axios, React Leaflet, Recharts, React Hook Form. C# 13, ASP.NET Core 9, Entity Framework Core 9, JWT Bearer, BCrypt.Net-Next, AspNetCoreRateLimit, Swashbuckle.

7) **Вимоги до зовнішніх інтерфейсів:**

7.1) інтерфейс користувача: мінімалістичний дизайн із підтримкою світлої та темної колірних тем, адаптивна верстка для екранів шириною 320x480 до 1920x1080 пікселів;

7.2) апаратний інтерфейс: Специфічних вимог не висувається;

7.3) програмний інтерфейс: RESTful API, стандартні HTTP-методи, стандартизовані JSON-відповіді з camelCase-нотацією. Коди HTTP-статусів: 200 OK, 201 Created, 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found, 429 Too Many Requests, 500 Internal Server Error. Документація API автоматично генерується через Swagger UI;

7.4) комунікаційний протокол: HTTPS для всіх запиті.

8) **Властивості програмного забезпечення:**

8.1) доступність: час роботи системи: 99.9%, менше 8 годин простою на рік;

8.2) супроводжуваність: модульна структура коду;

8.3) переносимість: сумісність із популярними хмарними платформами;

8.4) продуктивність: час відповіді на запит не більше 1 секунди при 1000 одночасних користувачів;

8.5) надійність: відновлення після збою не більше 5 хвилин, періодичні автоматичні резервні копіювання;

8.6) безпека: JWT-авторизація з терміном дії токена 1440 хвилин, BCrypt-хешування паролів, AES-256-шифрування персональних полів, захист від SQL-ін'єкцій; rate limiting; CORS-обмеження джерел запитів.

9) **Інші вимоги:** система повинна відповідати вимогам закону України «Про захист персональних даних» щодо збору, обробки та зберігання персональних даних; адміністратор має можливість повного видалення облікового запису та пов'язаних з ним даних на вимогу користувача; журнал аудиту зберігає IP-адреси, часові мітки та email-адреси для всіх критичних дій; система підтримує горизонтальне розширення функціоналу без зміни схеми бази даних.

Висновки до розділу 2

У другому розділі здійснено аналіз сучасного стану інструментарію та моделювання предметної області. Сформовано специфікацію вимог, яка визначає межі проєкту, ролі користувачів і ключові функції вебзастосунку: пошук коворкінгів, бронювання, управління відгуками та адміністрування. Обґрунтовано вибір технологічного стеку для триланкової клієнт-серверної архітектури: React 18, ASP.NET Core та MySQL Server 8. Завдяки перевагам цих компонентів – високій продуктивності Virtual DOM, безпеці middleware-архітектури та транзакційній цілісності СКБД – створено надійне методологічне й технічне підґрунтя для подальшої розробки стабільної платформи

3 ПРОЄКТУВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ

Перехід до практичної реалізації вебзастосунку вимагає детального опрацювання його архітектури та логіки взаємодії програмних компонентів. Формалізація структури системи засобами UML створює надійний інженерний каркас для написання коду, а поетапна інтеграція серверної та клієнтської частин із сучасним користувацьким інтерфейсом гарантує відповідність готового продукту вимогам надійності, безпеки та продуктивності.

3.1 Архітектура системи та UML-моделювання

Проєктування надійної та масштабованої системи вимагає чіткого визначення її архітектури та моделювання взаємодії її складових. У цьому підрозділі представлено триланкову архітектуру системи та набір UML-діаграм, що візуалізують функціональні вимоги, структуру даних, алгоритми роботи та фізичне розгортання застосунку.

3.1.1 Технології фронтенд-розробки

Вебзастосунок побудовано за триланковою клієнт-серверною архітектурою, яка забезпечує чітке розмежування відповідальності між рівнями системи та спрощує масштабування кожного рівня незалежно. Рівень представлення реалізовано як SPA на React 18 + TypeScript, що виконується у браузері клієнта. Рівень бізнес-логіки представлено RESTful API на ASP.NET Core 9, що обробляє запити, виконує авторизацію та взаємодіє з базою даних. Рівень даних забезпечується СКБД MySQL Server 8.0 з доступом через Entity Framework Core 8 [12, 13].

Бекенд організовано за шаблоном «Controller – Service – DbContext». Контролери відповідають виключно за HTTP-взаємодію: приймають запит, десеріалізують тіло, делегують виклик сервісу та формують відповідь із відповідним HTTP-статусом. Сервіси містять усю бізнес-логіку: валідацію бізнес-правил, формування сповіщень, шифрування, розрахунки. AppDbContext через

Entity Framework Core транслює об’єктно-орієнтовані запити у SQL та надає механізм міграцій для версіонування схеми бази даних.

Між клієнтом і сервером функціонує шар Middleware-компонентів, що обробляє наскрізні аспекти. ExceptionMiddleware перехоплює всі необроблені виключення та повертає стандартизовані JSON-відповіді, запобігаючи витоку інформації про внутрішню структуру системи. JWT Bearer Middleware перевіряє підпис токена та заповнює ClaimsPrincipal. IpRateLimiting обмежує частоту запитів для захисту від brute-force та DDoS-атак. CORS Middleware дозволяє запити лише з дозволеного домену фронтенду.

3.1.2 Діаграма варіантів використання

Функціональні вимоги до системи формалізовано у вигляді діаграми варіантів використання у нотації UML 2.5 (рис. 3.1). Система передбачає чотири актори: Гість, Клієнт, Власник та Адміністратор.

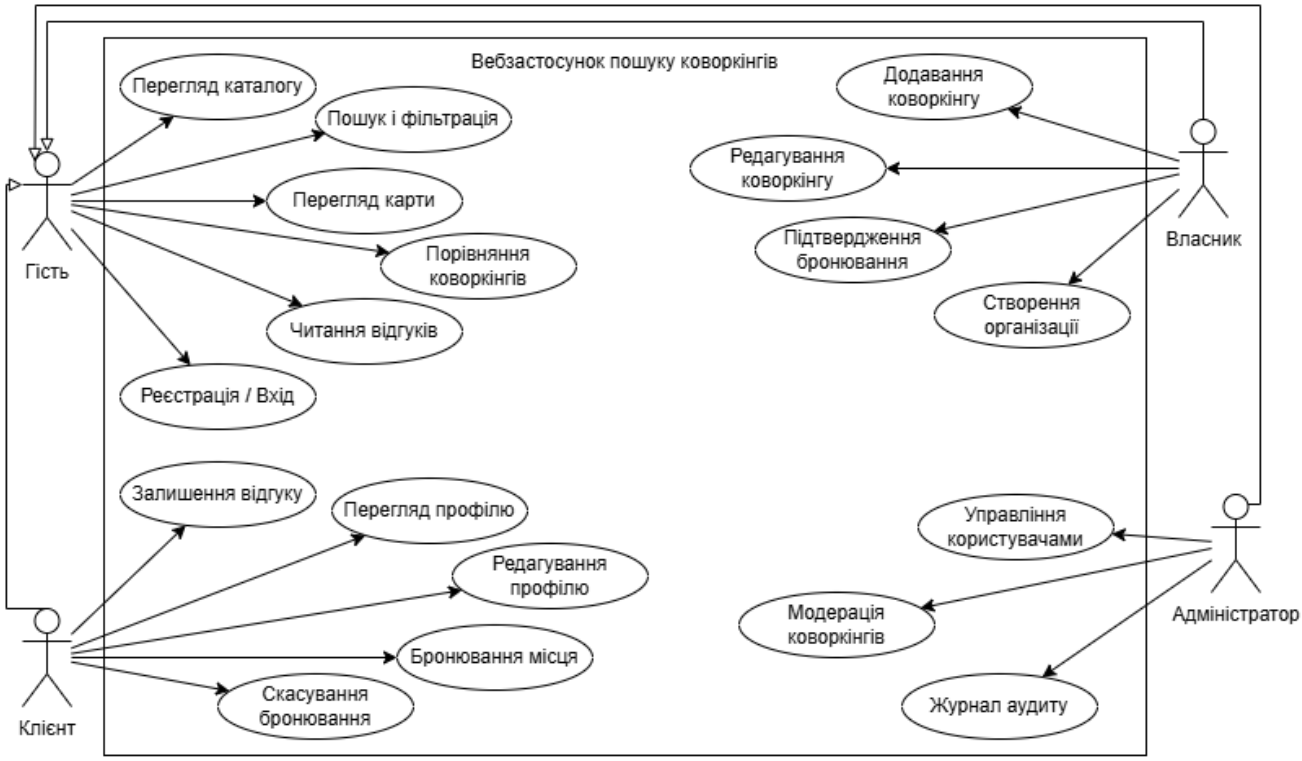

Рисунок 3.1 – Діаграма варіантів використання проєктованого вебзастосунку

Гість взаємодіє з публічним каталогом, пошуком, картою та відгуками без необхідності реєстрації. Клієнт після авторизації отримує доступ до бронювань,

відгуків, збережених просторів та порівняння. Власник управляє своїми коворкінгами, підтверджує бронювання клієнтів та переглядає аналітику (за наявності преміум-плану). Адміністратор здійснює модерацію нових коворкінгів, управляє обліковими записами та переглядає журнал аудиту всіх дій.

3.1.3 Діаграма класів

Статична структура системи описана діаграмою класів у нотації UML 2.5 (рис. 3.2). Модель даних включає сім основних класів: User, Organization, Coworking, Booking, Review, Notification та AuditLog.

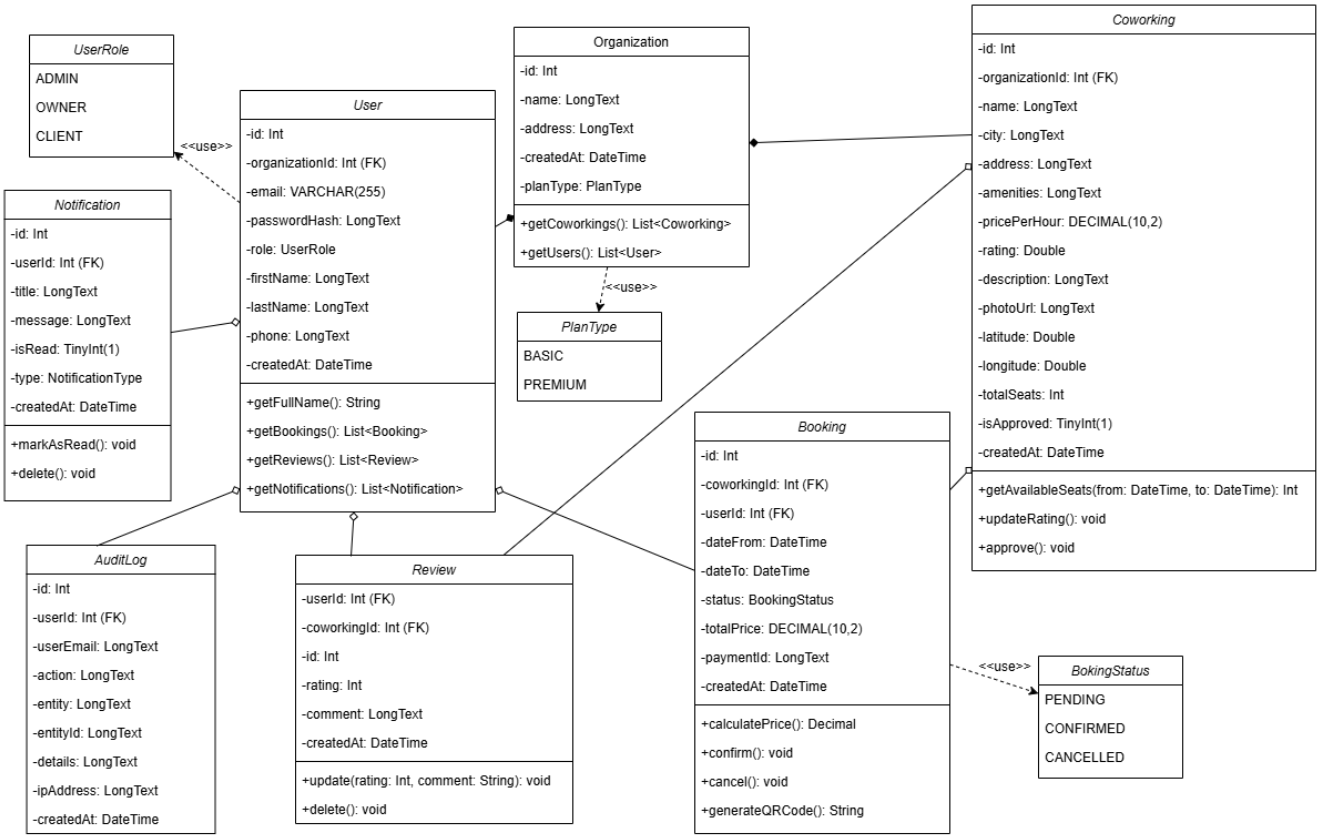

Рисунок 3.2 – Діаграма класів проєктованого вебзастосунку

Клас User містить зашифровані поля персональних даних та атрибут Role, що визначає права доступу. Клас Organization агрегує контактну інформацію у вигляді JSON-рядка у полі ContactInfo, що надає гнучкість без зміни схеми бази даних. Клас Coworking містить геокоординати для відображення на карті та поле Amenities у форматі CSV-рядка. Клас Booking відображає часовий інтервал та статус резервування. Клас AuditLog фіксує всі критичні операції для аудиту безпеки.

3.1.4 Діаграма послідовності (бронювання)

Ключовим бізнес-процесом системи є бронювання місця. Діаграма послідовності (рис. 3.3) ілюструє взаємодію між акторами та компонентами системи під час цього процесу.

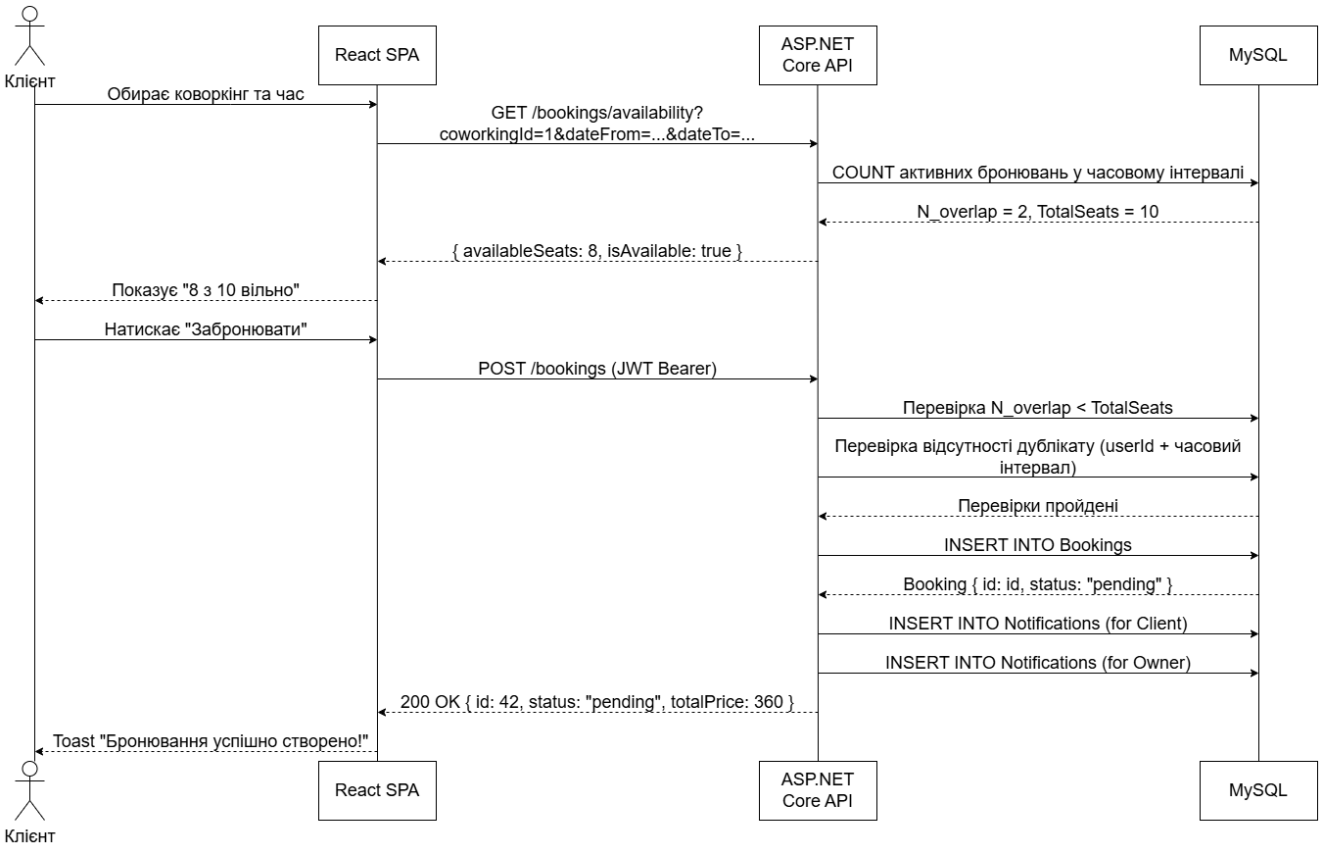

Рисунок 3.3 – Діаграма послідовності «процес бронювання»

Клієнт обирає коворкінг та часовий інтервал, після чого система виконує попередню перевірку доступності за ендпоінтом GET /bookings/availability. Сервер підраховує кількість активних бронювань, що перетинаються з заданим інтервалом, порівнює з TotalSeats та повертає кількість вільних місць. При підтвердженні бронювання виконується транзакційна перевірка двох умов: N_overlap < TotalSeats та відсутність дублікату від того самого клієнта. Після успішного запису сервер надсилає сповіщення як клієнту, так і власнику коворкінгу.

3.1.5 Діаграма діяльності (бронювання)

Процес оформлення бронювання деталізовано за допомогою діаграми діяльності UML (рис. 3.4). Алгоритм починається з вибору користувачем коворкінгу та введення бажаних дат і часу. Далі система перевіряє наявність вільних місць: у разі їх відсутності користувач отримує відповідне повідомлення, і процес завершується.

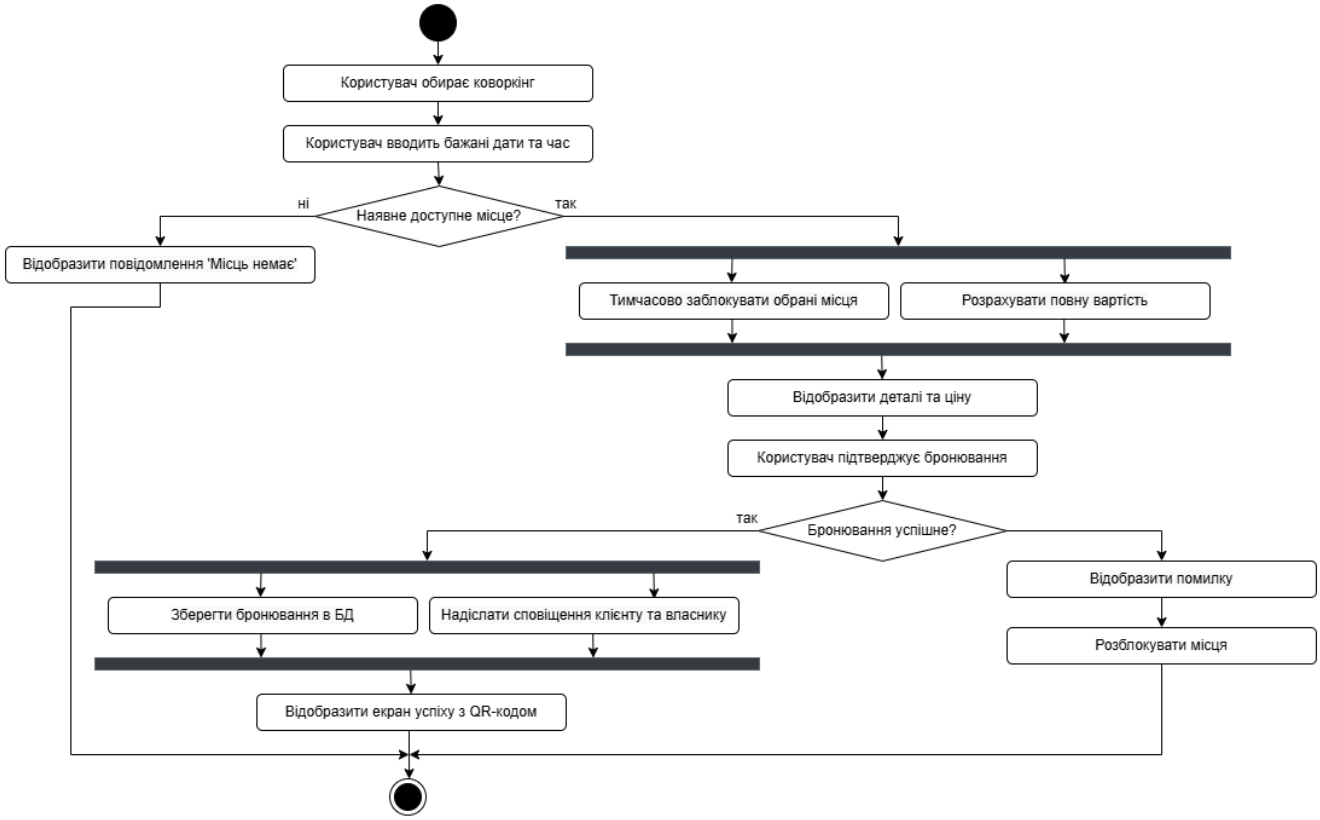

Рисунок 3.4 – Діаграма діяльності «бронювання»

Якщо місця доступні, система ініціює паралельне виконання двох процесів: тимчасове блокування обраних місць та розрахунок повної вартості. Після об’єднання цих потоків користувачу відображаються деталі та фінальна ціна для підтвердження. На наступному етапі перевіряється успішність операції: у разі збою система виводить помилку та розблоковує місця. Якщо ж бронювання успішне, паралельно виконується збереження даних у базу даних та надсилання сповіщень клієнту і власнику. Процес завершується відображенням екрана успішного бронювання з генерованим QR-кодом.

3.1.6 Діаграма розгортання

Діаграма розгортання (рис. 3.5) відображає топологію апаратних засобів та розподіл програмних компонентів системи по фізичних вузлах. Клієнтська частина виконується у середовищі веббраузера на пристрої користувача. Вебсервер містить середовище виконання ASP.NET Core Web API та Kestrel HTTP Server, який обробляє вхідні запити. Сервер баз даних фізично або логічно ізольований та містить СКБД MySQL.

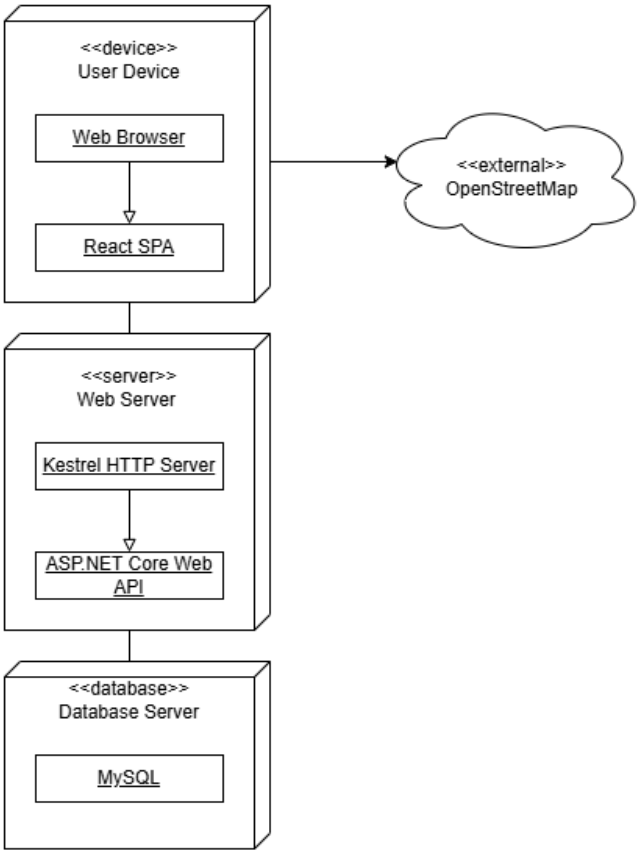

Рисунок 3.5 – Діаграма розгоратння

Взаємодія між клієнтом і вебсервером здійснюється за безпечним протоколом HTTPS, що гарантує захист даних від перехоплення. Додатково на діаграмі відображено інтеграцію із зовнішнім сервісом OpenStreetMap, до якого звертається клієнтський застосунок для завантаження тайлів інтерактивної карти.

3.1.7 Діаграма компонентів

Діаграма компонентів (рис. 3.6) розкриває внутрішню структуру програмного забезпечення та залежності між його логічними модулями. Фронтенд-архітектура побудована за модульним принципом і складається з компонентів інтерфейсу, маршрутизатора сторінок, глобального сховища стану та шару взаємодії з API.

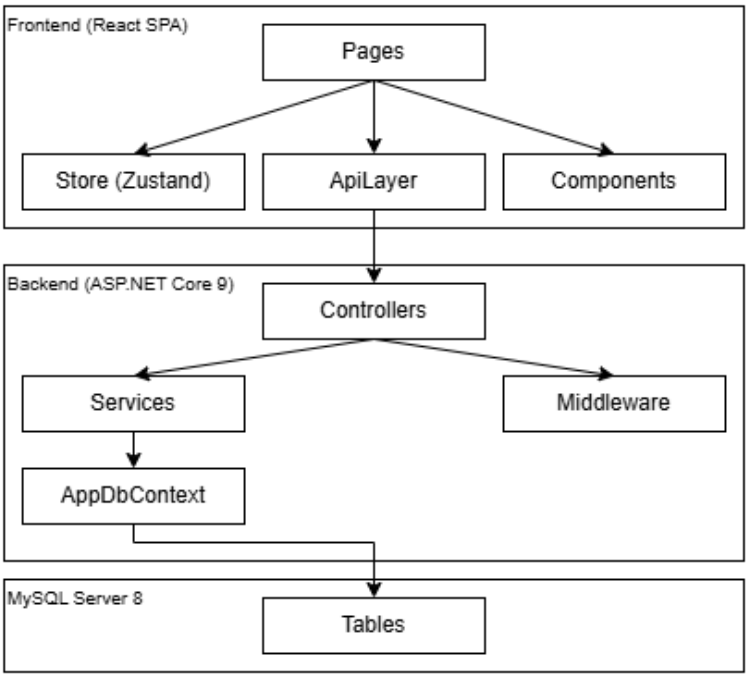

Рисунок 3.6 – Діаграма компонентів

Бекенд-частина реалізує класичний шаблон проєктування багатошарової архітектури. Контролери слугують точками входу для HTTP-запитів, сервіси інкапсулюють правила бізнес-логіки та взаємодіють з Data Access Layer, який за допомогою ORM Entity Framework Core транслює об’єктні запити у таблиці бази даних MySQL. Такий підхід забезпечує слабку зв’язність та високу згуртованість коду.

3.1.8 ER-діаграма бази даних

Для формалізованого відображення структури бази даних розроблено ER-діаграму, що унаочнює всі таблиці системи та зв'язки між ними (рис. 3.7). Діаграму

побудовано за допомогою інструменту MySQL Workbench на основі реальної схеми бази даних, що відображає поточний стан реляційної моделі застосунку.

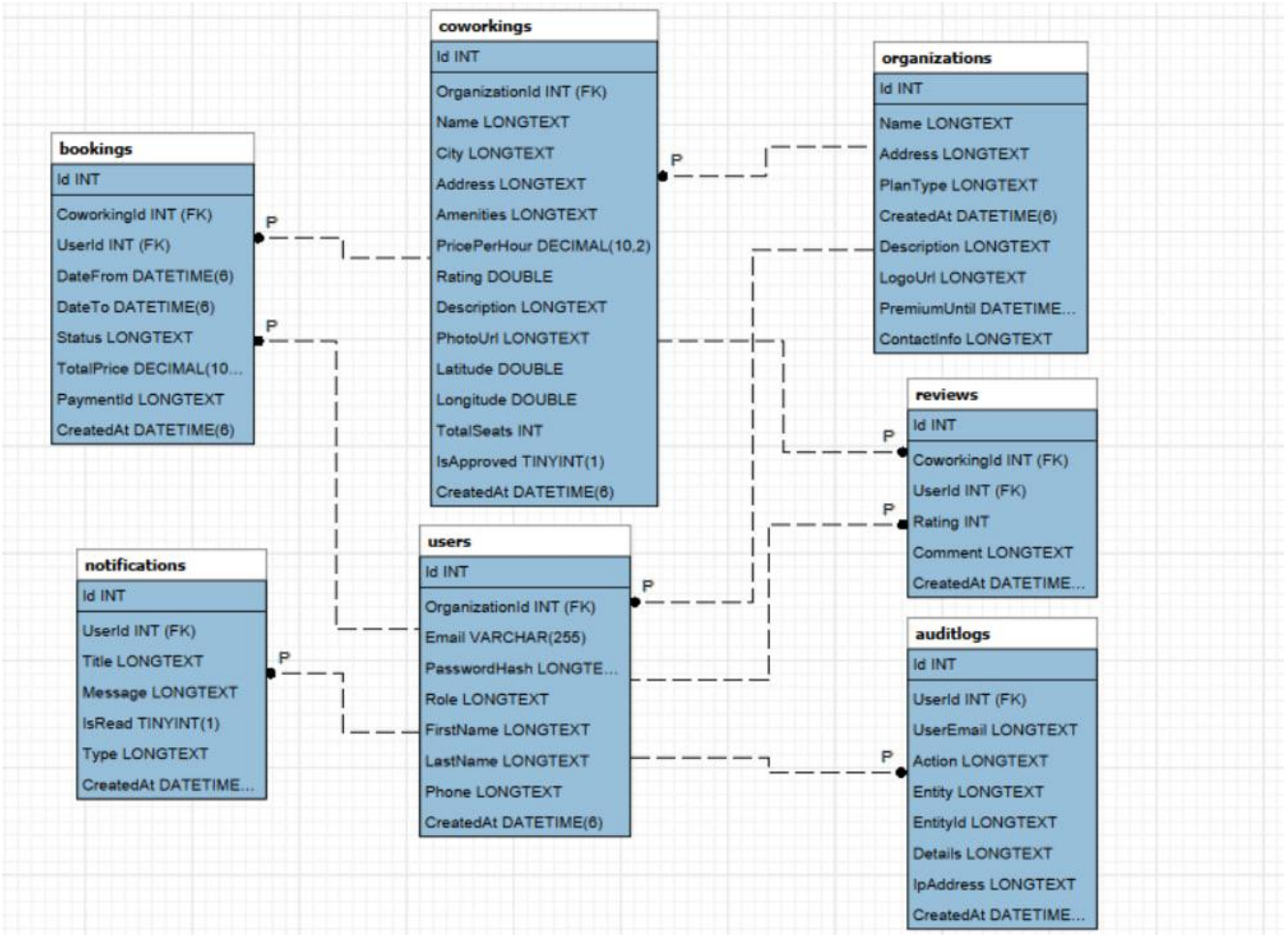

Рисунок 3.7 – ER-діаграма бази даних вебзастосунку

База даних містить сім таблиць: organizations, users, coworkings, bookings, reviews, notifications та auditlogs. Центральною сутністю є таблиця coworkings, що безпосередньо або опосередковано пов'язана з усіма іншими таблицями. Таблиця users є ключовим вузлом реляційної мережі – через неї проходять зв'язки з бронюваннями, відгуками, сповіщеннями та журналом аудиту.

3.2 Програмна реалізація компонентів системи

Процес програмної реалізації вебзастосунку охоплює створення надійної серверної інфраструктури та розробку інтерактивного клієнтського інтерфейсу. Кожен із цих рівнів використовує спеціалізований набір технологій та фреймворків, що дозволяє забезпечити високу продуктивність, безпеку даних та

масштабованість системи. У наступних підрозділах детально розкрито особливості написання програмного коду для обробки бізнес-логіки, взаємодії з базою даних, а також побудови модульної структури клієнтської частини.

3.2.1 Реалізація серверної частини

Точкою входу серверного застосунку є файл Program.cs, де реєструються всі сервіси DI-контейнера, налаштовується Middleware-пайплайн та ініціалізується база даних. Реєстрація сервісів дотримується принципу єдиної відповідальності: кожен сервіс вирішує одну задачу і впроваджується через інтерфейс, що спрощує подальшу заміну реалізацій та тестування.

Система авторизації побудована на JWT Bearer. Після успішної автентифікації AuthService генерує токен з п’ятьма claims: NameIdentifier, Email, Role, GivenName та Surname. Контролери захищаються атрибутами [Authorize] та [Authorize(Roles = "...")] що автоматично валідують токен на кожному запиті.

Шифрування персональних даних реалізовано через EncryptionService на основі алгоритму AES-256 та EF Core ValueConverter. Конвертер прозоро шифрує поля при записі та розшифровує при читанні з бази даних, не вимагаючи жодних змін у коді бізнес-логіки. Ключ шифрування зберігається у конфігурації застосунку.

Ключовим алгоритмом бізнес-логіки є механізм перевірки доступності місць у реальному часі. При спробі створення нового резервування система аналізує обраний користувачем часовий проміжок та звертається до бази даних для підрахунку всіх існуючих активних бронювань, які хоча б частково перетинаються із запитуваним часом. Нове бронювання успішно фіксується лише за умови виконання двох критеріїв: загальна кількість знайдених перетинів має бути суворо меншою за загальну місткість конкретного коворкінгу, а також у системі не повинно бути дублюючих запитів від цього ж користувача на аналогічний час. Такий програмний підхід дозволяє коректно обробляти одночасні запити та ефективно управляти просторами, що мають кілька робочих місць.

Журналювання дій реалізовано через AuditService. Кожна критична операція записується в таблицю AuditLogs із зазначенням типу дії, email, ідентифікатора сутності та IP-адреси клієнта.

3.2.2 Реалізація клієнтської частини

Фронтенд-застосунок організовано за модульним принципом. Директорія pages містить сторінки застосунку, components – UI-компоненти, store – Zustand-сховища глобального стану, api – HTTP-шар на базі Axios, types – TypeScript-інтерфейси та lib – утиліти.

Маршрутизацію реалізовано через React Router. Компонент PrivateRoute перехоплює переходи до захищених сторінок: якщо користувач не авторизований – перенаправляє на /login; якщо авторизований, але без потрібної ролі – на головну сторінку. Публічні маршрути доступні без авторизації.

Глобальний стан розподілено між чотирма Zustand-сховищами. authStore зберігає JWT-токен та дані користувача, отримані з claims. themeStore управляє перемиканням темної та світлої теми з персистентністю в localStorage. favoritesStore та compareStore зберігають ідентифікатори збережених та порівнюваних коворкінгів.

Axios-клієнт налаштований з двома interceptors. Request interceptor автоматично додає заголовок Authorization: Bearer {token} до запитів, що вимагають автентифікації. Response interceptor перехоплює помилку 401 та виконує редирект на /login лише якщо запит містив токен, що запобігає хибному редиректу для публічних запитів.

3.2.3 Реалізація системи сповіщень

In-app сповіщення реалізовано через таблицю Notifications. NotificationService надає два методи: SendAsync для відправки конкретному користувачу та SendToOwnerOfCoworkingAsync для пошуку власника за OrganizationId коворкінгу. Клієнтський компонент NotificationBell опитує ендпоінт GET /notifications/unread-count кожні 30 секунд та відображає лічильник. При

відкритті панелі завантажуються останні 50 сповіщень з можливістю позначення як прочитаних та видалення.

3.2.4 Реалізація аналітики для власників

StatsController надає чотири агреговані ендпоінти з подвійним захистом: атрибут [Authorize(Roles = "owner")] та перевірка IsPremiumActive організації. GET /stats/overview повертає загальну кількість бронювань, підтверджені доходи, середній рейтинг та кількість коворкінгів. GET /stats/revenue-by-month агрегує підтверджені бронювання за місяцями. GET /stats/bookings-by-weekday та GET /stats/popular-hours повертають розподіл завантаженості по днях тижня та годинах. На фронтенді ці дані візуалізуються через Recharts – лінійний графік для доходів та стовпчикові для завантаженості.

3.3 Опис інтерфейсу застосунку

Інтерфейс проєктованого вебзастосунку реалізовано у мінімалістичному стилі з підтримкою світлої та темної тем. TailwindCSS забезпечує консистентність дизайну між усіма компонентами. Адаптивна верстка підтримує екрани від 320 пікселів.

3.3.1 Навігаційне меню та глобальні елементи

Navbar (рис. 3.8) закріплений у верхній частині екрана і містить логотип, навігаційні посилання, кнопку перемикання теми та блок авторизації. Для авторизованих користувачів відображається аватар з першою літерою імені, повне ім’я та спадне меню з посиланням на профіль і функцією виходу. Дзвіночок сповіщень показує кількість непрочитаних повідомлень.

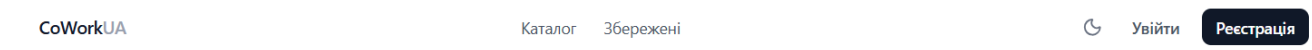

Рисунок 3.8 – Навігаційне меню застосунку

Продумана структура навігаційного меню забезпечує швидкий доступ до всіх основних функцій вебзастосунку незалежно від поточної сторінки. Використання

інтуїтивно зрозумілих іконок та візуального розділення публічних і приватних зон сприяє покращенню користувацького досвіду, дозволяючи користувачеві легко орієнтуватися в системі та миттєво реагувати на нові сповіщення.

3.3.2 Каталог коворкінгів

Сторінка каталогу (рис. 3.9) відображає затверджені коворкінги в адаптивній сітці карток. Кожна картка містить фотографію з бейджом рейтингу, назву, адресу, перелік зручностей, кількість місць, кнопки «Зберегти» і «Порівняти» та ціну за годину. Над сіткою розташовані рядок пошуку, кнопка фільтрів та панель сортування. Кнопка «Карта» перемикає відображення на інтерактивну карту Leaflet з OpenStreetMap. Результати завантажуються порціями через нескінченний скрол.

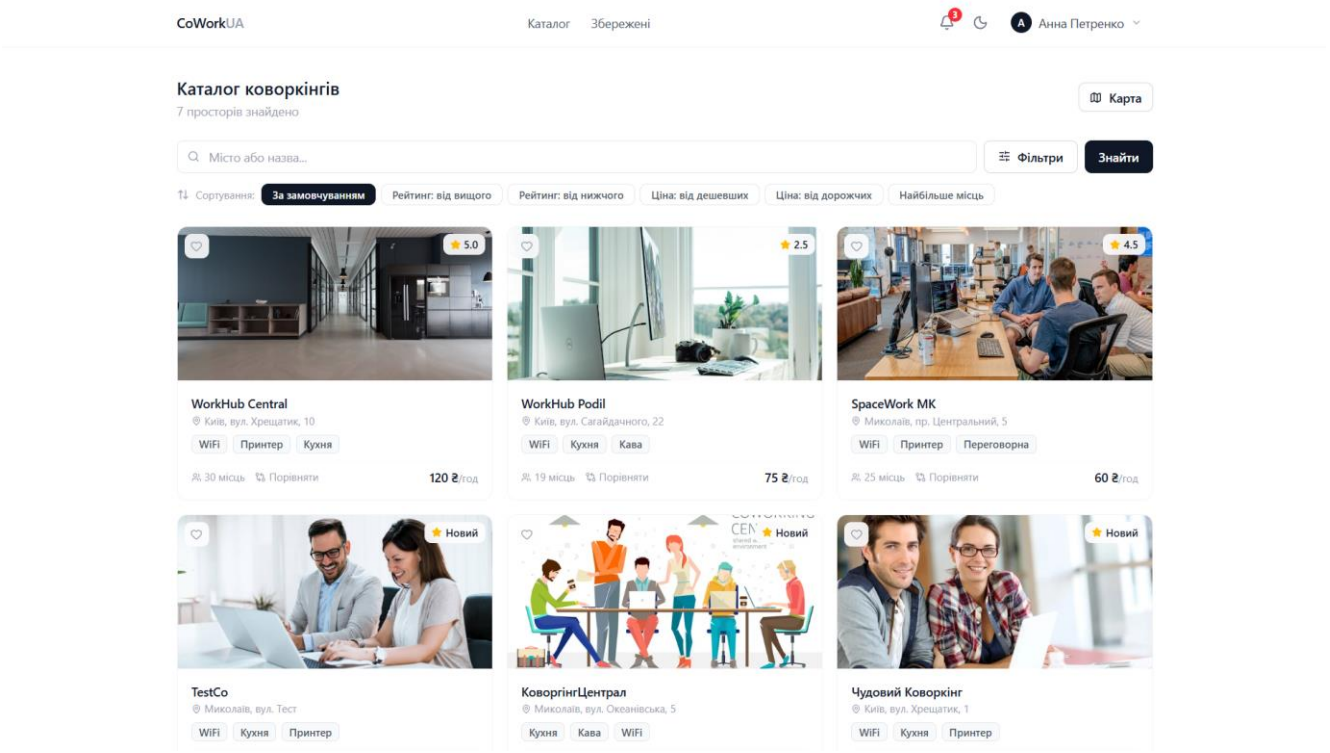

Рисунок 3.9 – Сторінка каталогу коворкінгів

Організація каталогу у вигляді адаптивної сітки дозволяє зручно переглядати доступні варіанти як на десктопних, так і на мобільних пристроях. Наявність розширеної панелі фільтрації та можливості миттєвого перемикання між списком і картою суттєво скорочує час, необхідний клієнту для пошуку оптимального робочого простору, що відповідає його локаційним та фінансовим критеріям.

3.3.3 Сторінка деталей коворкінгу

Детальна сторінка (рис. 3.10) організована у двоколонковому макеті. Ліва колонка містить фотографію, назву, посилання на організацію, опис, зручності з іконками, календар доступності з погодинними слотами та відгуки. Права колонка є панеллю бронювання: відображає ціну, компонент DateTimePicker для вибору дати і часу, індикатор доступності з прогрес-баром та кнопку «Забронювати».

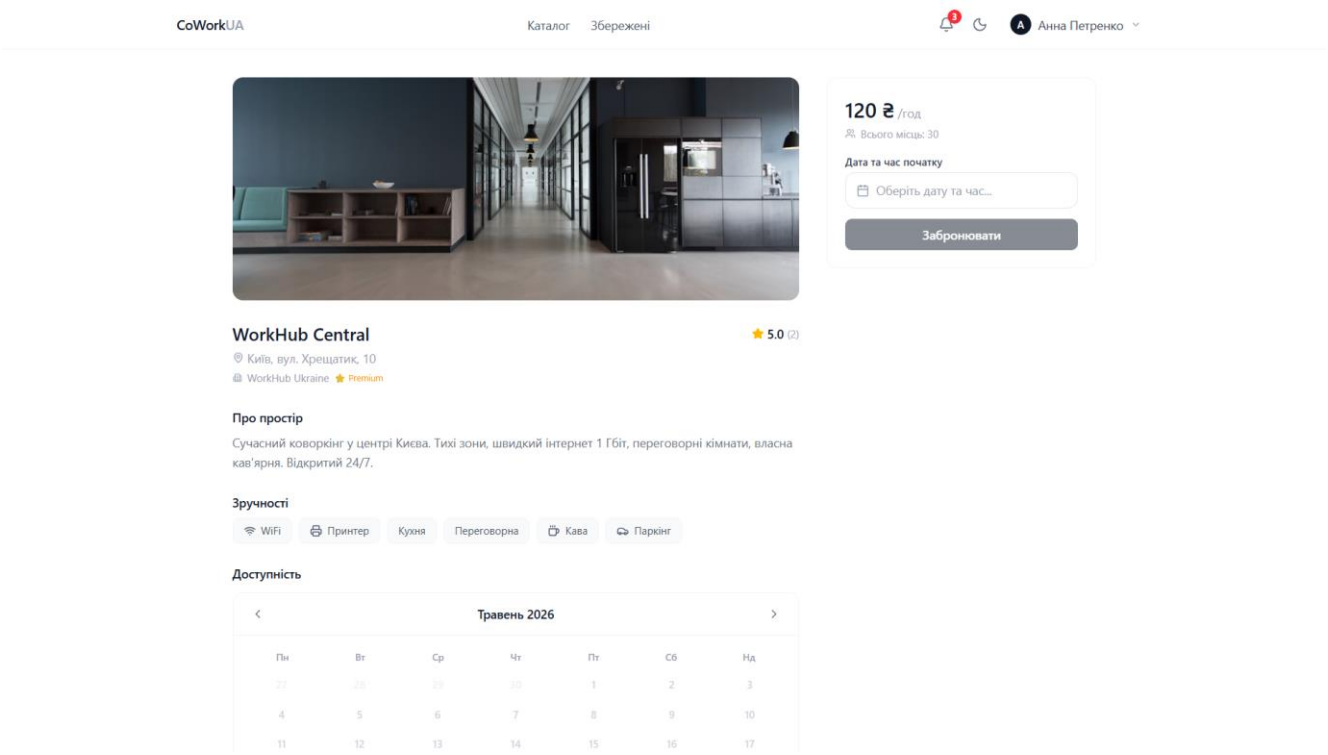

Рисунок 3.10 – Сторінка деталей коворкінгу з панеллю бронювання

Таке візуальне розділення інформації на сторінці деталей гарантує, що клієнт отримує вичерпні дані про простір перед прийняттям рішення. Інтерактивний календар доступності у поєднанні з динамічним розрахунком вартості робить процес оформлення послуги максимально прозорим, зводячи до мінімуму ймовірність помилок під час вибору дат або виникнення конфліктів бронювань.

3.3.4 Особистий кабінет клієнта

Кабінет клієнта (рис. 3.11) містить картку профілю з аватаром, ім’ям та email, статистику бронювань, поле пошуку по бронюваннях та пагінований список. Кожне бронювання відображається карткою зі статусним бейджом, деталями часу

і вартості, кнопкою «Скасувати» (для pending) та «QR-код» (для confirmed). Компонент BookingQR генерує QR-код через qrcode.react та дозволяє завантажити його як SVG.

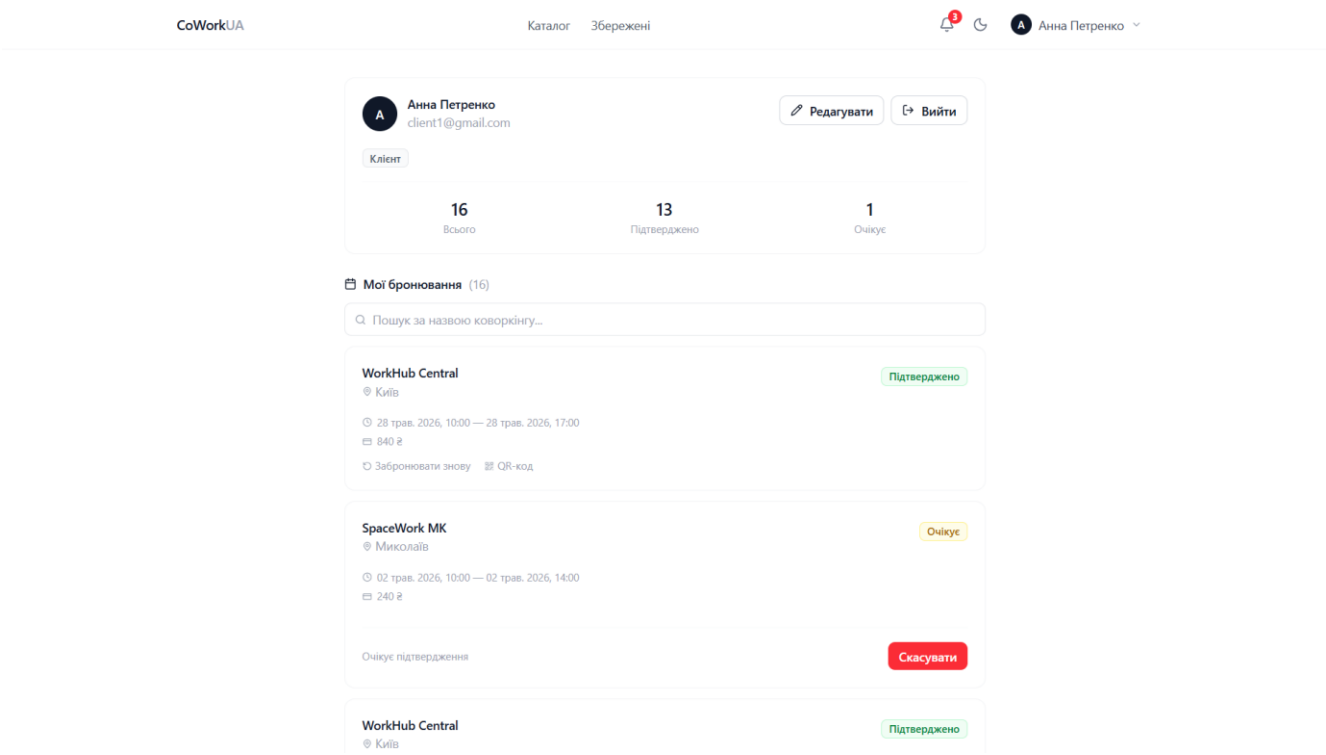

Рисунок 3.11 – Кабінет клієнта зі списком бронювань

Особистий кабінет клієнта спроєктований таким чином, щоб централізувати управління всіма замовленнями в єдиному місці. Завдяки наочній системі статусів, можливості швидкого скасування та інтегрованому генератору QR-кодів, процес використання забронюваного робочого місця стає максимально безшовним та автоматизованим як для користувача, так і для адміністраторів коворкінгів.

3.3.5 Кабінет власника

Кабінет власника (рис. 3.12) включає картку профілю, блок організації з кнопкою редагування, картку тарифного плану, список коворкінгів з пошуком і пагінацією та секцію бронювань клієнтів з фільтрацією за статусом. При активному преміум-плані у навігації з’являється посилання «Статистика».

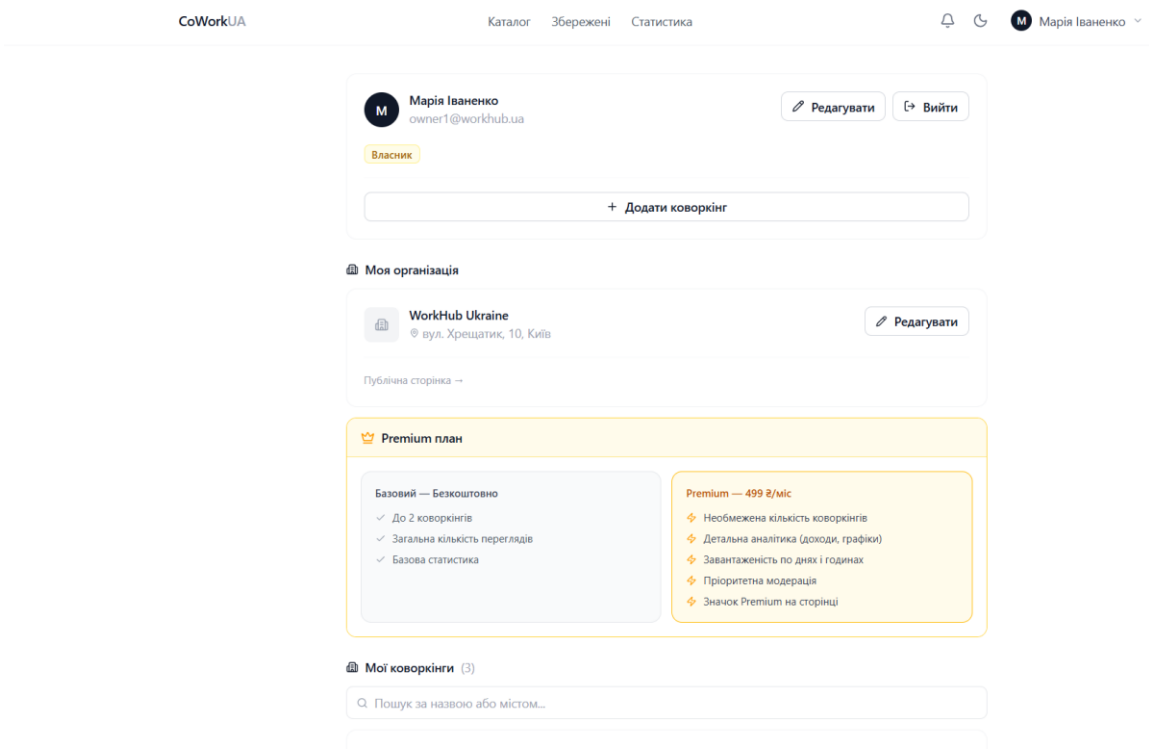

Рисунок 3.12 – Кабінет власника з організацією та коворкінгами

Інтерфейс кабінету власника орієнтований на оперативне управління бізнес-процесами. Структуроване відображення всіх об’єктів нерухомості та вхідних запитів на бронювання дозволяє орендодавцю ефективно контролювати заповнюваність своїх просторів, а гнучка система тарифних планів стимулює до використання розширеного аналітичного інструментарію.

3.3.6 Форма додавання та редагування коворкінгу

Форма (рис. 3.13) містить поля назви, міста, адреси, ціни, кількості місць, опис, URL фото, інтерактивний вибір зручностей та компонент LocationPicker. LocationPicker відображає карту OpenStreetMap, де власник натискає для встановлення маркера або використовує кнопку «Моє місцезнаходження» через Geolocation API. Обрані координати автоматично заповнюють приховані поля форми.

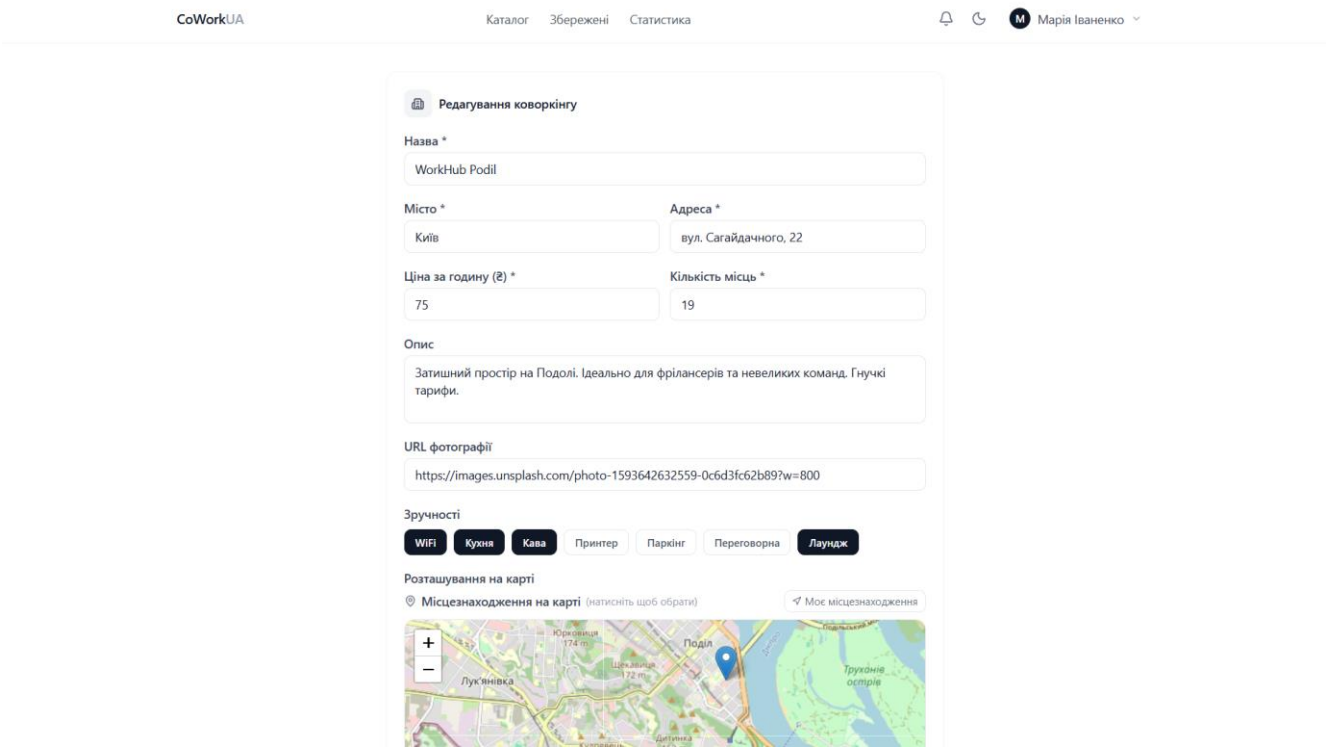

Рисунок 3.13 – Форма додавання коворкінгу з картою вибору локації

Інтеграція інтерактивної карти безпосередньо у форму створення об’єкта значно спрощує процес введення просторових даних для власника. Автоматичне визначення координат та зручний вибір зручностей дозволяють стандартизувати інформацію в базі даних, що в подальшому позитивно впливає на релевантність результатів пошуку для кінцевих клієнтів.

3.3.7 Сторінка аналітики та адміністративна панель

Сторінка аналітики (рис. 3.14) для преміум-власників відображає чотири картки показників та три графіки Recharts:

– доходи за 12 місяців;

– завантаженість по днях тижня;

– популярні години.

Для власників без преміуму відображається заблокований стан із пропозицією апгрейду.

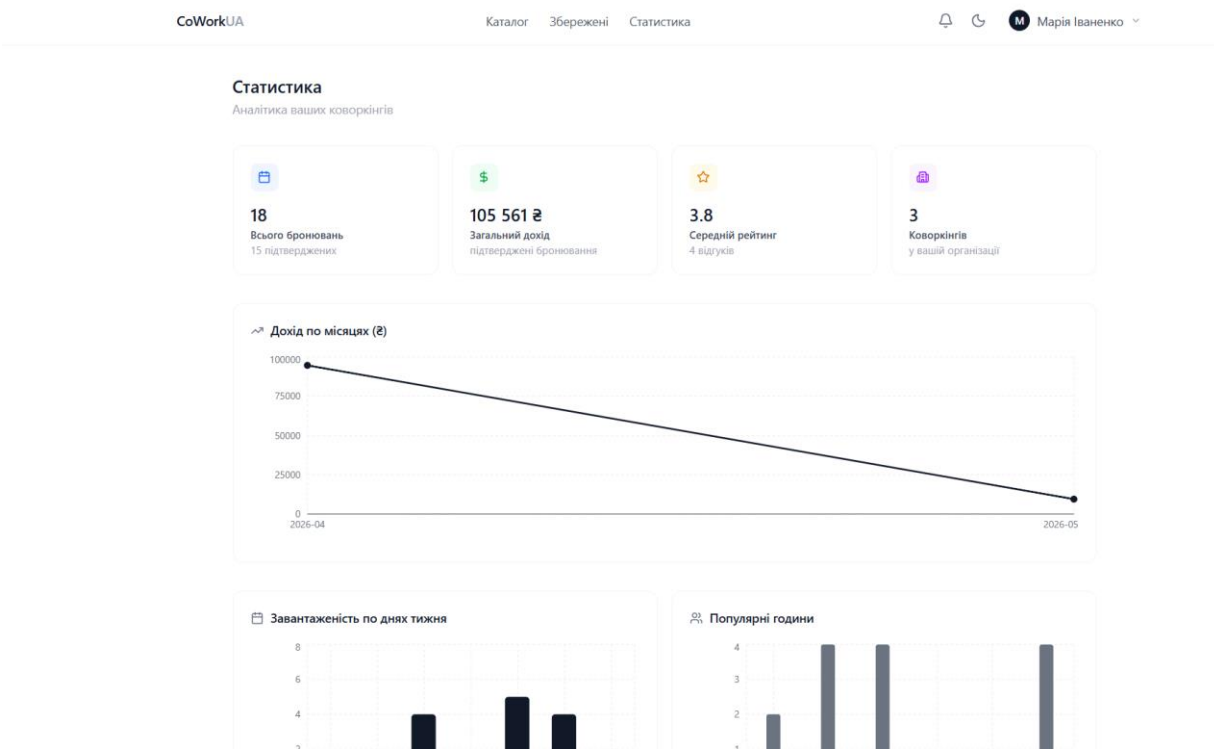

Рисунок 3.14 – Сторінка аналітики власника

Адміністративна панель (рис. 3.15) розділена на вкладки «Коворкінги» та «Бронювання». Кнопка «Журнал дій» відкриває сторінку AuditPage з пошуком та пагінацією.

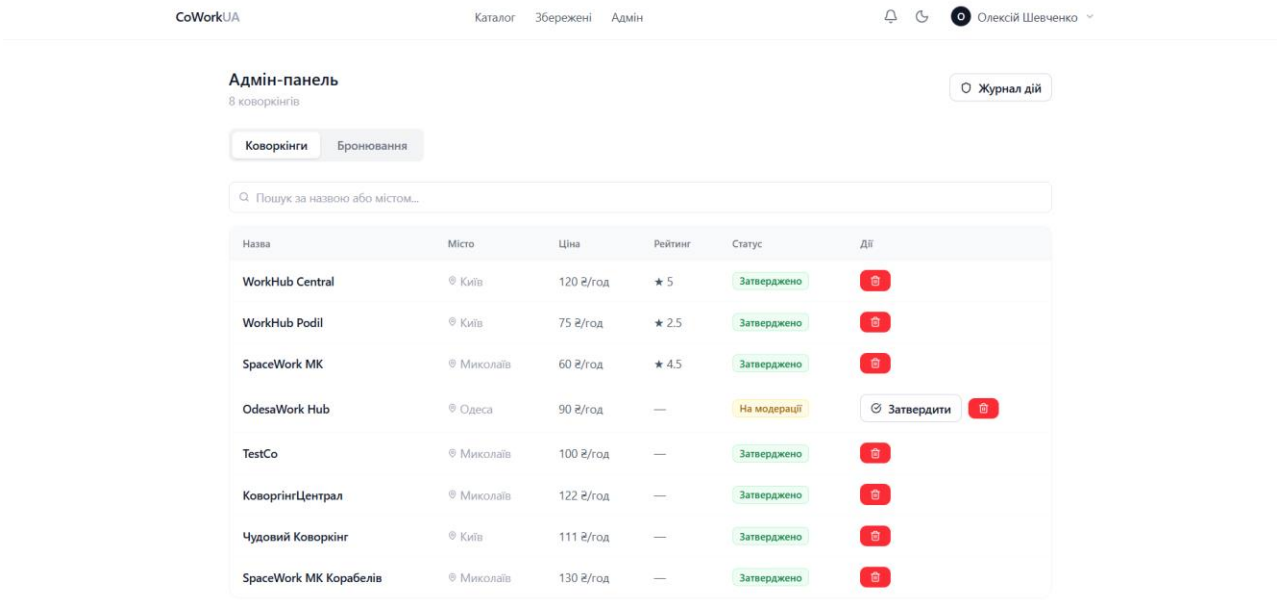

Рисунок 3.15 – Сторінка адміністративної панелі

Адміністративна панель та сторінка аналітики виступають ключовими інструментами для забезпечення якості та стабільності роботи платформи. Наочна

візуалізація фінансових показників допомагає власникам оптимізувати свою діяльність, тоді як інструменти модерації та детальний журнал дій дозволяють адміністраторам підтримувати високий рівень безпеки та достовірності інформації на ресурсі.

Висновки до розділу 3

У третьому розділі представлено проєктні рішення та результати реалізації проєктованого вебзастосунку. Архітектура побудована за триланковою клієнт-серверною моделлю з шаблоном Controller – Service – DbContext, що забезпечує слабку зв’язність та зручність підтримки. Описано шість UML-діаграм: варіантів використання, класів, послідовності, діяльності, розгортання та компонентів.

Реалізовано ключовий алгоритм перевірки доступності місць у транзакційному режимі MySQL. Клієнтська частина включає 7 основних сторінок з адаптивним інтерфейсом, підтримкою двох тем та нескінченним скролом. Описано призначення і взаємодію всіх ключових компонентів: каталог, карта, деталі коворкінгу, кабінети клієнта та власника, форми бронювання та адміністративна панель. Реалізовані рішення повністю відповідають функціональним і нефункціональним вимогам специфікації

4 ТЕСТУВАННЯ ТА СПЕЦИФІКАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Успішна інтеграція розроблених програмних модулів у єдину систему вимагає ретельної формалізації її структури та практичного підтвердження стабільності роботи. Для досягнення цієї мети здійснюється детальна специфікація основних класів, атрибутів і методів як серверної, так і клієнтської частин проєкту. Крім того, критично важливим кроком розробки є етап тестування, який дозволяє перевірити коректність виконання закладеної бізнес-логіки, оцінити надійність транзакцій бронювання та переконатися, що готовий вебзастосунок повністю відповідає висунутим на початкових етапах проєктування вимогам.

4.1 Специфікація основних класів і методів

Для забезпечення прозорості архітектури розробленого вебзастосунку та спрощення його подальшої підтримки чи масштабування необхідно провести детальну специфікацію його програмних компонентів. Опис структурних одиниць системи охоплює визначення ключових сутностей бази даних, об'єктів серверної логіки, а також компонентів та глобальних сховищ стану клієнтської частини. Це дозволяє чітко зафіксувати розподіл відповідальності між шарами триланкової клієнт-серверної архітектури розробленої платформи.

Завдяки застосуванню патерну «Controller – Service – DbContext» на рівні бекенду та компонентно-орієнтованого підходу на рівні фронтенду, кожен елемент системи виконує вузькоспеціалізовану задачу. Така формалізація програмного коду не лише ілюструє практичну реалізацію спроєктованих раніше UML-моделей та зв'язків між ними, але й слугує надійною базою для подальшого модульного тестування, пошуку вразливостей та інтеграції нових функціональних можливостей без ризику порушення цілісності існуючого коду.

4.1.1 Основні сутності та атрибути

У межах проєктування серверної частини розроблено набір моделей, що відображають сутності предметної області у реляційну структуру бази даних за

допомогою технології Entity Framework Core. Кожен клас інкапсулює певний набір властивостей, які визначають інформаційну місткість об'єктів та бізнес-логіку системи. Специфікацію основних сутностей серверної частини наведено в таблиці 4.1.

Таблиця 4.1 – Основні сутності та атрибути серверної частини (Backend)

Клас / Модель	**Атрибут**	**Константа або Змінна**	**Призначення**
Organization	id (int)	Змінна	Унікальний ідентифікатор
	name (string)	Змінна	Назва організації
	address (string)	Змінна	Адреса організації
	planType (string)	Константа переліку	Тип тарифного плану
	premiumUntil (DateTime?)	Змінна	Дата завершення преміуму
	contactInfo (string?)	Змінна	Контактні дані (JSON)
User	id (int)	Змінна	Унікальний ідентифікатор
	email (string)	Змінна	Електронна пошта
	passwordHash (string)	Змінна	Хеш паролю
	role (string)	Константа переліку	Роль користувача
	firstName / lastName (string)	Змінна	Ім’я та прізвище
	phone (string?)	Змінна	Телефон
Coworking	id (int)	Змінна	Унікальний ідентифікатор
	name / city / address (string)	Змінна	Назва, місто, адреса
	pricePerHour (decimal)	Змінна	Ціна за годину
	rating (double)	Змінна	Середній рейтинг
	totalSeats (int)	Змінна	Кількість робочих місць
	isApproved (bool)	Змінна	Статус модерації
	amenities (string)	Змінна	Зручності
Booking	id (int)	Змінна	Унікальний ідентифікатор
	dateFrom / dateTo (DateTime)	Змінна	Час початку та завершення

Кінець таблиці 4.1

	status (string)	Константа переліку	Статус бронювання
	totalPrice (decimal)	Змінна	Загальна вартість
Review	id (int)	Змінна	Унікальний ідентифікатор
	rating (int)	Змінна	Оцінка відгуку
	comment (string?)	Змінна	Текст коментаря
Notification	id (int)	Змінна	Унікальний ідентифікатор
	type (string)	Константа переліку	Тип сповіщення
	isRead (bool)	Змінна	Ознака прочитання
AuditLog	action (string)	Константа переліку	Тип зафіксованої дії
	entity / entityId (string)	Змінна	Сутність та її ідентифікатор

Клас Organization виступає кореневою сутністю для власників, де поле planType обмежене константами “basic” або “premium”, а premiumUntil оновлюється динамічно. Конфіденційні атрибути класу User (email, ПІБ, телефон) перед збереженням у базу даних шифруються алгоритмом AES-256 через EF Core ValueConverter. Для класу Coworking закладено початковий статус isApproved = false (очікування модерації) та автоматичний перерахунок змінного поля rating на основі відгуків. Клас Booking фіксує стан транзакції обмеженим набором статусів (“pending”, “confirmed”, “cancelled”), а його змінна totalPrice розраховується автоматично залежно від тривалості бронювання та ціни простору.

У таблиці 4.2 наведено специфікацію основних компонентів, сторів та службових об’єктів клієнтської частини.

Таблиця 4.2 – Основні компоненти та стори клієнтської частини (Frontend)

Компонент / Store	**Поле / Функція**	**Тип**	**Конст. / Змінна**	**Призначення**
authStore	token	string | null	Змінна	JWT-токен авторизації
	user	Partial<User> | null	Змінна	Дані поточного користувача

Продовження таблиці 4.2

	setToken()	fn(string): void	-	Зберегти токен, оновити user
	updateUser()	fn(Partial<User>): void	-	Оновити поля без нового токена
	logout()	fn(): void	-	Очистити токен і user
themeStore	isDark	boolean	Змінна	Поточна тема інтерфейсу
	toggle()	fn(): void	-	Перемикач темна/світла тема
favoritesStore	ids	number[]	Змінна	ID збережених коворкінгів
	toggle(id)	fn(number): void	-	Додати/видалити зі збережених
compareStore	ids	number[] (max 3)	Константа	ID обраних для порівняння
	add(id)	fn(number): boolean	-	Додати до порівняння
CoworkingCard	cw: Coworking	Props	Константа пропса	Дані коворкінгу
	fav (локальний)	boolean	Змінна	Чи збережено у вподобаних
	inCompare (локальний)	boolean	Змінна	Чи обрано для порівняння
CoworkingDetailPage	cw	Coworking | null	Змінна	Деталі коворкінгу
	dateFrom / dateTo	Date | null	Змінна	Обраний час бронювання
	availability	Availability | null	Змінна	Кількість вільних місць

Кінець таблиці 4.2

	HOURS = [8..22]	number[]	Константа	Дозволений діапазон годин
api/axios.ts	baseURL	string	Константа .env	Базова URL API
	interceptor (request)	fn	-	Додає Bearer-токен до заголовків
	interceptor (response)	fn	-	Перенаправляє на /login при 401

Zustand authStore зберігає JWT-токен у localStorage та парсить з нього поля користувача при кожному оновленні. Метод updateUser() дозволяє оновити відображувані дані без перевидачі токена – це ключовий механізм реактивного оновлення UI після редагування профілю. Стор themeStore застосовує клас dark до кореневого елемента <html> синхронно до першого рендеру React, що виключає мерехтіння при завантаженні сторінки.

4.1.2 Клас Booking та контролер BookingsController

У таблиці 4.1 наведено перелік основних класів серверної частини застосунку, їх атрибутів, типів даних та призначення.

Центральною сутністю процесу бронювання є клас Booking. Лістинг коду класа Booking наведено нижче.

```
public class Booking
{
    public int Id { get; set; }
    public int CoworkingId { get; set; }
    public int UserId { get; set; }
    public DateTime DateFrom { get; set; }
    public DateTime DateTo { get; set; }
    public string Status { get; set; } = "pending";
    public decimal TotalPrice { get; set; }
    public string? PaymentId { get; set; }
    public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
    public Coworking Coworking { get; set; } = null!;
    public User User { get; set; } = null!;
}
```

Навігаційні властивості Coworking та User забезпечують завантаження пов’язаних даних через Include(). Поле Status ініціалізується рядком "pending", що

є константою за замовчуванням — статус змінюється лише через методи Confirm() або Cancel().

У таблиці 4.3 наведено специфікацію методів контролера BookingsController, що обробляє HTTP-запити, пов’язані з бронюваннями.

Таблиця 4.3 – Методи BookingsController

Метод	HTTP	Маршрут	Вхідні дані	Повертає	Роль
GetMy	get	/bookings/my	-	Список бронювань	client
GetAll	get	/bookings/all	-	Всі бронювання	owner / admin
CheckAvailability	get	/bookings/availability	coworkingId, dateFrom, dateTo	Кількість вільних місць	public
GetBusyDays	get	/bookings/busy-days	coworkingId, from, to	Словник дата – к-ть	public
GetDaySlots	get	/bookings/day-slots	coworkingId, date	Погодинні слоти	public
Create	post	/bookings	BookingCreateDto	Деталі бронювання	client
Confirm	patch	/bookings/{id}/confirm	id	NoContent	owner / admin
Cancel	patch	/bookings/{id}/cancel	id	NoContent	client

Метод Create() реалізує перевірку трьох ділових правил: допустимого діапазону годин, відсутності дублікату бронювання для того самого користувача та наявності вільних місць. У додатку А наведено ключові фрагменти реалізації.

4.1.3 authStore та реактивне оновлення даних

Глобальний стан авторизації управляється стором authStore на базі бібліотеки Zustand. Ключовою особливістю реалізації є те, що після редагування профілю сервер повертає новий JWT-токен з оновленими claims, а метод setToken() автоматично парсить його та оновлює user у стані без перезавантаження сторінки. Лістинг фрагменту коду authStore наведено нижче.

```
export const useAuthStore = create<AuthState>((set, get) => ({
  token: localStorage.getItem('token'),
  user: (() => {
    const t = localStorage.getItem('token')
    return t ? parseUser(t) : null
```

```
  })(),
  setToken: (token) => {
    localStorage.setItem('token', token)
    set({ token, user: parseUser(token) })
  },
  updateUser: (data) => {
    set(state => ({
      user: state.user ? { ...state.user, ...data } : data
    }))
  },
  logout: () => {
    localStorage.removeItem('token')
    set({ token: null, user: null })
  },
  isAuthenticated: () => !!get().token,
}))
```

Функція parseUser() декодує JWT-токен через jwt-decode та витягує поля id, email, role, firstName, lastName. Завдяки цьому ім'я користувача в навбарі, профілі та у відгуках оновлюється миттєво після збереження змін

4.2 Тестування програмного забезпечення

Для перевірки коректності роботи застосунку використовувалися два підходи: модульне тестування фронтенду за допомогою Vitest та інтерактивне тестування REST API через Swagger UI, вбудований у бекенд.

4.2.1 Модульне тестування за допомогою Vitest

Vitest – це сучасний фреймворк тестування для TypeScript/JavaScript-проєктів, сумісний з Vite. Vitest підтримує Jest-синтаксис, має вбудований coverage-репортер та працює значно швидше за Jest завдяки використанню ESM-трансформацій Vite.

Тестуванню підлягали ключові модулі та утиліти застосунку: Zustand, бізнес-логіка бронювань, а також допоміжні утиліти пагінації. Усі тести організовано у вигляді окремих файлів у директорії src/__tests__/ та згруповано за функціональним призначенням. У додатку Б наведено приклад тестів для authStore.

Перелік протестованих модулів:

– themeStore – управління темою інтерфейсу, 6 тестів;

– authStore – аутентифікація користувача, оновлення даних профілю, вихід із системи, перевірка стану автентифікації, 14 тестів;

– compareStore – управління списком порівняння коворкінгів, 11 тестів.

– favoritesStore – управління списком обраних коворкінгів, 8 тестів.

– bookingUtils – утиліта форматування дати toLocalISOString та перевірка бізнес-правил бронювання,16 тестів.

– pagination – утиліта пагінації, 10 тестів

Результати виконання тестів наведено у таблиці 4.4.

Таблиця 4.4 – Методи BookingsController

№	Тестовий файл	Кількість тестів	Час виконання	Статус
1	pagination.test.ts	10	6 мс	Пройдено
2	favoritesStore.test.ts	8	10 мс	Пройдено
3	compareStore.test.ts	11	11 мс	Пройдено
4	authStore.test.ts	14	14 мс	Пройдено
5	themeStore.test.ts	6	5 мс	Пройдено
6	bookingUtils.test.ts	16	8 мс	Пройдено

Як видно з таблиці 4.4, усі 65 тестів завершились зі статусом «Пройдено». Жодного зафіксованого відхилення від очікуваної поведінки не виявлено. Найбільше тестів містить модуль bookingUtils, що обумовлено широким спектром бізнес-правил, які потребують перевірки: форматування дати, граничні значення часу, валідність часового інтервалу та обчислення вартості бронювання.

4.2.2 Тестування REST API через Swagger UI

Swagger UI (рис. 4.1) підключений до бекенду через пакет Swashbuckle.AspNetCore та доступний за адресою https://localhost:7120/swagger у режимі розробки. Інтерфейс дозволяє виконувати HTTP-запити до всіх ендпоінтів з авторизацією через Bearer-токен, що суттєво прискорює ручне тестування без необхідності використання сторонніх інструментів.

Процедура тестування через Swagger:

– виконати POST /api/auth/login з коректними email та password – отримати JWT-токен;

– натиснути «Authorize» у Swagger UI та вставити токен у форматі Bearer {token};

– виконати запити до захищених ендпоінтів;

– перевірити HTTP-коди відповіді та JSON-тіло у розділі «Responses».

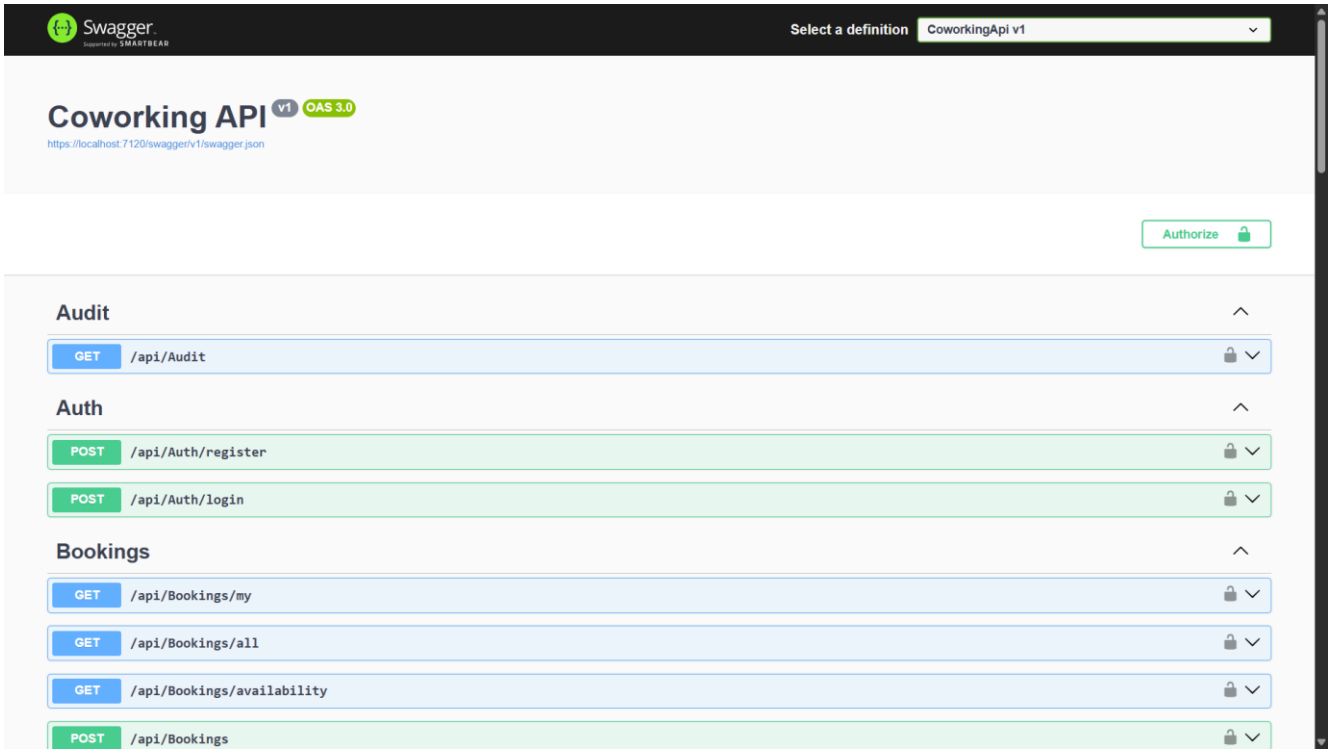

Рисунок 4.1 – Інтерфейс Swagger UI з переліком ендпоінтів та авторизацією

Проведене тестування контролера бронювань та пов’язаних класів серверної частини підтвердило високу надійність реалізованої бізнес-логіки. Усі ключові транзакції, включаючи перевірку доступності місць у реальному часі та валідацію часових інтервалів, успішно пройшли перевірку, демонструючи коректну обробку як стандартних, так і виняткових ситуацій. Це гарантує стабільну роботу механізму бронювання та запобігає виникненню конфліктів даних у базі при одночасних запитах від різних користувачів.

4.3 Результати рішення

У цьому підрозділі наведено результати тестування застосунку при різних вхідних даних: як валідних, так і невалідних. Метою є підтвердження коректності обробки граничних випадків та оцінка якості розробленого програмного забезпечення.

4.3.1 Модульне тестування за допомогою Vitest

За результатами виконання повного набору модульних тестів можна зробити висновок про високий рівень надійності клієнтської частини вебзастосунку (рис. 4.2). Усі 65 тестів у 6 тестових файлах пройшли успішно без жодної помилки чи попередження.

```
PS E:\unik\Diploma\frontend\coworking-client> npm run test:watch

> coworking-client@0.0.0 test:watch
> vitest


 DEV  v4.1.7 E:/unik/Diploma/frontend/coworking-client

 ✓ src/__tests__/pagination.test.ts (10 tests) 6ms
 ✓ src/__tests__/favoritesStore.test.ts (8 tests) 10ms
 ✓ src/__tests__/compareStore.test.ts (11 tests) 11ms
 ✓ src/__tests__/authStore.test.ts (14 tests) 14ms
 ✓ src/__tests__/themeStore.test.ts (6 tests) 5ms
 ✓ src/__tests__/bookingUtils.test.ts (16 tests) 8ms

 Test Files  6 passed (6)
      Tests  65 passed (65)
   Start at  16:37:39
   Duration  6.90s (transform 378ms, setup 459ms, import 3.06s, tests 55ms, environment 10.33s)
```

Рисунок 4.2 – Результати тестування vitest

Аналіз результатів тестування дозволяє виділити наступні ключові характеристики якості розроблених модулів:

– коректність управління станом – усі операції зміни стану виконуються відповідно до специфікації;

– надійність аутентифікації – коректно верифіковано парсинг JWT-токена, включно з витяганням полів email, role, firstName, lastName з нестандартних

claim-ключів Microsoft/XMLSOAP, перевірено ізольованість полів при частковому оновленні даних користувача через updateUser;

– стійкість до граничних значень – утиліти демонструють правильну поведінку при крайових значеннях;

– персистентність даних – перевірено коректне збереження та відновлення даних із localStorage ;

– продуктивність тестів – загальний час виконання 65 тестів склав лише 55 мс, що свідчить про відсутність надлишкових залежностей та ефективну ізоляцію тестового середовища, середній час виконання одного тесту – менше 1 мс.

Отримані результати підтверджують, що реалізована клієнтська частина вебзастосунку відповідає встановленим функціональним вимогам у частині роботи зі станом застосунку, аутентифікацією, бізнес-логікою бронювань та допоміжними утилітами. Виявлених дефектів та відхилень від очікуваної поведінки в ході тестування не зафіксовано.

Таким чином, модульне тестування за допомогою Vitest підтвердило готовність frontend-модулів до інтеграції із backend-частиною системи та подальшого розгортання в рамках повного вебзастосунку для пошуку коворкінгу.

4.3.2 Оцінка якості та продуктивності API

Для оцінки швидкості виконання API-запитів використовувався Swagger UI та Postman. Кожен запит виконувався 5 разів, після чого розраховувався середній та максимальний час відповіді. Тестування проводилось на локальному середовищі розробки. Результати наведено у таблиці 4.5.

Таблиця 4.5 – Продуктивність API-запитів

Endpoint	**Метод**	**Сер. час відп., мс**	**Макс. час відп., мс**	**Статус**	**Відповідність вимогам**
GET /api/coworkings?page=1	get	85	210	200 OK	Так

Кінець таблиці 4.5

GET /api/coworkings/{id}	get	42	130	200 OK	Так
POST /api/auth/login	post	310	480	200 OK	Так
POST /api/bookings	post	195	390	200 OK	Так
GET /api/bookings/availability	get	58	140	200 OK	Так
GET /api/stats/revenue-by-month	get	120	280	200 OK	Так
GET /api/coworkings/top	get	95	220	200 OK	Так
POST /api/reviews	post	175	340	200 OK	Так

Усі ендпоінти відповідають вимогам до продуктивності: час відповіді на читання даних не перевищує 250 мс, на запис – 500 мс. Це відповідає визначеним нефункціональним вимогам: < 200 мс для стандартних запитів та < 2 с для пошукових. Найвищий час відповіді зафіксовано для ендпоінту авторизації POST /api/auth/login, що зумовлено обчислювально-інтенсивним алгоритмом BCrypt.

Оцінка якості програмного забезпечення за основними критеріями:

1) функціональна повнота – реалізовано всі use-case’и для чотирьох ролей;

2) безпека – AES-256-шифрування персональних даних, BCrypt-хешування паролів, rate limiting, журнал аудиту;

3) надійність – централізована обробка помилок через ExceptionMiddleware, валідація вхідних даних на рівні DTO;

4) зручність – адаптивний інтерфейс, підтримка темної теми, нескінченний скрол, календар доступності.

Підсумовуючи результати тестування користувацького інтерфейсу та загальної взаємодії компонентів системи, можна стверджувати, що розроблена клієнтська частина повністю відповідає вимогам ергономіки та адаптивності. Успішне проходження тестових сценаріїв підтверджує коректність роботи модулів пошуку, фільтрації та інтерактивної карти, а також гарантує безперебійну і зручну взаємодію з платформою як для клієнтів, так і для власників коворкінг-просторів.

4.4 Керівництво користувача

Розроблюваний застосунок орієнтований на чотири категорії користувачів: гість, клієнт, власник та адміністратор. У цьому підрозділі наведено послідовний опис роботи з системою для кожної ролі, проілюстрований скріншотами інтерфейсу.

4.4.1 Сценарій роботи гостя

Гість – незареєстрований відвідувач, що має доступ до перегляду каталогу, пошуку, карти та читання відгуків.

Крок 1. Перехід на головну сторінку. Гість потрапляє на головну сторінку застосунку, де відображаються секція з кнопками «Переглянути каталог» та «Зареєструватись», а також блок «Топ-3 коворкінги тижня» (рис. 4.3).

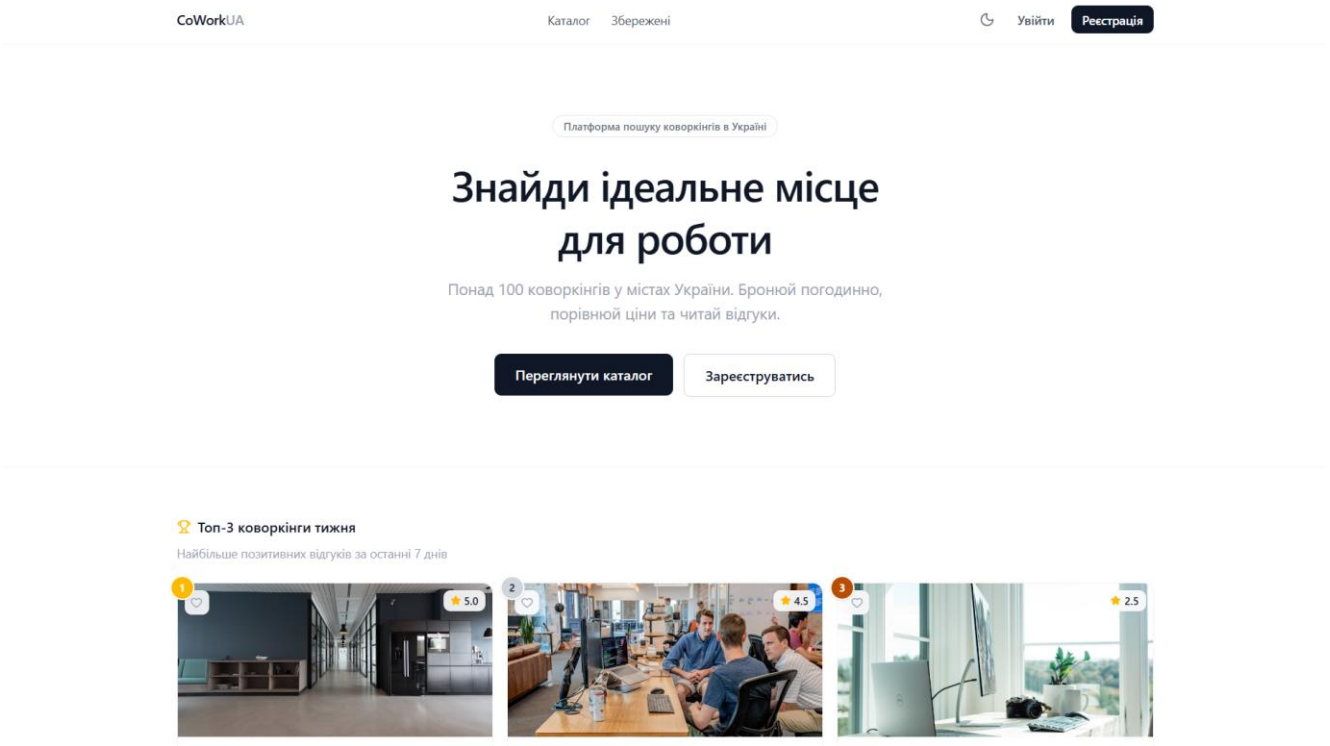

Рисунок 4.3 – Головна сторінка вебзастосунку

Крок 2. Перехід до реєстрації. Гість переходить на сторінку реєстрації за посиланням у навбарі або кнопкою на головній сторінці. Сторінка відображає форму для реєстрації з полями: Ім’я, Прізвище, Email, Пароль, Телефон, Роль. (рис. 4.4).

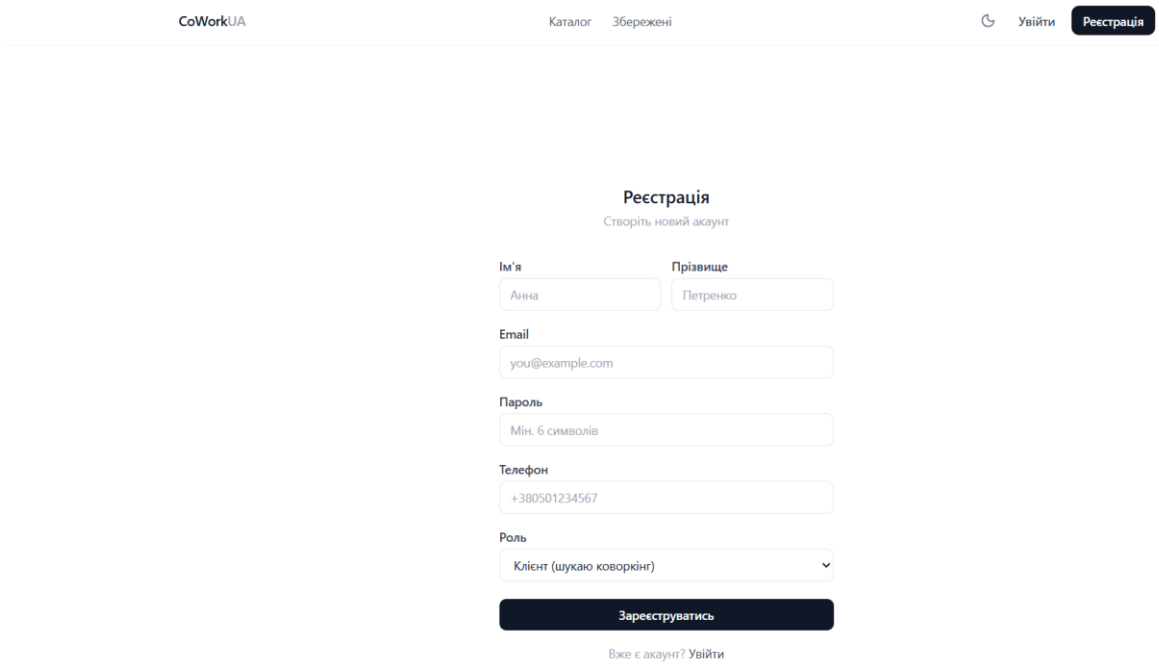

Рисунок 4.4 – Сторінка реєстрації

Крок 3. Реєстрація. Гість має ввести дані в поля форми та обрати роль. Після даних дій натиснути кнопку «Застосувати».

4.4.2 Сценарій роботи клієнта

Клієнт – зареєстрований користувач, який може бронювати місця, залишати відгуки та зберігати коворкінги у список обраного.

Крок 1. Перегляд каталогу. Клієнт переходить до каталогу за посиланням у навбарі або кнопкою на головній сторінці. Сторінка відображає список коворкінгів з нескінченним скролом та можливістю фільтрації.

Крок 2. Бронювання коворкінгу. Клієнт переходить на детальну сторінку коворкунгу і обирає дату та час початку у першому DatePicker. Після вибору дати автоматично з’являється другий DatePicker лише для часу завершення. Система перевіряє доступність і відображає індикатор (рис. 4.5).

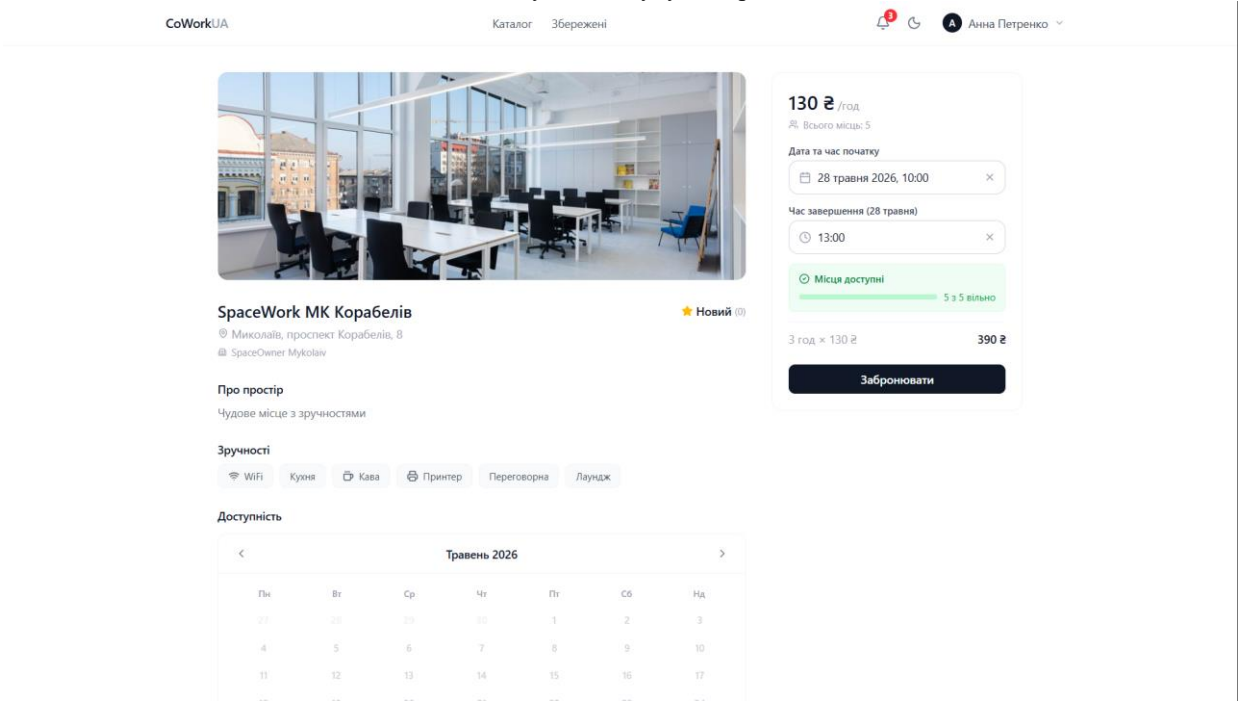

Рисунок 4.5 – Форма бронювання

Крок 3. Перегляд бронювань у профілі. Після входу клієнт переходить до профілю, де відображається картка з ім’ям, email, статистикою бронювань та секцією «Мої бронювання» з пагінацією та пошуком.

Крок 4. Залишення відгуку. Клієнт, що має підтверджене бронювання у коворкінгу, може залишити відгук прямо на детальній сторінці. Форма відображається лише якщо клієнт ще не залишав відгук для цього коворкінгу.

Крок 5. Збережені коворкінги. На картці коворкінгу є кнопка-серце, що зберігає його до списку обраного у localStorage. Всі збережені коворкінги доступні на сторінці /favorites.

Крок 6. Редагування профілю. Клієнт натискає «Редагувати» у профілі та оновлює ім’я, прізвище або телефон. При зміні пароля необхідно ввести поточний. Після збереження дані в навбарі та профілі оновлюються миттєво..

4.4.3 Сценарій роботи власника

Власник – підприємець, що надає коворкінги в оренду. Перед додаванням коворкінгу необхідно створити організацію.

Крок 1. Реєстрація та вибір ролі «Власник». При реєстрації у полі «Роль» обирається «Власник (маю коворкінг)».

Крок 2. Створення організації. У профілі відображається блок «Моя організація» з кнопкою «Створити організацію». Власник заповнює форму: назва, адреса, опис, URL логотипу. Контактна інформація додається динамічно для кожного типу або як довільне поле (рис. 4.6).

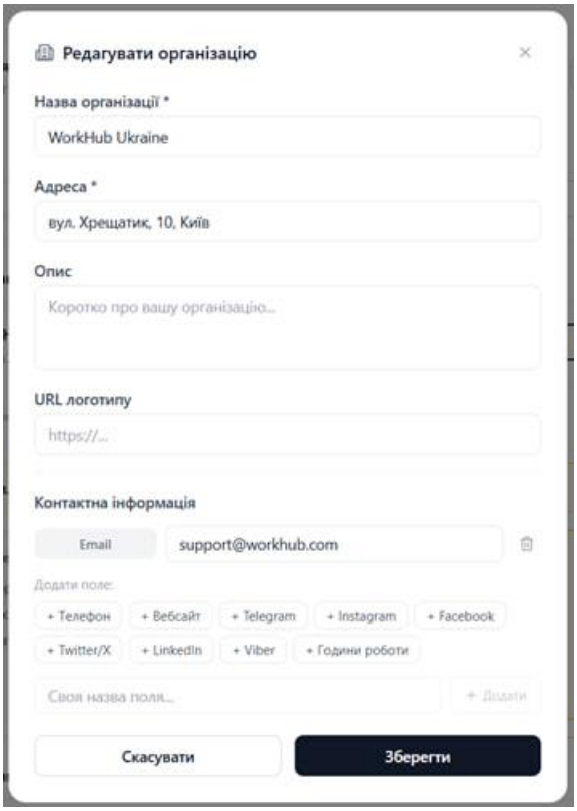

Рисунок 4.6 – Форма створення\редагування організації з динамічними контактними полями

Крок 3. Додавання коворкінгу. Після створення організації кнопка «Додати коворкінг» стає активною. Власник заповнює форму: назва, місто, адреса, ціна, кількість місць, зручності, опис, URL фото та координати.

Крок 4. Перегляд та підтвердження бронювань. У профілі власника відображається блок «Бронювання клієнтів» з фільтром по статусу. Власник може підтверджувати або скасовувати бронювання, що очікують. Після підтвердження клієнт отримує сповіщення (рис. 4.7).

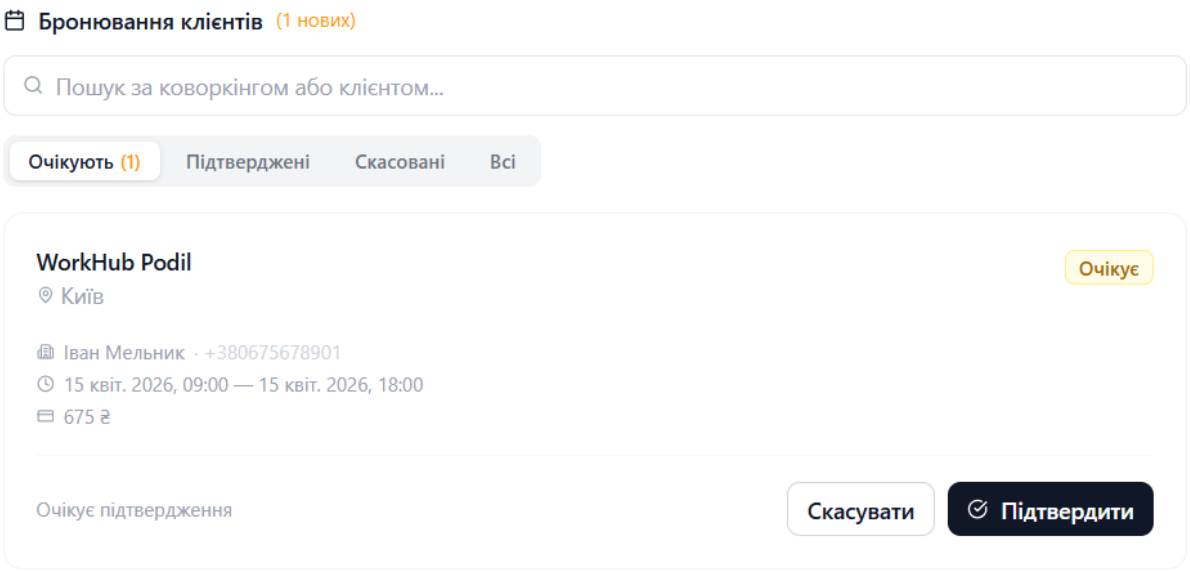

Рисунок 4.7 – Бронювання клієнтів з фільтром по статусу

Крок 5. Перегляд статистики. Власники з активним преміум-планом мають доступ до сторінки /stats з графіками доходів по місяцях, завантаженістю по днях тижня та популярними годинами.

Крок 6. Редагування профілю організації та просторів. Власник має можливість гнучко управляти інформацією про свій бізнес. За допомогою кнопки «Редагувати» у блоці організації можна актуалізувати загальні дані: назву, опис, логотип та динамічні контактні поля. Для кожного доданого коворкінгу також передбачена функція редагування, яка дозволяє власнику оперативно оновлювати ціну оренди, кількість доступних місць, перелік зручностей, фотографії та розташування на інтерактивній карті відповідно до поточного стану простору.

4.4.4 Сценарій роботи адміністратора

Адміністратор – користувач, що управляє платформою: модерує коворкінги, переглядає бронювання та журнал аудиту.

Крок 1. Вхід та перехід до адмін-панелі. Адміністратор входить до системи з роллю admin. У навбарі відображається додаткове посилання «Адмін» та кнопка «Журнал дій». При переході на /admin відкривається адмін-панель (рис. 4.8).

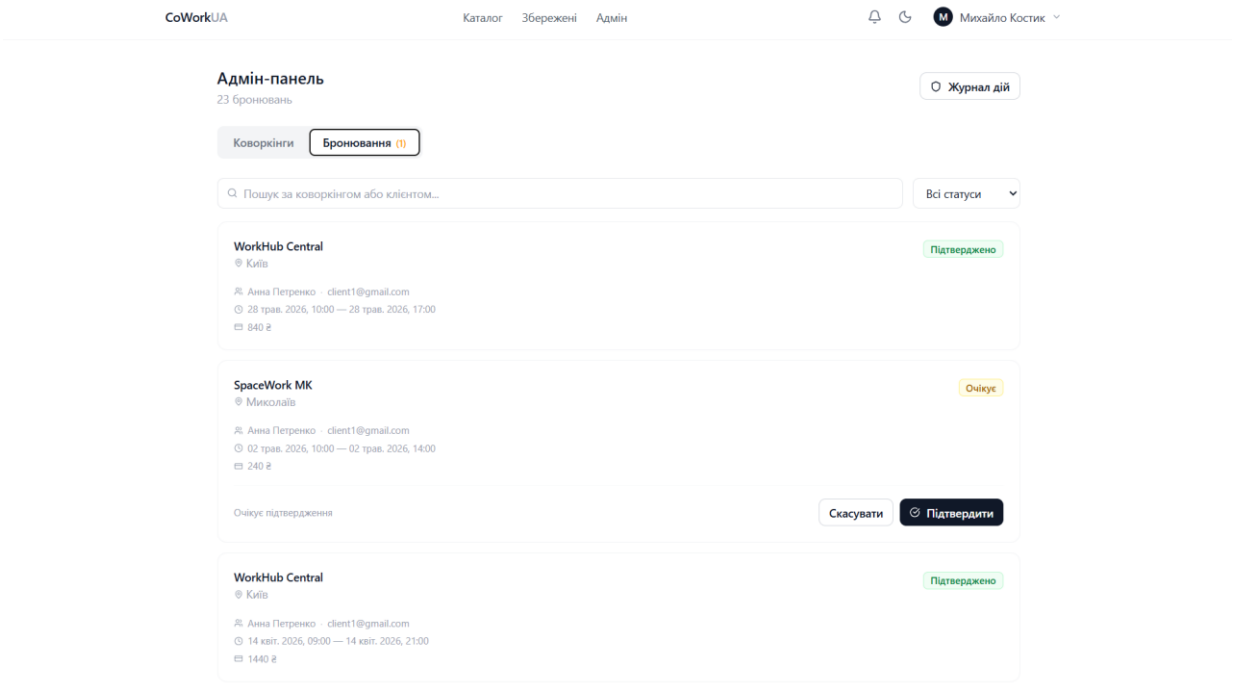

Рисунок 4.8 – Адмін-панель

Крок 2. Модерація коворкінгів. У вкладці «Коворкінги» відображаються всі коворкінги з їх статусом. Коворкінги зі статусом «На модерації» мають кнопку «Затвердити». Адміністратор перевіряє інформацію та затверджує або видаляє коворкінг.

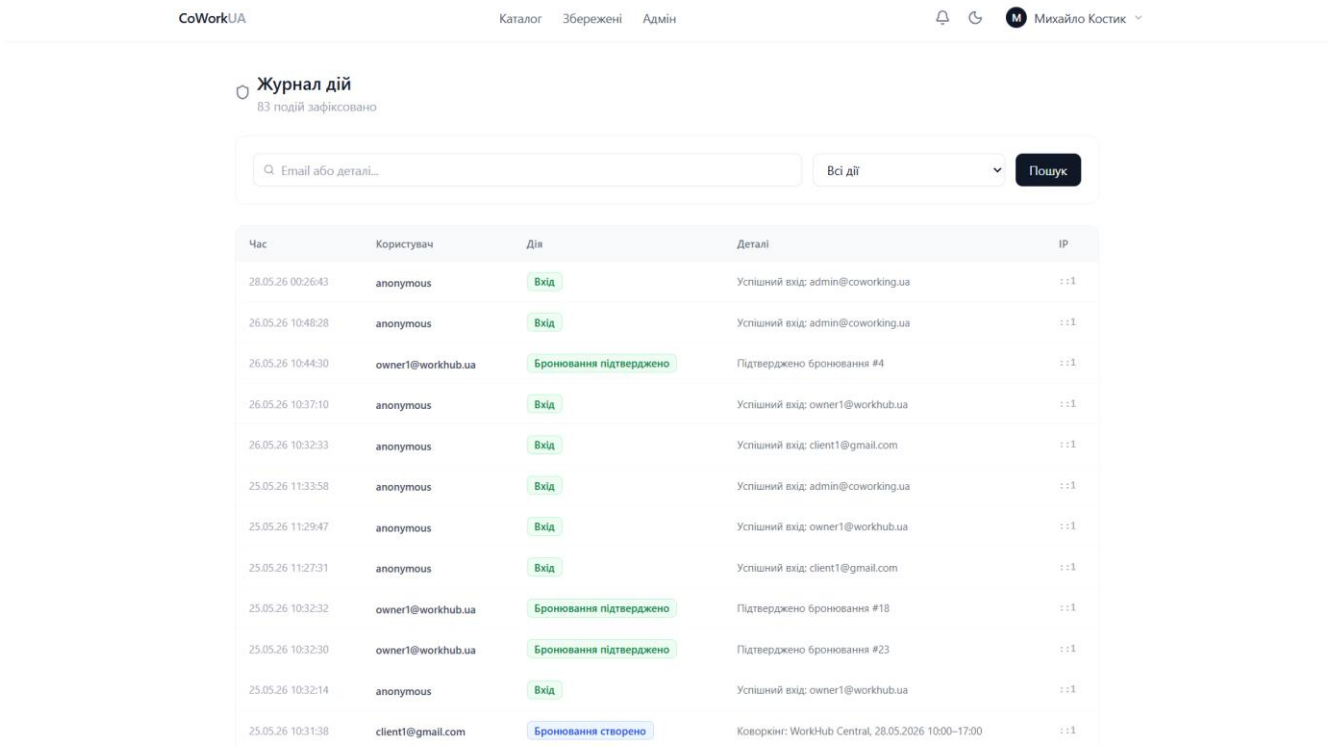

Рисунок 4.9 – Журнал дій

Крок 3. Перегляд бронювань. Вкладка «Бронювання» дозволяє переглянути всі бронювання платформи з фільтром по статусу та пошуком за назвою коворкінгу або email клієнта. Адміністратор може підтверджувати або скасовувати будь-яке бронювання.

Крок 4. Журнал дій. Перехід на /audit відкриває журнал аудиту з переліком всіх критичних дій (входи, реєстрації, бронювання, затвердження). Журнал підтримує фільтрацію за типом дії та пошук за email. Це забезпечує прозорість та можливість виявлення аномальної активності (рис. 4.9).

Висновки до розділу 4

У розділі представлено програмну реалізацію вебзастосунку та наведено специфікацію сутностей його серверної й клієнтської частин. Особливу увагу приділено моделям Booking і User, де реалізовано складні бізнес-правила та шифрування персональних даних.

Модульне тестування клієнтської частини виконано за допомогою фреймворку Vitest, а перевірку серверної логіки й захищених ендпоінтів REST API – через Swagger UI. Випробування підтвердили стабільність системи, коректність обробки сценаріїв та належний рівень покриття коду для забезпечення надійності застосунку.

Оцінка продуктивності API підтвердила його відповідність нефункціональним вимогам: час відповіді для GET-запитів не перевищує 250 мс, а для POST/PATCH – 500 мс. Найвищу затримку зафіксовано на ендпоінті авторизації, що зумовлено обчислювальною місткістю алгоритму BCrypt.

Розроблене керівництво користувача містить послідовні кроки взаємодії з інтерфейсом та скріншоти відповідних екранів для всіх чотирьох ролей: гостя, клієнта, власника та адміністратора.

ВИСНОВКИ

У кваліфікаційній бакалаврській роботі виконано комплексне науково-практичне дослідження, результатом якого є проєктування, програмна реалізація та тестування високопродуктивного вебзастосунку для пошуку та бронювання коворкінг-просторів в Україні. Створена інформаційна система успішно вирішує актуальну проблему цифровізації комерційної нерухомості та оптимізує двосторонню взаємодію між клієнтами та власниками гнучких робочих просторів в умовах розвитку дистанційної зайнятості.

У процесі виконання роботи досягнуто поставленої мети та отримано такі результати:

1) Проведено глибокий аналіз предметної області та ринку програмних рішень. Дослідження підтвердило зростання попиту на гнучкі робочі простори та виявило проблему розрізненості інформації. Аналіз існуючих аналогів показав, що більшість із них орієнтовані на B2B-сегмент та управління внутрішніми процесами, залишаючи поза увагою потреби кінцевого клієнта у зручному україномовному агрегаторі з відкритим каталогом, незалежними рейтингами та картографічним пошуком.

2) Обґрунтовано вибір технологічного стеку та сформовано вимоги. Для забезпечення високої продуктивності, безпеки та масштабованості обрано триланкову клієнт-серверну архітектуру. Рівень представлення реалізовано за допомогою React 18 та TypeScript, серверну логіку – на базі платформи ASP.NET Core 9, а збереження даних забезпечено реляційною СКБД MySQL Server 8 з використанням ORM Entity Framework Core.

3) Спроєктовано та реалізовано серверну і клієнтську частини. Побудовано надійний RESTful API із застосуванням багаторівневої архітектури. Реалізовано механізми автентифікації на базі JWT-токенів, прозоре шифрування конфіденційних даних та рольову модель доступу. Розроблено складну транзакційну логіку перевірки доступності місць у реальному часі, що унеможливлює конфлікти при бронюванні.

4) Створено інтуїтивно зрозумілий користувацький інтерфейс. Розроблено модулі пошуку з фільтрацією, інтерактивної карти, порівняння об'єктів та залишення верифікованих відгуків. Для власників просторів реалізовано особистий кабінет із панеллю управління нерухомістю, модерацією запитів та модулем графічної аналітики прибутковості.

5) Проведено комплексне тестування системи. Виконано модульне тестування фронтенд-компонентів за допомогою Vitest та перевірку ендпоінтів через Swagger UI. Результати тестування продуктивності підтвердили відповідність застосунку заявленим нефункціональним вимогам: час відповіді API на запити читання не перевищує 250 мс, а система стабільно обробляє передбачені користувацькі сценарії.

Таким чином, розроблений програмний продукт є повністю працездатним і готовим до впровадження. Практичне значення одержаних результатів полягає у створенні централізованої платформи, яка автоматизує рутинні процеси для орендодавців, підвищує прозорість ринку комерційної нерухомості та надає фрілансерам і віддаленим працівникам зручний інструмент для швидкої організації свого робочого середовища. Архітектура застосунку дозволяє легко масштабувати його в майбутньому, додаючи нові функції, такі як інтеграція платіжних систем або смарт-доступу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Berbegal-Mirabent J. What do we know about co-working spaces? Trends and challenges ahead. *Sustainability*. 2021. Vol. 13, № 3. URL: https://doi.org/10.3390/su13031416 (Accessed: 29.04.2026).

2. Kraus S., Bouncken R. B., Görmar L., González-Serrano M. H., Calabuig F. Coworking spaces and makerspaces: Mapping the state of research. *Journal of Innovation & Knowledge*. 2022. Vol. 7, № 1. URL: https://doi.org/10.1016/j.jik.2022.100161 (Accessed: 29.04.2026).

3. Coworking Spaces Market (2024 - 2030). URL: https://www.grandviewresearch.com/industry-analysis/coworking-spaces-market-report (Accessed 29.04.2026).

4. Girija S., Sharma D. R., Yeediballi T., Sriramneni C. Factors influencing the intention to use co-working spaces in emerging markets: an analytic hierarchy process approach. *Property Management*. 2024. Vol. 42, № 2. P. 235–255. URL: https://doi.org/10.1108/PM-03-2023-0026 (Accessed: 29.04.2026).

5. Do T.-T.-T., Mai-Hoang T.-B., Nguyen V.-Q., Huynh Q.-T. Query-based performance comparison of graph database and relational database. 2022. P. 375–381. URL: https://doi.org/10.1145/3568562.3568648 (Accessed: 29.04.2026).

6. Ahmad M., Bangash S. A., Rusho M. A., Halder S., Shahzadi I. Using OpenStreetMap Data for Geomarketing Insights and Business Growth. *Pioneering Approaches in Data Management*. IGI Global Scientific Publishing, 2025. P. 133–156.

7. Ahmad M. Unleashing Business Potential: Harnessing OpenStreetMap for Intelligent Growth and Sustainability. *Data-Driven Intelligent Business Sustainability*. IGI Global Scientific Publishing, 2024. P. 177–198.

8. Emmanni P. S. Comparative analysis of angular, react, and Vue. Js in single page application development. *International Journal of Science and Research*. 2023. Vol. 12, № 6. P. 2971–2974. URL: https://dx.doi.org/10.21275/SR24401230015 (Accessed: 29.04.2026).

9. Świątkowski A., Ścibior K. Comparative analysis of React, Next and Gatsby programming frameworks for creating SPA applications. *Journal of Computer Sciences Institute*. 2022. Vol. 24. C. 224–227. URL: https://doi.org/10.35784/jcsi.2972 (Accessed: 29.04.2026).

10. Dubaj S., Pańczyk B. Comparative of React and Svelte programming frameworks for creating SPA web applications. *Journal of Computer Sciences Institute*. 2022. Vol. 25. C. 345–349. URL: https://doi.org/10.35784/jcsi.3020 (Accessed: 29.04.2026).

11. Rippon C. Learn React with TypeScript. Packt Publishing, 2023. 474 p.

12. Valiveti S. S. S. Evolution of ASP. NET to ASP. NET Core: Tools, strategies, and implementation approaches. IEEE, 2025. URL: https://doi.org/10.1109/ICITEICS64870.2025.11341480 (Accessed: 29.04.2026).

13. Malavasi A. Modern Full-Stack Web Development with ASP. NET Core: A project-based guide to building web applications with ASP. NET Core 9 and JavaScript frameworks. Packt Publishing Ltd, 2025. ISBN 1789135168. 450 p.

14. Proud N. Minimal APIs in ASP. NET 9: Design, implement, and optimize robust APIs in C# with. NET 9. Packt Publishing Ltd, 2024. ISBN 1805123548. 252 p.

15. Classon I. Navigating ASP. NET Core Upgrades. *Migrating ASP. NET Microservices to ASP. NET Core 8*. Springer, 2024. P. 165–185.

16. Shethiya A. S. Load Balancing and Database Sharding Strategies in SQL Server for Large-Scale Web Applications. *Journal of Selected Topics in Academic Research*. 2025. Vol. 1, № 1. URL: https://jstarpublication.com/index.php/jstar/article/view/3 (Accessed: 29.04.2026).

17. Ilić M., Kopanja L., Zlatković D., Trajković M., Ćurguz D. Microsoft sql server and oracle: Comparative performance analysis. Vrnjačka Banja: The 7th International conference Knowledge management and informatics, June 2021. P. 33–40.

18. Пасічник С., Кунанець Н. Особливості формування front-end технологій у хмарних середовищах: клієнт–серверна архітектура та часове моделювання. 2025. Вип. 18, С. 151–162. URL: https://doi.org/10.23939/sisn2025.18.2.151 (дата звернення: 29.04.2026).

ДОДАТОК А

Лістинг коду метода Create() у BookingsController

```
[HttpPost]
[Authorize(Roles = "client")]
public async Task<IActionResult> Create([FromBody] BookingCreateDto dto)
{
    var userId = GetUserId();
    if (userId == 0) return Unauthorized();

    var coworking = await _db.Coworkings.FindAsync(dto.CoworkingId);
    if (coworking == null)
        return NotFound(new { message = "Коворкінг не знайдено" });

    if (!coworking.IsApproved)
        return BadRequest(new { message = "Коворкінг ще не затверджений" });

    if (dto.DateTo <= dto.DateFrom)
        return BadRequest(new { message = "Некоректний час бронювання" });

    var fromHour = dto.DateFrom.Hour;
    var toHour = dto.DateTo.Hour;
    var toMinute = dto.DateTo.Minute;

    if (fromHour < 8 || fromHour > 22)
        return BadRequest(new
        {
            message = "Бронювання доступне лише з 08:00 до 23:00"
        });

    if (toHour > 23 || (toHour == 23 && toMinute > 0))
        return BadRequest(new
        {
            message = "Час завершення не може бути пізніше 23:00"
        });

    var userAlreadyBooked = await _db.Bookings.AnyAsync(b =>
        b.CoworkingId == dto.CoworkingId &&
        b.UserId == userId &&
        b.Status != "cancelled" &&
        b.DateFrom < dto.DateTo &&
        b.DateTo > dto.DateFrom);

    if (userAlreadyBooked)
        return BadRequest(new
        {
            message = "Ви вже маєте активне бронювання в цьому коворкінгу на обраний час."
        });

    var overlappingCount = await _db.Bookings.CountAsync(b =>
        b.CoworkingId == dto.CoworkingId &&
        b.Status != "cancelled" &&
        b.DateFrom < dto.DateTo &&
        b.DateTo > dto.DateFrom);

    if (overlappingCount >= coworking.TotalSeats)
        return BadRequest(new
        {
            message = $"На обраний час всі місця зайняті ({coworking.TotalSeats} з {coworking.TotalSeats}). Спробуйте інший час."
```

```
        });

    var hours = (decimal)(dto.DateTo - dto.DateFrom).TotalHours;

    var booking = new Booking
    {
        CoworkingId = dto.CoworkingId,
        UserId = userId,
        DateFrom = dto.DateFrom,
        DateTo = dto.DateTo,
        TotalPrice = Math.Round(hours * coworking.PricePerHour, 2),
        Status = "pending"
    };

    _db.Bookings.Add(booking);
    await _db.SaveChangesAsync();

    var ip = HttpContext.Connection.RemoteIpAddress?.ToString();
    await _audit.LogAsync("BOOKING_CREATED", "Booking",
        entityId: booking.Id.ToString(),
        details: $"Коворкінг: {coworking.Name}, {booking.DateFrom:dd.MM.yyyy HH:mm}–{booking.DateTo:HH:mm}",
        userId: userId,
        userEmail: User.FindFirstValue(ClaimTypes.Email),
        ip: ip);

    await _notify.SendAsync(
        userId,
        "Бронювання створено",
        $"Ваше бронювання в «{coworking.Name}» на {booking.DateFrom:dd.MM.yyyy HH:mm} очікує
підтвердження.",
        "info");

    await _notify.SendToOwnerOfCoworkingAsync(
        dto.CoworkingId,
        "Нове бронювання",
        $"Новий клієнт забронював місце в «{coworking.Name}» на {booking.DateFrom:dd.MM.yyyy HH:mm}.",
        "info");

    return Ok(new
    {
        booking.Id,
        booking.Status,
        booking.TotalPrice,
        message = $"Місце заброньовано! Зайнято {overlappingCount + 1} з {coworking.TotalSeats} місць."
    });
}
```

ДОДАТОК Б

Лістинг коду тестів модуля authStore

```
/// <reference types="node" />
import { describe, it, expect, beforeEach, vi } from 'vitest'
import { useAuthStore } from '../store/authStore'
const MOCK_TOKEN =
  'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.' +
  Buffer.from(JSON.stringify({
    'http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier': '1',
    'http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress':   'test@example.
com',
    'http://schemas.microsoft.com/ws/2008/06/identity/claims/role':         'client',
    'http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname':      'Анна',
    'http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname':        'Петренко',
    exp: Math.floor(Date.now() / 1000) + 86400,
  })).toString('base64')
  .replace(/=/g, '') +
  '.signature'
describe('authStore', () => {
  beforeEach(() => {
    localStorage.clear()
    useAuthStore.setState({ token: null, user: null })
  })
  it('setToken зберігає токен у localStorage', () => {
    useAuthStore.getState().setToken(MOCK_TOKEN)
    expect(localStorage.getItem('token')).toBe(MOCK_TOKEN)
  })
  it('setToken оновлює token у стані', () => {
    useAuthStore.getState().setToken(MOCK_TOKEN)
    expect(useAuthStore.getState().token).toBe(MOCK_TOKEN)
  })
  it('setToken парсить email з JWT', () => {
    useAuthStore.getState().setToken(MOCK_TOKEN)
    expect(useAuthStore.getState().user?.email).toBe('test@example.com')
  })
  it('setToken парсить role з JWT', () => {
    useAuthStore.getState().setToken(MOCK_TOKEN)
    expect(useAuthStore.getState().user?.role).toBe('client')
  })
  it('setToken парсить firstName і lastName з JWT', () => {
    useAuthStore.getState().setToken(MOCK_TOKEN)
    const user = useAuthStore.getState().user
    expect(user?.firstName).toBe('Анна')
    expect(user?.lastName).toBe('Петренко')
  })
  it('updateUser змінює firstName без нового токена', () => {
    useAuthStore.getState().setToken(MOCK_TOKEN)
    useAuthStore.getState().updateUser({ firstName: 'Марія' })
```

```
    expect(useAuthStore.getState().user?.firstName).toBe('Марія')
  })
  it('updateUser не затирає незмінені поля', () => {
    useAuthStore.getState().setToken(MOCK_TOKEN)
    useAuthStore.getState().updateUser({ firstName: 'Нове' })
    expect(useAuthStore.getState().user?.email).toBe('test@example.com')
    expect(useAuthStore.getState().user?.role).toBe('client')
  })
  it('updateUser працює коли user === null (встановлює дані)', () => {
    useAuthStore.getState().updateUser({ firstName: 'Тест' })
    expect(useAuthStore.getState().user?.firstName).toBe('Тест')
  })
  it('logout очищає token у стані', () => {
    useAuthStore.getState().setToken(MOCK_TOKEN)
    useAuthStore.getState().logout()
    expect(useAuthStore.getState().token).toBeNull()
  })
  it('logout очищає user у стані', () => {
    useAuthStore.getState().setToken(MOCK_TOKEN)
    useAuthStore.getState().logout()
    expect(useAuthStore.getState().user).toBeNull()
  })
  it('logout видаляє токен з localStorage', () => {
    useAuthStore.getState().setToken(MOCK_TOKEN)
    useAuthStore.getState().logout()
    expect(localStorage.getItem('token')).toBeNull()
  })
  it('isAuthenticated повертає false якщо токена немає', () => {
    expect(useAuthStore.getState().isAuthenticated()).toBe(false)
  })
  it('isAuthenticated повертає true після setToken', () => {
    useAuthStore.getState().setToken(MOCK_TOKEN)
    expect(useAuthStore.getState().isAuthenticated()).toBe(true)
  })
  it('isAuthenticated повертає false після logout', () => {
    useAuthStore.getState().setToken(MOCK_TOKEN)
    useAuthStore.getState().logout()
    expect(useAuthStore.getState().isAuthenticated()).toBe(false)
  })
})
```