а**Чек-листи Checklists**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ФАХОВИЙ БІЗНЕС-КОЛЕДЖ
Циклова комісія комп’ютерної інженерії та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему

ВЕБСЕРВІС ДЛЯ ОРГАНІЗАЦІЇ ТА ОБЛІКУ БРОНЮВАНЬ НЕРУХОМОСТІ ІЗ БАГАТОРІВНЕВОЮ СИСТЕМОЮ ДОСТУПУ

Виконав: студент групи 1П-22
Спеціальності
121 Інженерія програмного забезпечення
Олександр ДУЛОВ
Керівник
Вікторія НЕМЧЕНКО

Черкаси 2026

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ФАХОВИЙ БІЗНЕС-КОЛЕДЖ

Циклова комісія комп'ютерної інженерії та інформаційних технологій

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувачка циклової комісії КІ та ІТ

_____ Наталя ФАЛЬЧЕНКО
(підпис)
«_____» _____2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Дулову Олександру Анатолійовичу

1. Тема кваліфікаційної роботи «Вебсервіс для організації та обліку бронювань нерухомості із багаторівневою системою доступу»

Керівник роботи Немченко Вікторія Юріївна, викладач другої категорії

затверджені наказом закладу фахової передвищої освіти від «06» жовтня 2025 року №84/1-у

2. Строк подання студентом кваліфікаційної роботи 02.06.2026

3. Вихідні дані до кваліфікаційної роботи мови програмування TypeScript та JavaScript, фреймворк Vue.js 3, інструмент збірки Vite, серверна платформа Node.js, фреймворк NestJS, реляційна база даних PostgreSQL, ORM Prisma, механізми автентифікації JWT та Google OAuth 2.0 (із 2FA), зовнішні сервіси Monobank API та Let's Encrypt SSL, середовище розробки Visual Studio Code, інструменти тестування Jest та Supertest.

4. Зміст кваліфікаційної роботи аналіз цифрових рішень у сферах оренди нерухомості та вебхостингу; постановка задачі та визначення вимог до вебплатформи; вибір архітектури та технологічного стеку (NestJS, Vue 3, PostgreSQL); проєктування серверної частини, бази даних та інтерфейсу; реалізація модулів оренди (сітка бронювань), керування хостингом, авторизації (OAuth, 2FA), білінгу (Monobank API) та техпідтримки; тестування функціональних сценаріїв та backend API.

5. Дата видачі завдання 15.09.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Терміни виконання етапів	Примітка про виконання
1	Вступ	14.10.2025	
2	РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ВЕБСЕРВІСІВ ДЛЯ ОБЛІКУ НЕРУХОМОСТІ	09.12.2025	
3	РОЗДІЛ 2 МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ СИСТЕМИ	10.03.2026	
4	РОЗДІЛ 3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ВЕБСЕРВІСУ	28.04.2026	
5	РОЗДІЛ 4. ТЕСТУВАННЯ ТА АНАЛІЗ ЕФЕКТИВНОСТІ ВЕБСЕРВІСУ		
6	Висновки	12.05.2026	
7	Оформлення кваліфікаційної роботи (чистовий варіант)	26.05.2026	
8	Перевірка кваліфікаційної роботи на наявність ознак плагіату (за 10 днів до захисту)	02.06.2026	
9	Подання кваліфікаційної роботи на затвердження завідувачу кафедри (за 7 днів до захисту)	10.06.2026	

Студент _____ Олександр ДУЛОВ
(підпис)

Керівник роботи _____ Вікторія НЕМЧЕНКО
(підпис)

АНОТАЦІЯ

Кваліфікаційна робота присвячена розробці веб-платформи подобової оренди нерухомості та автоматизації бронювань «OrendaUA». У роботі проаналізовано ринок оренди, виявлено проблеми адміністрування та обґрунтовано доцільність побудови спеціалізованої SaaS-системи. Особливу увагу приділено клієнт-серверній архітектурі (SPA + REST API) та моделюванню бази даних у СКБД PostgreSQL за допомогою Prisma ORM.

Програмно реалізовано бекенд-частину на базі фреймворку NestJS (TypeScript) та захищено автентифікацією JWT та Google OAuth 2.0. Інтерфейс користувача розроблено на Vue 3 з використанням Tailwind CSS та Vite. Створено ключові модулі: інтерактивний календар («календарна сітка бронювань») для візуального ведення бронювань без овербукінгу, модуль обліку фінансів, систему управління ролями персоналу (RBAC) та модуль тікетів для підтримки клієнтів.

Практичне значення полягає у створенні масштабованої платформи, яка автоматизує щоденну рутину орендодавців, спрощує фінансовий контроль та підвищує якість взаємодії з гостями.

Ключові слова: АВТОМАТИЗАЦІЯ ОРЕНДИ, ВЕБ-ПЛАТФОРМА, NESTJS, VUE 3, POSTGRESQL, КАЛЕНДАРНА СІТКА БРОНЮВАНЬ, БРОНЮВАННЯ, PRISMA ORM, REST API.

ABSTRACT

The qualification thesis is devoted to the development of a web platform for short-term property rental and booking automation called OrendaUA. The study analyzes the rental market, identifies administration challenges, and substantiates the feasibility of building a specialized SaaS system. Particular attention is paid to the client-server architecture (SPA + REST API) and database modeling in PostgreSQL DBMS using Prisma ORM.

The backend part of the system was implemented using the NestJS framework (TypeScript) and secured with JWT authentication and Google OAuth 2.0. The user interface was developed with Vue 3 using Tailwind CSS and Vite. The key modules created include an interactive booking calendar ("booking grid") for visual reservation management without overbooking, a financial accounting module, a role-based access control (RBAC) system for staff management, and a ticketing module for customer support.

The practical significance of the project lies in the creation of a scalable platform that automates the daily routine of property owners, simplifies financial control, and improves the quality of interaction with guests.

Keywords: RENTAL AUTOMATION, WEB PLATFORM, NESTJS, VUE 3, POSTGRESQL, BOOKING GRID, RESERVATIONS, PRISMA ORM, REST API.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ

ABAC Атрибутивна модель управління доступом (Attribute-Based Access Control)

ADR Середня вартість за добу (Average Daily Rate)

API Програмний інтерфейс (Application Programming Interface)

CRM Керування взаємовідносинами з клієнтами (Customer Relationship Management)

CRUD Створення, читання, оновлення та видалення (Create, Read, Update, Delete)

CSRF Підробка міжсайтових запитів (Cross-Site Request Forgery)

JWT Веб-токен JSON (JSON Web Token)

ORM Об'єктно-реляційне відображення (Object-Relational Mapping)

PMS Система управління нерухомістю (Property Management System)

RBAC Рольова модель управління доступом (Role-Based Access Control)

REST Архітектурний стиль передачі репрезентативного стану (Representational State Transfer)

RevPAR Дохід на доступний об'єкт (Revenue Per Available Room)

SaaS Програмне забезпечення як послуга (Software as a Service)

СКБД Система керування базами даних

ВСТУП

Актуальність теми. Сучасний стан глобального ринку нерухомості характеризується переходом від традиційних моделей управління до високотехнологічних екосистем. Процеси оренди, бронювання та експлуатації житлового фонду стають дедалі складнішими, що вимагає впровадження спеціалізованого програмного забезпечення класу Property Management Systems. Згідно з дослідженнями галузевих експертів, автоматизація рутинних операцій у сфері гостинності дозволяє підвищити ефективність використання номерного фонду на 25% та значно знизити ризик виникнення конфліктних ситуацій, спричинених помилками персоналу.

Проблема ефективного обліку об’єктів нерухомості та контролю за станом бронювань є критичною як для приватних орендодавців, так і для великих управляючих компаній. Використання застарілих методів фіксації даних (паперові носії, таблиці Excel) призводить до розсинхронізації інформації, що в умовах динамічної зміни попиту стає причиною фінансових збитків. Окремим викликом є організація колективної роботи. В сучасних реаліях над одним об’єктом нерухомості працює ціла група фахівців: власники, адміністратори, менеджери з продажу та обслуговуючий персонал. Відсутність чітко регламентованого доступу до функцій системи не лише створює незручності в роботі, а й несе загрозу витоку персональних даних клієнтів та комерційної таємниці.

Розв’язання цих проблем потребує розробки вебсервісів, які базуються на принципах надійної архітектури та багаторівневої системи безпеки. Використання сучасного стеку технологій, зокрема фреймворку Nest.js для серверної частини та Vue.js для клієнтської, дозволяє реалізувати складну бізнес-логіку з високою швидкістю відгуку. Впровадження моделей контролю доступу на основі ролей (RBAC) та атрибутів (ABAC) є необхідною умовою для створення професійного інструментарію управління нерухомістю. Таким чином,

розробка вебсервісу для організації та обліку бронювань із багаторівневою системою доступу є актуальною науково-прикладною задачею, що має високу практичну значущість для розвитку цифрової інфраструктури ринку послуг.

Об’єкт дослідження – процес автоматизації обліку та управління бронюваннями об’єктів нерухомості в умовах багаторівневої організаційної структури.

Предмет дослідження – моделі, методи та програмні інструменти побудови вебсервісів із розподіленим доступом (RBAC/ABAC) для координації роботи персоналу та клієнтів у сфері Property Management.

Метою кваліфікаційної роботи є розробити та дослідити функціонування вебсервісу для автоматизації процесів обліку нерухомості, який забезпечує інтерактивну візуалізацію бронювань та гнучке розмежування прав доступу користувачів для підвищення ефективності управління бізнес-процесами.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– Проаналізувати предметну область, дослідити існуючі архітектурні рішення та програмні аналоги для виявлення оптимальних стратегій автоматизації.

– Сформувати детальні функціональні та технічні вимоги до системи, зокрема щодо реалізації багаторівневої системи доступу.

– Обґрунтувати вибір інструментальних засобів розробки (Nest.js, Prisma, PostgreSQL, Vue.js), спираючись на їх продуктивність та безпекові характеристики.

– Спроектувати архітектуру системи та логічну структуру бази даних, що здатна підтримувати складні зв’язки між об’єктами, користувачами та статусами бронювань.

– Розробити програмні модулі автентифікації на основі JWT-токенів та впровадити механізми перевірки прав доступу.

- Реалізувати клієнтську частину вебсервісу з використанням реактивних компонентів для забезпечення зручного інтерфейсу (календарна сітка бронювань).

- Провести комплексне тестування розробленого ПЗ, проаналізувати його надійність, безпеку та зручність для кінцевого користувача.

Практичне значення одержаних результатів. Створений вебсервіс є готовим програмним рішенням, яке може бути інтегроване в роботу малих та середніх компаній, що займаються орендою нерухомості. Практична цінність полягає у мінімізації помилок при бронюванні, забезпеченні прозорого фінансового обліку та можливості гнучкого делегування повноважень співробітникам (від суперадміна до клінінг-менеджера). Результати дослідження можуть бути використані для подальшої розробки галузевих CRM-систем.

Апробація кваліфікаційної роботи. Результати, наведені в межах поточної роботи, були представлені на XVIII Всеукраїнській науково-практичній конференції «Тенденції розвитку ІТ-технологій в Україні».

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ВЕБСЕРВІСІВ ДЛЯ ОБЛІКУ НЕРУХОМОСТІ

1.1 Аналіз предметної області: автоматизація процесів бронювання та обліку

Сьогодні бізнес-процеси у всіх галузях економіки швидко перетворюються на цифрові. У сфері управління нерухомістю та надання послуг з оренди, як короткострокової, так і довгострокової, виникає потреба в спеціалізованих інформаційних системах. Це допомагає залишатися конкурентоспроможним та отримувати більші прибутки. Під час вивчення цієї галузі виявляється, що управління фондом нерухомості потребує роботи з величезним об'ємом даних, складними документами та постійним відстеженням стану об'єктів в режимі реального часу.

Історично, ведення записів про нерухомість та бронювання здійснювалося за допомогою паперових журналів або загальних електронних таблиць, наприклад, Microsoft Excel. Основним завданням таких систем є автоматизація рутинних операцій, пов'язаних із бронюванням, обліком клієнтів, фінансовим контролем та розподілом завдань між персоналом. Аналіз бізнес-процесів типового агентства нерухомості або компанії з управління апартаментами дозволяє виокремити такі ключові об'єкти та суб'єкти компанії, персонал розподіляється на різні ролі, кожна з яких має певний рівень доступу (наприклад, власник бізнесу, менеджер з бронювань, адміністратор, технічний працівник або кімнатар).

Клієнти – це ті, хто використовує послуги оренди нерухомості. Про них зберігається вся необхідна інформація, наприклад, контактні дані, історія бронювань та статус оплати, щоб можна було краще вести з ними справи, використовуючи системи управління відносинами з клієнтами.

Автоматизація процесів бронювання передбачає створення одного загального місця, де зберігається вся інформація, і кожна зміна стану об'єкта видобувається відразу всім уповноваженим користувачам. Найкращим способом подати стан об'єктів у часі є так званий "календарний календар бронювань" – це двовимірна таблиця, де по одній осі знаходяться об'єкти нерухомості, а по іншій – дати. Впредметної області:

Нерухомість (квартири, будинки, офіси, готельні номери) – це фізичні активи, які випускаються в оренду. Кожен об'єкт має певні ознаки: тип, місткість, адреса, ціна, поточний стан (вільний, зайнятий, на ремонті, потрібно прибирання).

Бронювання – це вид транзакції, яка показує, що певний об'єкт нерухомості зарезервовано для конкретного клієнта на певний час. Воно має різні стани: нове, підтверджене, оплачене, скасоване, завершене.

Персонал – це люди, які працюють в системі і взаємодіють з платформою у відповідності зі своїми обов'язками. В залежності від структури икористання такого інтерфейсу дозволяє менеджеру швидко дізнатися, як завантажений фонд, знайти вільні дні для нових бронювань та уникати простоїв.

Крім управління операціями, автоматизація обліку допомагає з поєднанням фінансової інформації. У традиційній системі для підрахунку прибутковості кожного об’єкта або ефективності роботи менеджера потрібно багато часу на збирання даних. Натомість інтегровані вебсервіси можуть автоматично створювати фінансові звіти, рахувати показники RevPAR (дохід за доступну кімнату) та ADR (середній щоденний тариф), що є дуже важливим для прийняття рішень про управління.

Сфера управління нерухомістю (Property Management) сьогодні вимагає впровадження спеціалізованих інформаційних систем для забезпечення конкурентоспроможності.

На основі аналізу бізнес-процесів виокремлено такі ключові компоненти:

– Об'єкти нерухомості (Properties) – фізичні активи з набором атрибутів (адреса, вартість, статус).

– Бронювання (Bookings) – транзакційна сутність, що фіксує резервування об'єкта.

– Персонал (Staff) – суб'єкти з різними рівнями доступу.

Клієнти (Guests) – кінцеві споживачі послуг, дані яких обробляються в межах CRM-підходу.

1.2 Огляд та порівняння систем-аналогів

Аналіз конкурентів сфокусовано на трьох ключових групах аналогів: великі міжнародні PMS, локальні PMS-рішення та платформи-агрегатори .

Міжнародні хмарні PMS-системи (наприклад, Cloudbeds, Smoobu). Це комплексні SaaS-рішення, що надають широкий функціонал (Channel Manager, вбудовані платіжні шлюзи), які мають такі переваги як висока функціональна повнота та глобальна підтримка. Але також у них є недоліки, які полягають у високій вартості підписки, надмірній складність інтерфейсу для малих та середніх агентств. Використання монолітних архітектур часто ускладнює інтеграцію з локальними фінансовими інструментами, а застарілі технології рендерингу призводять до низької швидкості відгуку в інтерактивних елементах, таких як календарна сітка бронювань.

Локалізовані PMS-рішення (наприклад MaxiBooking, Bnovo) – це такі системи, які орієнтовані на місцеві ринки, які пропонують базовий функціонал обліку та бронювань. Мають ряд переваг, як нижча ціна, краща адаптація під локальне законодавство та валюту. Але також мають суттєві недоліки, такі як статична рольова модель доступу (класичний RBAC), що не дозволяє реалізувати гнучкість, необхідну для мультитенантності та контролю доступу на рівні об'єкта (ABAC). Вони рідко використовують сучасний стек (Vue.js, Nest.js, PostgreSQL), що негативно впливає на продуктивність

Платформи-агрегатори (наприклад Airbnb, Booking.com) – це такі платформи, що є ключовими джерелами бронювань. Їхні вбудовані інструменти управління призначені виключно для обслуговування бронювань, отриманих через їхній канал, але вони також мають недоліки, такі як відсутність системами централізованого обліку та не мають інструментів для контролю внутрішнього персоналу. Вони не можуть запобігти овербукінгу, який виникає при одночасній роботі з кількома каналами без зовнішньої, централізованої PMS.

Cloudbeds – це комплексне хмарне SaaS-рішення, яке позиціонується як система з високою функціональною повнотою. Серед переваг Cloudbeds – глобальна підтримка, наявність Channel Manager та вбудованих платіжних шлюзів. Однак, використання переважно монолітних архітектур (Java/PHP) призводить до високої вартості підписки, надмірної складності інтерфейсу для малих та середніх агентств та низької швидкості відгуку в інтерактивних елементах, таких як календарна сітка бронювань. Система використовує класичну рольову модель контролю доступу (RBAC) з обмеженою гранулярністю.

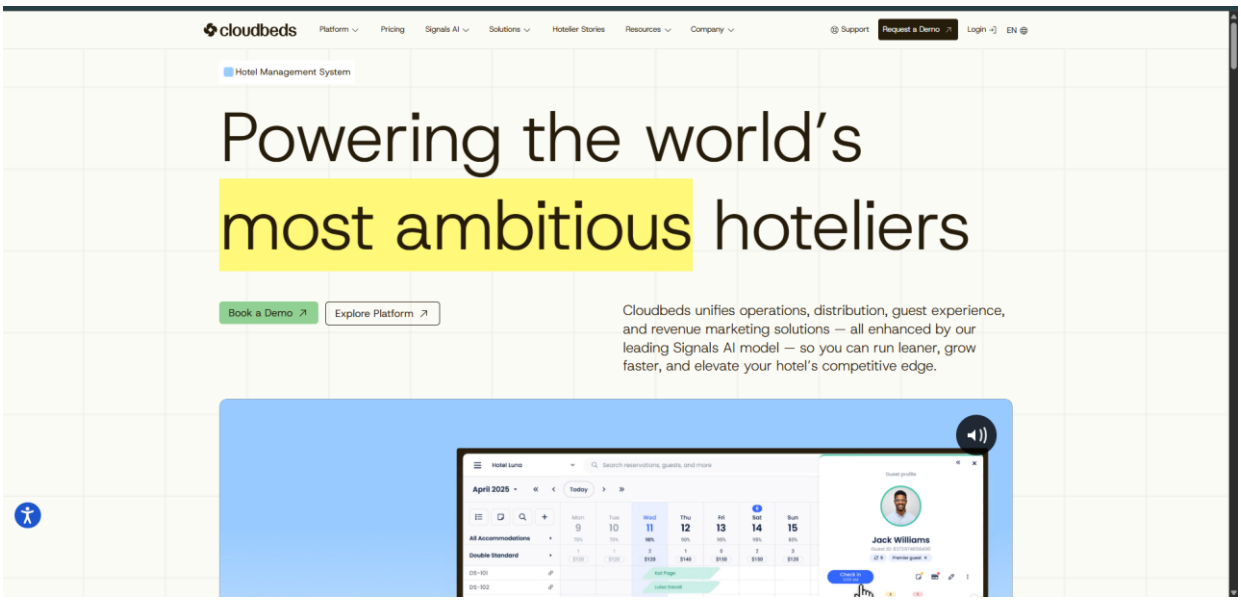

Рисунок 1.1 – Головна сторінка Cloudbeds

Booking.com є ключовим джерелом бронювань, що використовує високо адаптивну мікросервісну архітектуру та має високу адаптивність UI/UX, орієнтовану на кінцевого споживача. Однак, ця платформа не є системою централізованого обліку. Її вбудовані інструменти управління призначені виключно для обслуговування бронювань, отриманих через її канал. Booking.com не надає інструментів для контролю внутрішнього персоналу агентства та не може запобігти овербукінгу, який виникає при одночасній роботі з кількома каналами без зовнішньої, централізованої PMS.

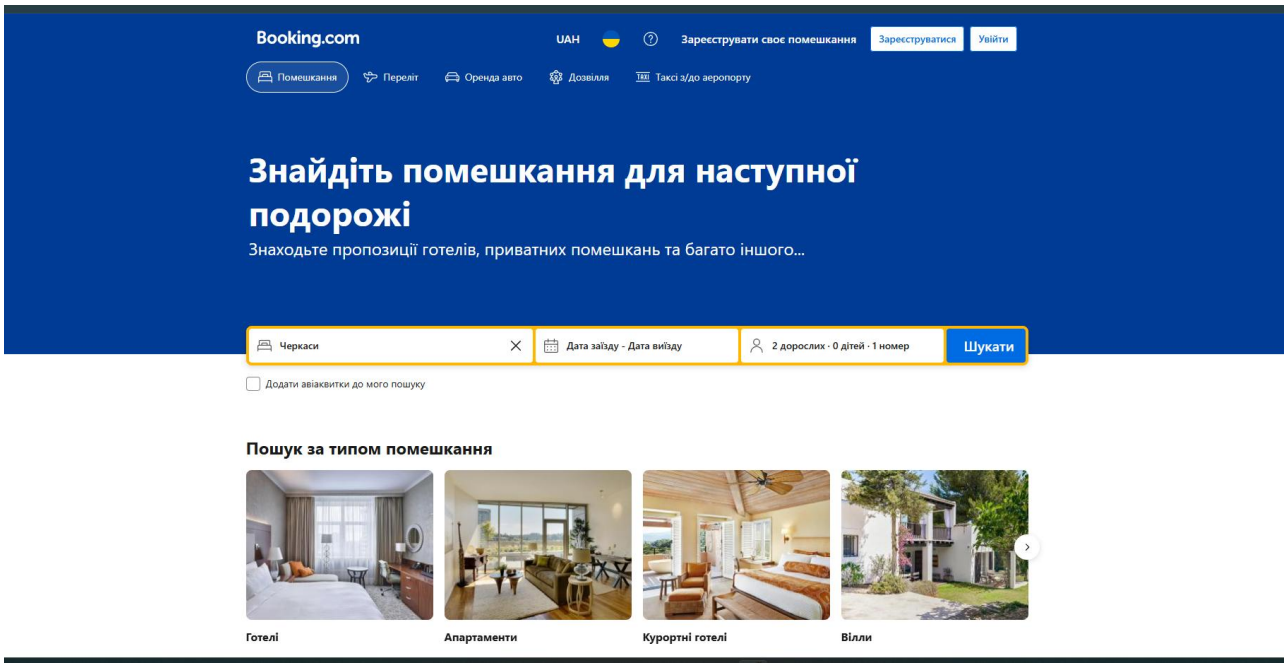

Рисунок 1.2 – Головна сторінка Booking

Guesty є популярною PMS-системою для керування об'єктами оренди та бронюваннями з різних каналів продажу. Платформа дозволяє синхронізувати календарі та зменшує ризик подвійних бронювань. Проте система розрахована переважно на великий бізнес і може бути занадто дорогою та складною для невеликих агентств нерухомості.

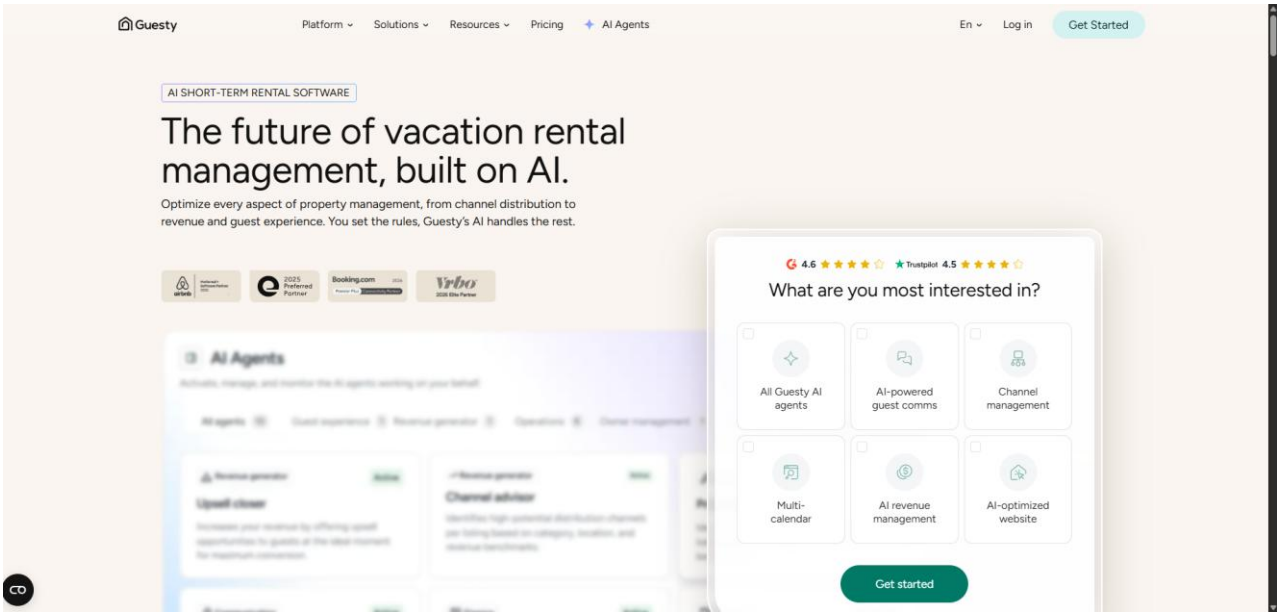

Рисунок 1.3 – Головна сторінка Guesty

Таблиця 1.1 – Порівняльний аналіз ключових аналогів

Критерій	Cloudbeds	Guesty	Booking.com
1	2	3	4
Архітектурний підхід	Переважно Legacy	Мікросервісна хмарна архітектура	Розподілена мікросервісна для високого глобального навантаження
Технологічний стек (Backend)	Java/PHP, закриті фреймворки	Сучасний API-орієнтований стек, закритий код	Багатомовний, орієнтований на масштабованість, закритий код
Система доступу	Класичний RBAC обмежена гранулярність	Розширена RBAC з розподілом ролей	Обмежена RBAC (доступ лише для партнерів-власників)

Продовження таблиці 1.1

1	2	3	4
Запобігання овербукінгу	На рівні бізнес-логіки	Централізована синхронізація календарів між каналами	На рівні каналу продажів (не запобігає колізіям при роботі з іншими каналами)
Адаптивність UI/UX	Часто застарілий, неоптимізований UI	Сучасний інтерфейс для керування великою кількістю об'єктів	Високий, орієнтований на кінцевого споживача та конверсію
Ключовий недолік	Висока вартість та складність впровадження	Висока вартість підписки та надлишкова функціональність для малого бізнесу	Не є системою централізованого обліку та не має інструментів контролю внутрішнього персоналу

1.3 Огляд сучасних архітектурних підходів у веб-розробці

Зі зростанням складності систем традиційна монолітна архітектура, де всі компоненти системи тісно пов'язані в єдиному виконуваному модулі, починає виявляти свої недоліки, зокрема складність масштабування та оновлення. Як альтернатива моноліту, у сучасній інженерії програмного забезпечення набув поширення мікросервісний підхід. Мікросервісна архітектура передбачає декомпозицію системи на набір невеликих, незалежних сервісів, кожен з яких виконує єдину бізнес-функцію і розгортається окремо. Наприклад, у контексті системи обліку нерухомості можуть бути виділені окремі сервіси для

автентифікації користувачів, обробки платежів, управління бронюваннями та генерації звітів. Основною перевагою мікросервісів є можливість незалежного масштабування найбільш навантажених вузлів системи та використання різних технологічних стеків для різних задач. Однак, варто зазначити, що мікросервісний підхід значно ускладнює інфраструктуру, тестування та моніторинг системи. Тому для проєктів на початкових етапах життєвого циклу або з відносно детермінованим навантаженням часто обирають модульний моноліт, який у майбутньому може бути рефакторений у мікросервіси

1.3.1 REST-архітектура

Для забезпечення взаємодії між клієнтом та сервером, а також між окремими сервісами всередині системи, необхідний стандартизований протокол обміну даними. Де-факто стандартом у сучасній веб-розробці є архітектурний стиль REST (Representational State Transfer), запропонований Роєм Філдінгом. REST-архітектура базується на ряді фундаментальних принципів, серед яких:

1. Клієнт-серверна модель (чітке розділення UI та зберігання даних).
2. Відсутність збереження стану на сервері (Stateless) – кожен запит від клієнта повинен містити всю інформацію, необхідну для його обробки, сервер не зберігає контекст між запитами.
3. Кешованість (Cacheability) – відповіді сервера повинні явно вказувати, чи можуть вони бути закешовані клієнтом для зменшення мережевого навантаження.
4. Уніфікований інтерфейс (Uniform Interface) – доступ до ресурсів здійснюється через стандартні HTTP-методи (GET, POST, PUT, DELETE, PATCH), де ресурси ідентифікуються за допомогою універсальних ідентифікаторів (URI).

Застосування REST API у системі обліку нерухомості дозволяє стандартизувати операції створення, читання, оновлення та видалення (CRUD) для таких сутностей, як об'єкти нерухомості, бронювання та профілі користувачів. Використання формату JSON (JavaScript Object Notation) для

передачі даних у REST-архітектурі забезпечує високу швидкість серіалізації/десеріалізації та нативну підтримку в середовищі JavaScript/TypeScript, що є оптимальним для обраного технологічного стеку. Таким чином, огляд архітектурних підходів показує, що для реалізації вебсервісу обліку нерухомості найбільш доцільним є використання клієнт-серверної архітектури з можливістю модульної декомпозиції бекенду та організацією взаємодії на основі принципів REST API. Це забезпечить необхідний баланс між продуктивністю розробки, гнучкістю системи та зручністю її подальшого масштабування.

1.4 Формування функціональних та нефункціональних вимог до вебсервісу

Процес інженерії вимог є фундаментальним етапом життєвого циклу розробки програмного забезпечення, що визначає напрямок проєктування та успішність кінцевого продукту. Відповідно до міжнародного стандарту інженерії вимог (наприклад, IEEE 29148), вимоги до програмної системи традиційно поділяються на функціональні, які описують очікувану поведінку системи, та нефункціональні, що визначають атрибути якості, такі як продуктивність, безпека та масштабованість. На основі комплексного аналізу предметної області було сформовано базовий набір функціональних вимог до цільового вебсервісу обліку нерухомості

1.4.1. Функціональні вимоги

Центральним елементом користувацького інтерфейсу є календарна сітка бронювань (Property Management Grid). Вимоги до цього модуля включають: візуалізацію об'єктів нерухомості по вертикальній осі та календарних дат по горизонтальній; відображення поточного статусу об'єкта (вільний, заброньований) за допомогою інтуїтивного кольорового кодування; можливість

створення нового бронювання шляхом виділення вільного діапазону дат безпосередньо на календарній сітці.

Вимоги до модуля бронювання передбачають програмну реалізацію механізму управління станами (спрощеної моделі скінченного автомата). Кожне бронювання повинно проходити через логічний ланцюжок статусів: "Створено" → "Підтверджено" → "Оплачено повністю" → "Завершено" або "Скасовано". Система повинна фіксувати не лише сам факт зміни статусу, але й зберігати аудиторський слід (audit trail).

Вебсервіс повинен підтримувати надійну багаторівневу систему управління доступом на основі рольової моделі контролю доступу (RBAC). До ключових вимог належить можливість створення облікових записів працівників, призначення їм наперед визначених ролей (наприклад, "Суперадмін", "Менеджер", "Покоївка") та налаштування жорсткої матриці прав доступу

До функціональних вимог аналітичного модуля входить автоматизований збір даних та розрахунок базових галузевих метрик: відсоток завантаженості (Occupancy Rate), середня вартість за добу (ADR) та дохід на доступний об'єкт (RevPAR). Платформа повинна надавати керівництву гнучкі інструменти для генерації фінансових звітів

1.4.2. Нефункціональні вимоги

Нефункціональні вимоги визначають атрибути якості системи, необхідні для успішної експлуатації в умовах високого навантаження та роботи з критично важливими даними.

1. Продуктивність та швидкодія

– Час відгуку API: Середній час відгуку на запити операцій читання (GET) не повинен перевищувати 50 мс. Час виконання транзакційних операцій запису (POST/PATCH) повинен становити 80–120 мс.

– Надійність: Система повинна підтримувати високу доступність на рівні не менше 99.9% на рік.

– Масштабованість: Архітектура Nest.js на Node.js повинна підтримувати горизонтальне масштабування серверних екземплярів, щоб забезпечити обслуговування до 5000 активних користувачів щомісяця.

2. Безпека та цілісність даних

– Контроль доступу: Повинна бути реалізована гібридна модель авторизації що забезпечує чітку ізоляцію даних між різними платформами-орендарями (Multi-tenancy).

– Автентифікація: Обов'язкове використання протоколу JWT-токенів для безстанової (stateless) авторизації. Паролі користувачів повинні зберігатися у хешованому вигляді за допомогою алгоритму bcrypt.1

– Цілісність транзакцій: Система повинна гарантувати, що не може бути створено двох записів бронювання, дати яких перетинаються для одного об'єкта. Цей захист має бути реалізований на рівні ізольованих транзакцій у СКБД PostgreSQL.

– Захист від вразливостей: Необхідно впровадити механізми захисту від типових веб-вразливостей, таких як XSS та CSRF.

3. Зручність користування Usabilit

– Адаптивність: Інтерфейс клієнтської частиниповинен коректно відображатися на всіх поширених пристроях десктоп, планшет, мобільний телефон.

– Інтуїтивність UI: Календарна сітка бронювань повинна використовувати інтуїтивне кольорове кодування для миттєвого відображення статусу об'єкта.

Висновки до розділу 1

У першому розділі було проведено комплексний аналіз предметної області вебсервісів для обліку нерухомості, який підтвердив актуальність розробки спеціалізованої Property Management System (PMS) для усунення проблем,

пов’язаних з ручним обліком та ризиком овербукінгу. В рамках дослідження було розглянуто та порівняно існуючі програмні аналоги, що дозволило сформувати унікальну конкурентну перевагу розроблюваної системи OrendaUA на основі сучасної архітектури та багаторівневої системи безпеки. На основі аналізу галузі та технологічних тенденцій було сформовано детальні функціональні вимоги до ключових модулів (календарна сітка бронювань, управління персоналом, аналітика) та критично важливі нефункціональні вимоги (продуктивність, масштабованість, безпека). З урахуванням цих вимог обґрунтовано вибір інструментального стеку: Nest.js, Vue.js, PostgreSQL та Prisma ORM, який забезпечує високу швидкість розробки, надійну цілісність даних та використання типобезпечного підходу (TypeScript, Prisma). Також було визначено, що безпека системи базуватиметься на безстановому протоколі JWT та алгоритмі хешування паролів bcrypt.

РОЗДІЛ 2 МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ СИСТЕМИ

Перехід від концептуального визначення вимог до етапу безпосереднього проєктування (аналітико-модельний рівень згідно з ЖЦ ПЗ) передбачає прийняття ключових архітектурних рішень, які визначать надійність, масштабованість та безпеку майбутнього вебсервісу.

2.1 Принципи побудови багаторівневих систем доступу

Проєктування інформаційної системи класу Property Management System висуває підвищені вимоги до інформаційної безпеки, оскільки вебсервіс зберігає і обробляє важливі дані. Комерційну таємницю агентств нерухомості, фінансові потоки від бронювань, а також персональні дані клієнтів.

Фундаментальним принципом побудови системи безпеки в таких умовах є принцип найменших привілеїв. Згідно з цим принципом, кожному суб'єкту системи надається мінімально необхідний набір прав доступу, достатній виключно для виконання його поточних посадових обов'язків. Для реалізації цього принципу сучасна інженерія програмного забезпечення пропонує дві домінуючі моделі контролю доступу: рольову та атрибутивну.

2.1.1 Рольова модель управління доступом

Модель RBAC, передбачає розподіл прав доступу за ролями. Кожному користувачу призначається своя роль, яка визначає доступні для нього можливості в системі.

Основними складовими моделі є користувачі, ролі та права доступу. Перевагою такого підходу є простота керування доступом, оскільки права надаються ролям, а не кожному користувачу окремо. Водночас у системах із

великою кількістю користувачів може виникати потреба у створенні значної кількості ролей, що ускладнює їх подальше адміністрування.

2.1.2 Атрибутивна модель управління доступом

Статична рольова модель часто обмежує, тому замість неї беруть ABAC, який керує доступом. Система вирішує, пускати когось чи ні, аналізуючи купу деталей Зазвичай перевіряють три речі: хто просить доступ до чого саме та за яких умов. Єдиний серйозний мінус - усе це вимагає багато ресурсів і може гальмувати систему, бо обчислення тут доволі важкі.

2.1.3 Гібридна модель

У кваліфікаційній роботі обґрунтовано доцільність використання гібридної моделі контролю доступу, архітектура якої розділена на два логічні рівні. На першому етапі – рівні маршрутизатора API через Middleware або Guards - застосовується модель RBAC. Оскільки інформація про роль зберігається в JWT-токені, система робить це не звертаючись до бази даних. На другому етапі, який розгортається на рівні бізнес-логіки у Services, діє спрощений підхід, що відповідає моделі.

На рис 2.1 відображено алгоритм послідовної перевірки прав доступу, де первинним фільтром виступає перевірка валідності токена та ролі, а вторинним – перевірка атрибутів належності ресурсу на рівні бази даних).

Таблиця 2.1 – Порівняльний аналіз моделей доступу для вебсервісу PMS

Критерій	RBAC	ABAC	Гібридна модель
Гранулярність	1	4	3
Складність	1	4	2
Продуктивність	4	1	4
Role Explosion	4	1	1

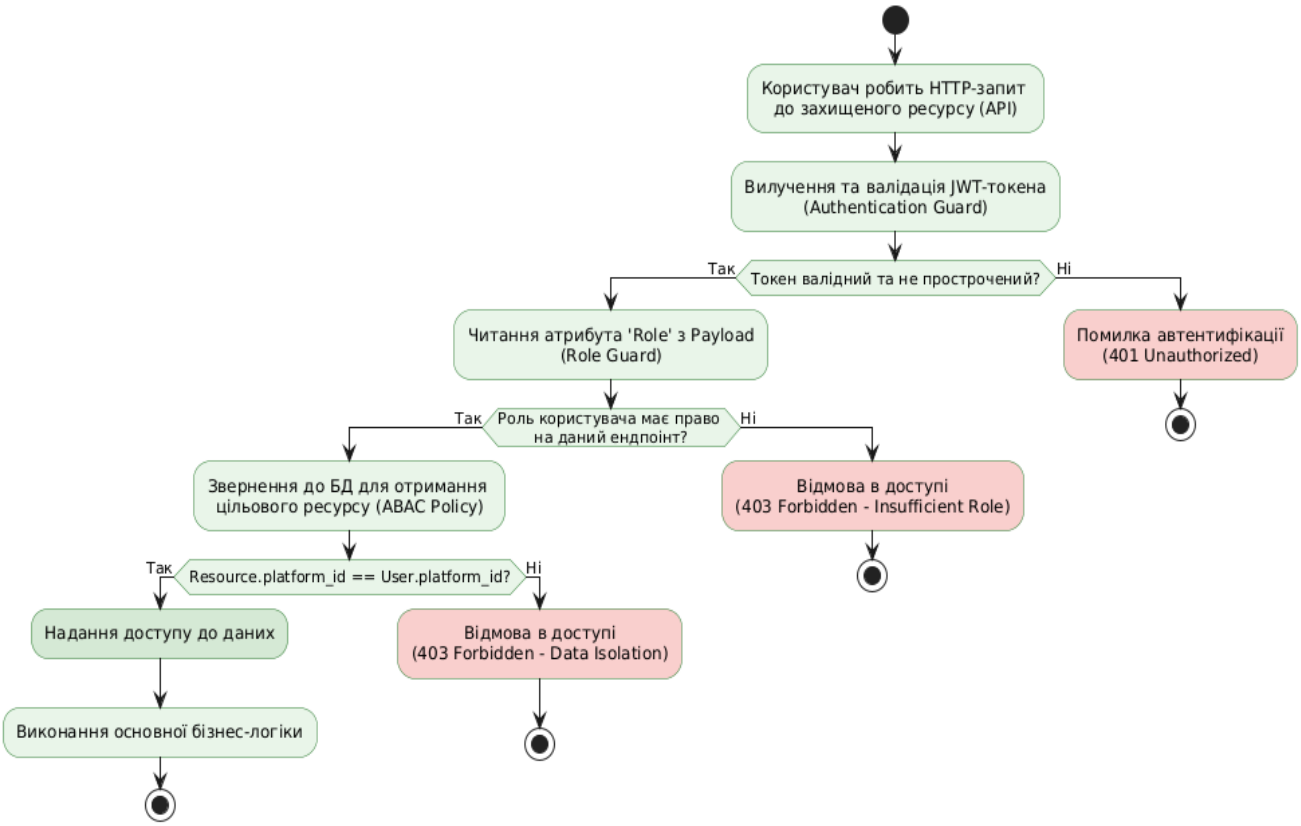

Рисунок 2.1 – Блок-схема алгоритму авторизації за гібридною моделлю (RBAC + ABAC)

2.2 Обґрунтування вибору технологічного стеку для бекенду

2.2.1 Середовище виконання Node.js

Для реалізації серверної логіки обрано середовище виконання Node.js. У традиційних багатопотокових серверах, кожен новий запит створює або займає окремий потік операційної системи, що призводить до значних витрат оперативної пам'яті при тисячах одночасних з'єднань. Натомість Node.js використовує єдиний потік та цикл подій, що робить його ідеальним рішенням для розробки застосунків реального часу, таких як календарна сітка бронювань, де необхідна миттєва синхронізація станів між різними клієнтами.

2.2.2 Фреймворк Nest.js

Незважаючи на популярність мінімалістичних фреймворків, для розробки корпоративних систем їх використання часто призводить до неструктурованого коду, через відсутність строгих архітектурних конвенцій. Тому як основний фреймворк обрано Nest.js. Цей прогресивний фреймворк побудований поверх Express і надає готову архітектуру, дотримується об'єктно-орієнтованого програмування. Nest.js активно використовує TypeScript та реалізує контейнер управління для впровадження залежностей

База даних PostgreSQL. Враховуючи, що система оперує фінансовими показниками та структурованими даними зі складними зв'язками, використання NoSQL баз даних є недоцільним через високий ризик порушення цілісності даних . Оптимальним вибором є об'єктно-реляційна система управління базами даних PostgreSQL. Вона забезпечує повну відповідність принципам ACID (Атомарність, Узгодженість, Ізольованість, Довговічність). Це критично важливо для реалізації механізму транзакцій під час створення бронювання: якщо в процесі розрахунку вартості або запису гостя виникає помилка, уся транзакція відкочується, що унеможливлює появу "фантомних" або частково оплачених бронювань.

ORM-бібліотека Prisma. Для взаємодії програмного коду з PostgreSQL обрано Prisma ORM. На відміну від традиційних інструментів, Prisma використовує декларативну схему даних на основі якої автоматично генерує строго типізований клієнт. Це гарантує наскрізну безпеку типів, будь-які зміни в структурі бази даних миттєво підсвічуються як помилки компіляції в коді Nest.js, що унеможливлює виникнення раптових збоїв під час виконання.

На рис. 2.2 продемонстровано послідовність викликів між клієнтом, контролерами та базою даних під час генерації токена доступу та його подальшого використання для проходження через захисники маршрутів.

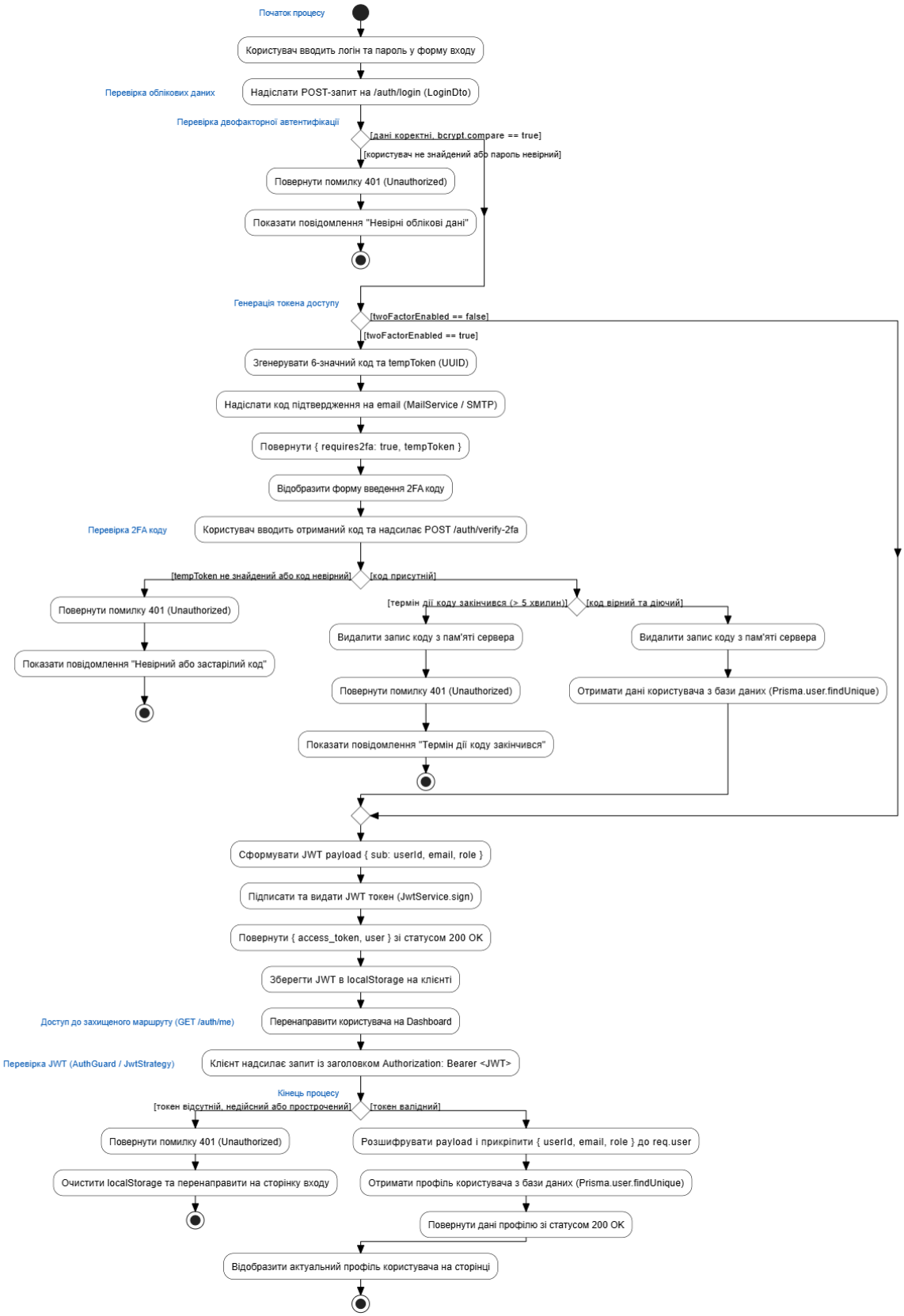

Рисунок 2.2 UML-діаграма послідовності: Механізм автентифікації та захисту маршрутів на основі JWT. (Опис рисунка для тексту

2.3 Обґрунтування вибору технологічного стеку для фронтенду

Для розробки клієнтської частини інформаційної системи OrendaUA було обрано сучасний технологічний стек у складі фреймворку Vue.js 3, інструменту збірки Vite, стилістичного каркаса Tailwind CSS 4 та мови програмування TypeScript. Головним критерієм вибору стала необхідність створення швидкого, інтерактивного та легко масштабованого односторінкового додатка із високим рівнем швидкодії інтерфейсу.

Вибір Vue.js зумовлений його високою реактивністю, модульною архітектурою та низьким порогом входження порівняно з аналогами React чи Angular. Компонентний підхід фреймворку є оптимальним для реалізації складних динамічних елементів, таких як інтерактивна сітка бронювань та кабінет адміністрування середовища. Використання інструменту збірки Vite замість класичного Webpack дозволило значно прискорити процес перезавантаження модулів під час розробки та забезпечити швидке завантаження готового додатка.

Для стилізації інтерфейсу було обрано Tailwind CSS 4. Завдяки концепції utility-first цей фреймворк дозволяє швидко створювати адаптивні та сучасні інтерфейси без необхідності написання громіздких файлів стилів, що суттєво зменшує кінцевий розмір та спрощує підтримку коду. Застосування мови TypeScript забезпечило строгу типізацію даних на клієнті, що дозволило виявляти помилки на етапі компіляції та спростило інтеграцію з типізованими API-запитами бекенду. Навігацію між розділами системи без перезавантаження сторінок забезпечує бібліотека Vue Router, яка також відповідає за захист закритих маршрутів кабінету користувача на клієнтському рівні.

2.4 Аналіз протоколів та методів безпеки

Для захисту інформаційної системи OrendaUA від несанкціонованого доступу та забезпечення цілісності даних реалізовано комплекс засобів безпеки. Автентифікація користувачів здійснюється за допомогою підписаних токенів JWT та інтеграції з Google OAuth 2.0. Додатковий рівень захисту забезпечує двофакторна автентифікація з надсиланням одноразових кодів на електронну пошту користувача. Облікові записи захищено криптографічним хешуванням паролів за допомогою алгоритму Bcrypt із додаванням солі.

Керування доступом базується на рольовій моделі, яка розділяє звичайних користувачів та адміністраторів, а також на списках контролю доступу для гнучкого налаштування прав персоналу готелів. На мережевому рівні весь обмін даними зашифровано через протокол HTTPS, а захист від SQL-ін'єкцій та міжсайтового підроблення запитів реалізовано через параметризацію запитів у Prisma ORM та налаштування політик CORS. Фінансові операції поповнення балансу верифікуються виключно на стороні сервера шляхом прямих запитів до Monobank API, що унеможливлює підробку транзакцій на клієнті.

2.5 Проєктування архітектури системи та логічної структури бази даних

Ефективність функціонування будь-якої інформаційної системи класу Property Management System залежить від оптимальності та надійності спроєктованої структури бази даних. Логічна схема бази даних визначає, як інформація зберігається, оновлюється та пов'язується між собою, гарантуючи при цьому її цілісність, консистентність та відсутність надлишковості. Для розроблення програмного ядра було обрано об'єктно-реляційну систему управління базами даних PostgreSQL. Вибір реляційної моделі зумовлений жорсткою структурою фінансових та операційних даних системи управління

нерухомістю, де критично важливим є дотримання принципів ACID (атомарність, узгодженість, ізольованість, довговічність) для запобігання втраті фінансових записів та виникненню подвійних бронювань.

З огляду на специфіку розроблюваного вебсервісу як платформи формату Software as a Service, база даних спроєктована за мультитенантною архітектурою з логічною ізоляцією даних на рівні ідентифікаторів орендаря. Логічна структура бази даних приведена до третьої нормальної форми та складається з набору взаємопов'язаних відношень .

Відношення користувачів системи формується на базі таблиці User. Вона є єдиною точкою автентифікації та зберігає глобальні облікові записи. Окрім стандартних ідентифікаційних даних, таких як унікальна електронна пошта, ім'я та хеш пароля, таблиця містить опціональне поле googleId для підтримки протоколу OAuth 2.0. Оскільки система працює за комерційною моделлю, до сутності користувача додано білінгові та лімітуючі атрибути: plan тип активної підписки, наприклад free або start, maxObjects та maxPlatforms, а також balance для фіксації поточного фінансового стану клієнта. Поле role визначає глобальні права в системі.

Для реалізації концепції віртуального агентства нерухомості використовується відношення Platform. Кожна платформа ідентифікується унікальним первинним ключем id у форматі UUID. Використання UUID замість автоінкрементних цілих чисел унеможливлює вразливості типу IDOR та приховує від конкурентів загальну кількість компаній у системі. Платформа має назву, лічильник працівників, статус активності та дату завершення підписки. Кожна платформа через зовнішній ключ userId жорстко пов'язана з користувачем, який її створив.

Управління персоналом та правами доступу всередині конкретної платформи здійснюється через відношення PlatformEmployee. Ця таблиця реалізує відношення багато-до-багатьох між таблицями User та Platform.

Операційний блок системи представлений відношеннями об'єктів нерухомості та транзакцій резервування. Відношення Object описує фізичні

одиниці квартири, номери, будинки. Воно містить локальний ідентифікатор, прив'язку до компанії platformId, коротку назву, повну назву, текстовий опис та масив посилань на завантажені фотографі.

Головною сутністю вебсервісу є відношення бронювання. Запис ідентифікує конкретну подію оренди і містить часові рамки startDate та endDate для забезпечення мілісекундної точності. Поле type розрізняє статус резервації наприклад, booking – попередня бронь, checkin – клієнт фактично заселився. Інформація про гостя зберігається у полях guestName, guestPhone та guestsCount. Фінансові показники фіксуються в атрибутах price та deposit.

Додатково в системі передбачено підсистему клієнтської підтримки, яка складається з відношень Ticket та TicketMessage, що дозволяє власникам агентств створювати звернення до технічного відділу платформи із зазначенням пріоритетів та збереженням історії листування. Усі таблиці в базі даних автоматично фіксують час створення та оновлення записів, що є обов'язковою вимогою для ведення журналів аудиту.

База даних системи спроєктована з дотриманням принципів нормалізації для забезпечення цілісності даних та мінімізації надлишковості.

Перша нормальна форма: Досягнута завдяки атомарності даних. Наприклад, у таблиці Booking дані про гостя так як ім'я, телефон, кількість розділені на окремі атрибути, а не зберігаються у складеному форматі. Винятком є поле photos у таблиці Object, яке зберігається як масив посилань, що відповідає функціоналу PostgreSQL для неатомарних, але необхідних для клієнта даних.

Друга нормальна форма: Забезпечена тим, що всі неключові атрибути повністю залежать від первинного ключа об'єкта. У системі відсутні складені ключі, що виключає часткові функціональні залежності.

Третя нормальна форма досягнута шляхом усунення транзитивних залежностей. Наприклад, замість того, щоб зберігати повні дані про власника у таблиці Platform, ми зберігаємо лише зовнішній ключ userId, який є посиланням на таблицю User. Аналогічно, уся інформація про права доступу персоналу

винесена у зв'язкову таблицю PlatformEmployee, що унеможливлює дублювання даних про права, якщо користувач змінює платформу або роль.

Дотримання 3NF гарантує, що операції оновлення, видалення та вставки даних не призведуть до порушення цілісності, що є особливо важливим при роботі з фінансовими записами та статусами бронювань.

Діаграма компонентів (рис. 2.4) демонструє взаємодію основних технологічних блоків системи: клієнтського Single Page Application, розробленого на Vue.js, серверного Nest.js REST API, який реалізує бізнес-логіку, та СКБД PostgreSQL для зберігання даних. Взаємодія між цими блоками захищена механізмами JWT-токенів та Google OAuth 2.0.

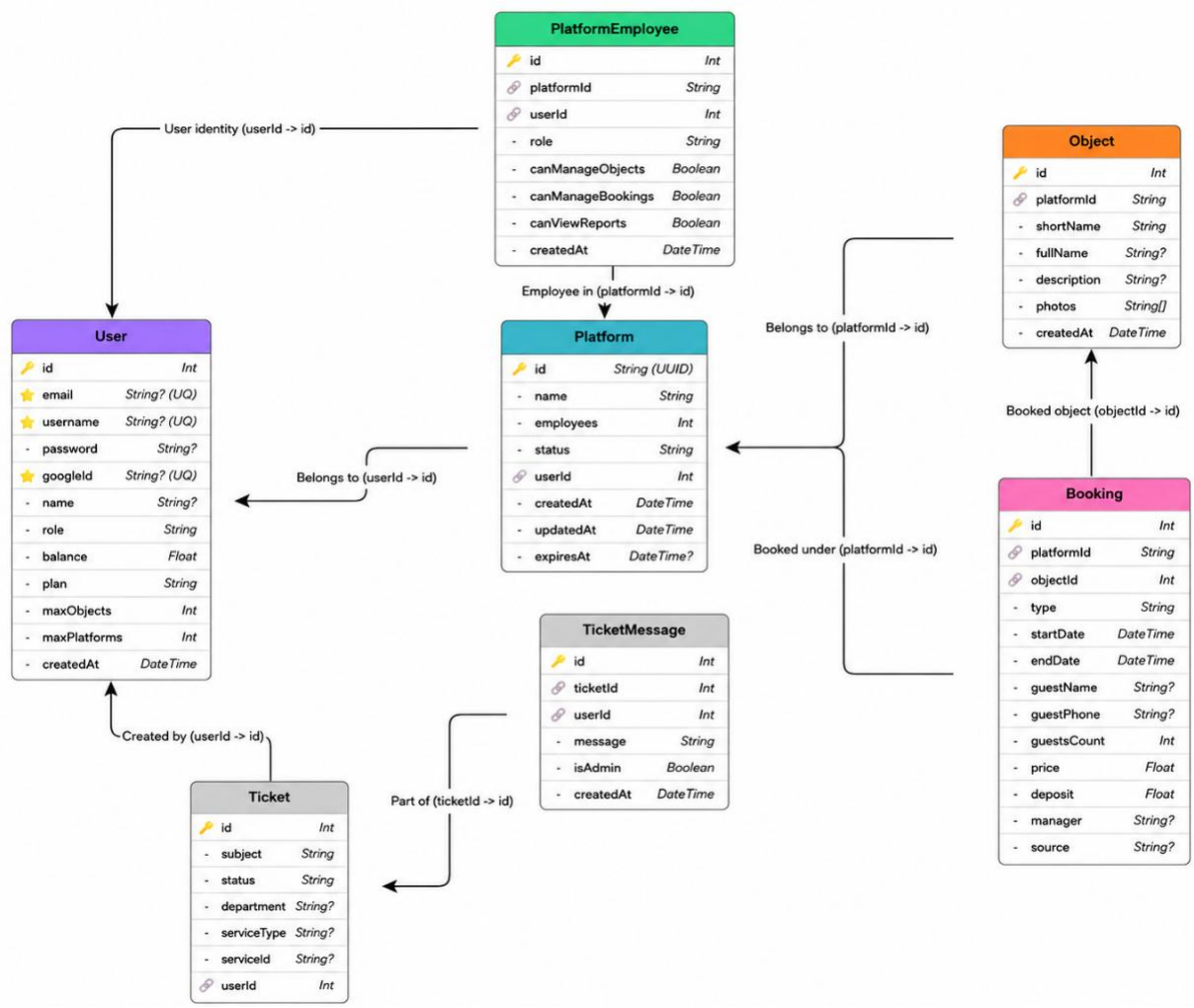

Рисунок 2.3 – ER-діаграма логічної структури бази даних системи

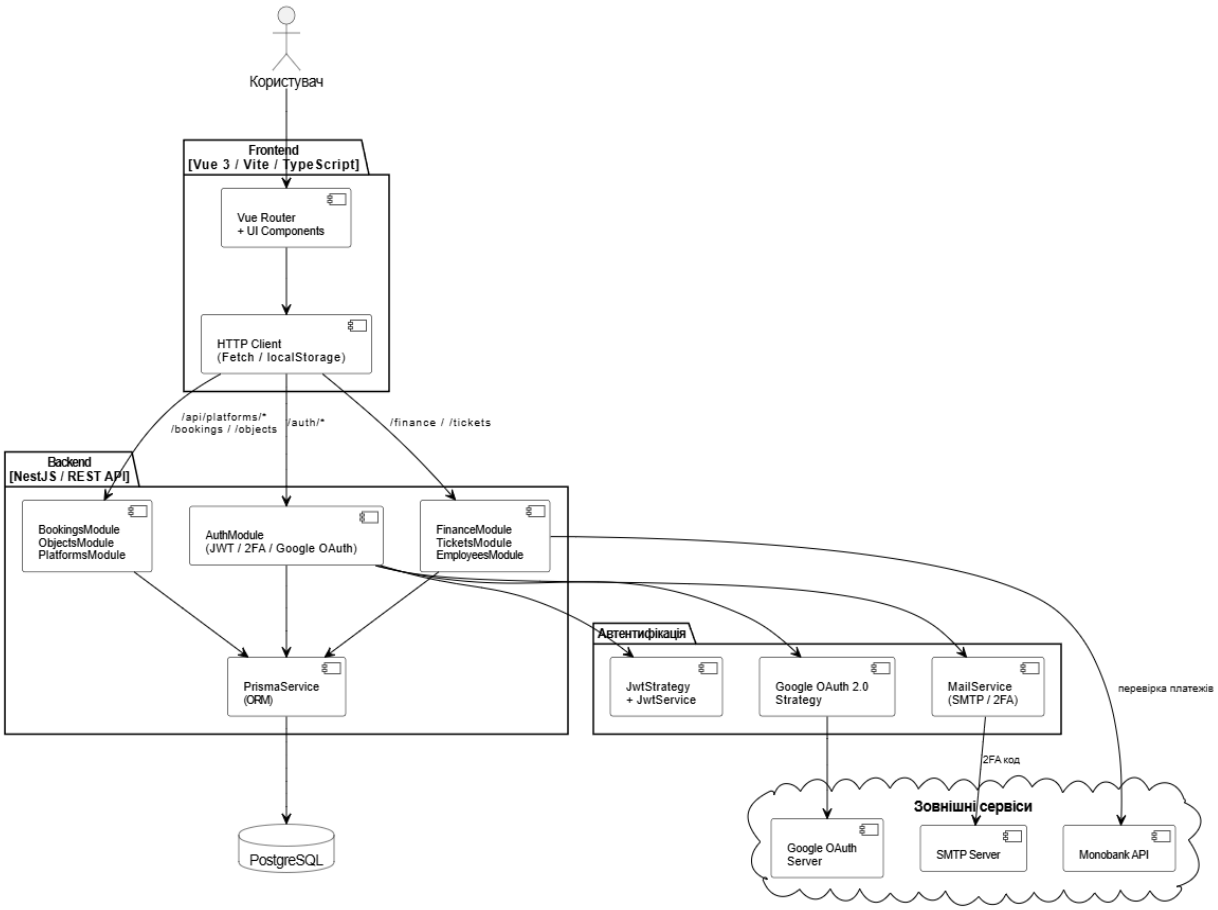

Рисунок 2.4 – Діаграма компонентів вебсервісу

2.6 Розробка архітектури багаторівневого доступу

Проєктування підсистеми розмежування прав у мультитенантній платформі вимагає відходу від класичногорольового доступу на користь складеної ієрархічної архітектури. Вона має враховувати, що один і той самий користувач може бути власником одного агентства нерухомості, і водночас бути запрошеним як звичайний менеджер до іншого. Відповідно, розроблена архітектура доступу поділяється на два незалежних рівні авторизації: глобальний рівень системи та локальний рівень орендаря.

Глобальний рівень визначає права користувача щодо самої платформи і контролюється виключно полем role у таблиці User. Цей рівень має обмежену кількість градацій. Роль суперадміністратора надає абсолютний доступ до панелі управління сервісом, дозволяючи керувати підписками агентств, переглядати

загальну статистику навантаження, поповнювати баланс клієнтів та опрацьовувати тікети технічної підтримки. Проте суперадміністратор не має доступу до комерційної інформації клієнтів персональних даних гостей або фінансових звітів конкретних агентств. Стандартна роль user надається при реєстрації та дозволяє створювати власні платформи в межах лімітів обраного тарифного плану.

Локальний рівень авторизації активується, коли користувач взаємодіє з даними конкретного агентства нерухомості. На цьому рівні глобальна роль user не відіграє ролі. Управління доступом реалізується за допомогою гібридної моделі на основі атрибутів, імплементованої через булеві прапорці у таблиці PlatformEmployee.

Замість жорсткого кодування текстових посад права збираються як конструктор за допомогою трьох основних прапорців. Атрибут canManageObjects надає право на створення, редагування параметрів та видалення об'єктів нерухомості. Атрибут canManageBookings є основним операційним правом, що дозволяє створювати та скасовувати резервації, працювати з інтерактивною шахматкою та вносити фінансові дані гостя. Атрибут canViewReports відкриває доступ до агрегованої фінансової аналітики та статистики завантаженості фонду.

Комбінація цих прапорців дозволяє формувати будь-які посадові ролі:

Власник - особа, яка створила платформу. Автоматично отримує всі прапорці у стані true та має виключне право запрошувати нових працівників і змінювати налаштування компанії.

Менеджер з бронювань - співробітник офісу. Має прапорець canManageBookings = true, але canManageObjects = false. Він повноцінно веде календар, реєструє гостей та фіксує передоплати, але не може видалити квартиру з бази чи змінити її базову ціну.

Прибиральник / Технічний персонал: Працівник із мінімальними привілеями. Усі його прапорці встановлені у false. Програмний інтерфейс обмежує його бачення до режиму тільки читання інтерактивної сітки.

Прибиральник бачить, які номери звільняються і потребують обслуговування, але інтерфейс приховує від нього фінансову інформацію, вартість проживання та номери телефонів клієнтів.

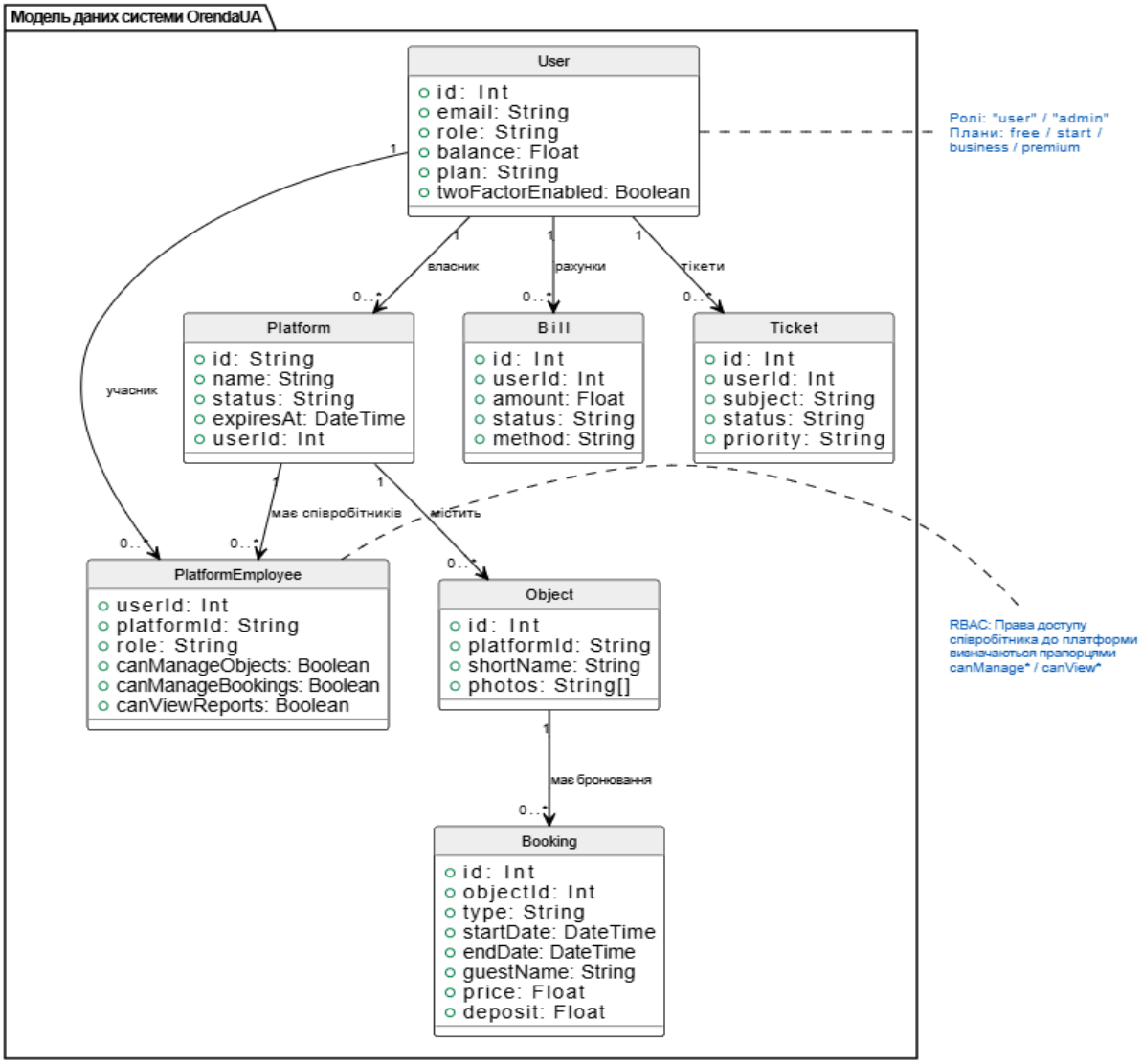

Рисунок 2.5 – UML-діаграма класів ключових об’єктів системи OrendaUA.

Програмна реалізація цієї моделі на стороні серверного фреймворку Nest.js виконується за допомогою перехоплювачів маршрутів Guards. Після успішної автентифікації сервер генерує JSON Web Token, в корисне навантаження, якого шифрується інформація про ідентифікатор користувача. Під час будь-якого HTTP-запиту до захищених ресурсів API, витягує ідентифікатор платформи з тіла запиту, робить запит до БД і перевіряє відповідний булевий прапорець .

Якщо умова не виконується, сервер негайно перериває обробку та повертає помилку HTTP 403 Forbidden.

2.7 Моделювання бізнес-процесів

Функціонування платформи обліку нерухомості базується на виконанні ряду взаємопов'язаних бізнес-процесів, які опрацьовують значні масиви даних у реальному часі. Оскільки розроблювана система належить до класу B2B і є інструментом для внутрішнього управління, її безпосередніми користувачами є виключно співробітники агентств нерухомості власники, менеджери, адміністратори. Кінцеві споживачі послуг не взаємодіють із системою безпосередньо усі дані про їхнє проживання вносяться уповноваженим персоналом.

Найбільш критичним з алгоритмічної та фінансової точок зору є бізнес-процес фіксації оренди – створення попередньої броні або безпосереднього заселення. Головним ризиком у багатокористувацьких системах, де кілька менеджерів одночасно обробляють телефонні дзвінки або запити з агрегаторів, є виникнення стану гонки. Це може призвести до овербукінгу- помилкової фіксації двох різних гостей на один і той самий об'єкт в дати, що перетинаються.

Алгоритм обробки резервації та захисту від колізій ініціюється виключно авторизованим працівником. Детальний покроковий сценарій цього бізнес-процесу:

Ініціалізація запиту менеджером: Менеджер отримує запит від клієнта наприклад, через телефонний дзвінок, месенджер або сповіщення з порталу Booking.com. Менеджер відкриває календарну сітку бронювань в системі, візуально знаходить вільну квартиру на запитані дати та виділяє цей діапазон курсором.

Заповнення параметрів оренди: На екрані з'являється модальне вікно, де менеджер вносить деталі: обирає тип операції, вводить ім'я та телефон гостя.

Також менеджер вказує джерело ліда наприклад, напряму, фіксує своє ім'я у полі manager, визначає загальну вартість проживання та суму внесеного авансу.

Локальна валідація: Фронтенд-застосунок виконує базові перевірки що дата виїзду є пізнішою за дату заїзду і відправляє HTTP-запит до серверного API. До заголовка запиту автоматично додається JWT-токен менеджера.

Автентифікація та перевірка повноважень: Сервер декодує токен, ідентифікує користувача та перевіряє в таблиці, чи має цей співробітник прапорець для обраної платформи. Якщо прав немає, повертається статус 403 Forbidden.

Оновлення інтерфейсу: Сервер повертає успішну відповідь 201 Created з даними нового запису. Календарна сітка бронювань миттєво відмальовує новий кольоровий блок оренди наприклад, жовтий для booking, зелений для checkin.

На рис. 2.6 відображено послідовність взаємодії між менеджером агентства, клієнтським додатком та бекендом під час створення запису про оренду, з деталізацією механізму блокування подвійних бронювань на рівні транзакцій СУБД.

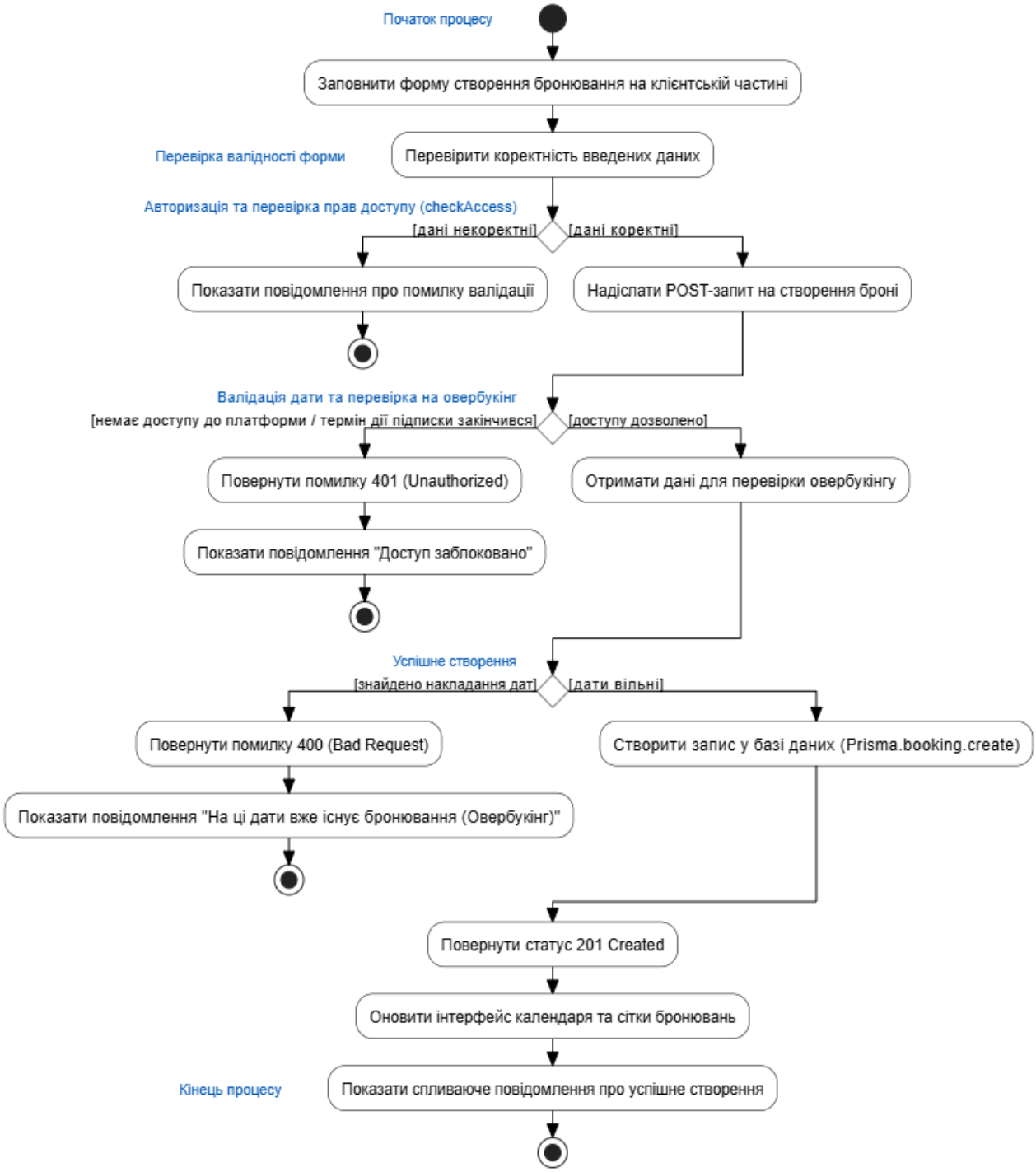

Рисунок 2.6 – UML-діаграма діяльностей: Бізнес-процес створення броні/заселення менеджером

2.8 Проєктування структури REST API та основних ендпоінтів

Для забезпечення взаємодії між ізольованим клієнтським застосунком та серверною частиною було розроблено програмний інтерфейс на основі архітектурного стилю REST. Цей підхід розглядає кожну сутність бази даних як незалежний ресурс, доступ до якого здійснюється через стандартизовані HTTP-методи: отримання, створення, оновлення та видалення. Форматом обміну даними обрано JSON, що гарантує високу швидкість десеріалізації в середовищі JavaScript/TypeScript.

Структура REST API логічно поділена на кілька ключових модулів, які відповідають структурі спроєктованої бази даних PostgreSQL:

Модуль автентифікації та користувачів (Users & Auth) Цей модуль є єдиною відкритою точкою входу в систему.

Метод POST /api/auth/register відповідає за створення нового облікового запису власника в таблиці User.

Метод POST /api/auth/login здійснює валідацію пошти та пароля за допомогою bcrypt)і повертає згенерований JWT-токен.

Метод GET /api/users/me повертає профіль поточного авторизованого користувача, включаючи його баланс, тарифний план та обмеження за кількістю об'єктів.

Метод POST /api/platforms дозволяє створити нове агентство, за умови, що користувач не перевищив ліміт.

Метод PATCH /api/platforms/:platformId/employees/:userId дозволяє власнику динамічно змінювати права своїх менеджерів.

Модуль об'єктів нерухомості. Відповідає за управління номерним фондом квартир або готельних номерів.

Метод GET /api/platforms/:platformId/objects повертає список усіх об'єктів агентства. Дані містять коротку назву, опис та масив.

Метод POST /api/platforms/:platformId/objects дозволяє додати нову квартиру, якщо ліміт не вичерпано.

Метод POST /api/platforms/:platformId/bookings ініціює транзакцію створення броні або заселення. У тілі запиту передаються параметри гостя, фінансові дані та джерело.

Метод PATCH /api/platforms/:platformId/bookings/:bookingId дозволяє змінити статус наприклад, перевести попередню бронь booking у статус фактичного проживання checkin після приїзду клієнта.

Модуль технічної підтримки. Для забезпечення якісного обслуговування B2B-клієнтів реалізовано внутрішній Service Desk.

Метод GET /api/tickets повертає список звернень поточного користувача.

Метод POST /api/tickets створює новий запит.

Метод POST /api/tickets/:ticketId/messages дозволяє додавати нові повідомлення до існуючого тікета.

Усі вхідні дані, які передаються на перелічені ендпоінти, проходять сувору валідацію за допомогою Data Transfer Objects та бібліотеки class-validator, що унеможливлює SQL-ін'єкції та гарантує консистентність даних ще до моменту звернення сервера до бази даних.

2.9 Розробка макетів користувацького інтерфейсу (UI/UX)

Проєктування користувацького інтерфейсу системи управління нерухомістю є складним інженерним завданням, оскільки менеджер або адміністратор агентства взаємодіє з системою протягом усього робочого дня. Інтерфейс повинен мінімізувати когнітивне навантаження, бути інтуїтивно зрозумілим і забезпечувати швидкий доступ до критичних операцій. Проєктування здійснювалося у спеціалізованому середовищі Figma з дотриманням принципів Component-Driven Design.

Інтерфейс системи логічно поділений на кілька ключових робочих областей, кожна з яких орієнтована на виконання специфічних завдань певними ролями працівників.

Головний дашборд управління платформами Цей екран є першою точкою входу після успішної автентифікації. Якщо користувач є власником кількох агентств нерухомості, на дашборді відображаються картки кожної платформи. Кожна картка містить коротку статистику та кнопку переходу до робочого простору. Також тут розміщено віджет фінансового профілю користувача, інформацію про його тарифний план та кнопку переходу до модуля створення тікетів (звернень до суперадміністраторів сервісу).

Розкладка календарної сітки бронювань є ключовим і найбільш технологічно складним елементом інтерфейсу, що є візуалізацією об'єднаних даних з таблиць Object та Booking.

Структура: Візуально календарна сітка бронювань являє собою двовимірну матрицю. У лівій закріпленій колонці по вертикалі розташовані об'єкти нерухомості, згруповані за категоріями або адресами. По горизонталі у верхній частині розміщено нескінченну стрічку календаря роки, місяці, дні тижня.

Елементи оренди: На перетині об'єктів і дат відображаються прямокутні блоки, що репрезентують записи з таблиці Booking. Довжина блоку відповідає тривалості проживання.

Кольорове кодування: Для миттєвого зчитування статусу застосовано колірну індикацію: жовтий колір означає попереднє бронювання, зелений – фактичне проживання, червоний – заборгованість по оплаті.

Взаємодія: Менеджер може створити новий запис шляхом натискання і розтягування курсора на порожніх клітинках. При кліку на існуючий блок відкривається бокова панель або модальне вікно з деталями: ім'я гостя, телефон, джерело, сума авансу та поле для текстових нотаток. Застосування бібліотеки управління станом дозволяє перемальовувати сітку при будь-яких змінах миттєво, без оновлення сторінки браузера.

Панель управління персоналом та налаштування Цей розділ доступний лише власникам платформи або працівникам з відповідними адміністративними правами. Інтерфейс являє собою таблицю, що виводить дані з PlatformEmployee. Власник може натиснути кнопку Запросити працівника, вказати його електронну пошту та за допомогою системи чекбоксів гнучко налаштувати права доступу: "Дозволити управління об'єктами", "Дозволити роботу з сіткою", "Дозволити перегляд фінансових звітів".

Панель суперадміністратора. Окрема захищена зона, доступ до якої мають виключно співробітники підтримки самого сервісу. Інтерфейс складається зі зведених графіків загального навантаження системи, таблиці всіх зареєстрованих користувачів, модулів для ручного коригування балансу та зміни тарифних планів. Також тут розміщено дошку для опрацювання тікетів від клієнтів, що дозволяє технічній підтримці відповідати на звернення та переводити їх у статус closed.

Таким чином, розроблена архітектура бази даних, логіка бізнес-процесів, структура REST API та макети користувацького інтерфейсу формують комплексну та масштабовану систему. Проектування на аналітико-модельному рівні гарантує, що майбутній програмний код буде стійким та безпечним з точки зору ізоляції даних орендарів та зручним для повсякденної операційної роботи менеджерів агентств нерухомості.

Висновки до розділу 2

У другому розділі здійснено перехід від аналізу вимог до моделювання та проєктування архітектури системи. Було детально обґрунтовано необхідність використання гібридної моделі контролю доступу RBAC + ABAC, яка гарантує принцип найменших привілеїв та чітку ізоляцію даних між незалежними платформами . Розроблено логічну структуру бази даних PostgreSQL, приведену до третьої нормальної форми (3NF), з акцентом на ключові сутності: User,

Platform, PlatformEmployee, Object та Booking. Встановлено критично важливий бізнес-процес створення бронювання, для захисту якого від колізій було спроєктовано механізм ізольованих транзакцій на рівні СУБД. Завершальним етапом стало проєктування модульної структури REST API, яка включає незалежні модулі Auth, Platforms, Objects та Bookings, а також розробка макетів користувацького інтерфейсу, центральним елементом якого є інтуїтивно зрозуміла календарна сітка боронованью з кольоровим кодуванням

РОЗДІЛ 3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ВЕБСЕРВІСУ

3.1. Реалізація клієнтської частини та користувацького інтерфейсу

Взаємодія з системою починається з екрана автентифікації. Після успішного входу користувач потрапляє до особистого кабінету. Головна панель виконана з використанням темної теми, що знижує навантаження на зір під час тривалої роботи. На цьому екрані користувач отримує зведену інформацію щодо поточного стану акаунта: доступний баланс, термін дії підписки та статистику по кількості працівників у підключених платформах.

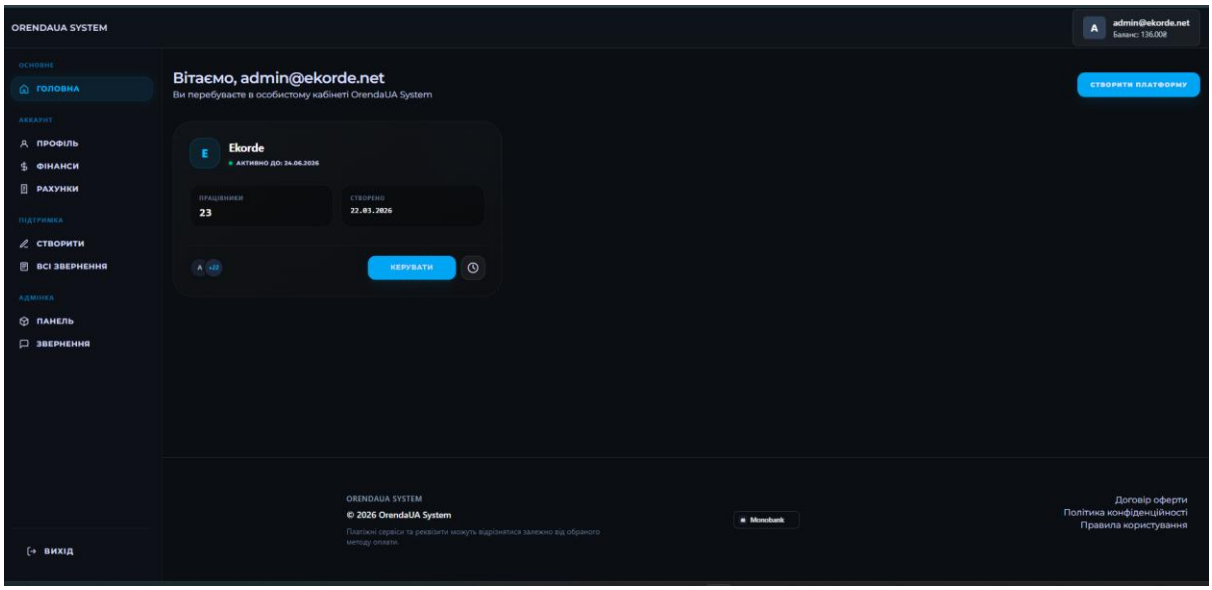

Рисунок. 3.1 – Головна панель (Дашборд) менеджера системи

Як видно з рис. 3.1, бокова навігаційна панель забезпечує швидкий доступ до профілю, фінансового блоку та системи підтримки.

Управління номерним фондом здійснюється всередині конкретної платформи. Центральним елементом цього внутрішнього інтерфейсу є календарна сітка бронювань. Вона реалізована у вигляді двовимірної матриці, де

по вертикальній осі розташовані доступні об'єкти нерухомості, а по горизонтальній – календарні дати.

Календарна сітка бронювань підтримує кольорове кодування статусів. Вільні дати відображаються світлим фоном, підтверджені бронювання та ті, що очікують на оплату, виділяються відповідними кольорами. Це значно знижує навантаження на менеджера та дозволяє миттєво оцінювати завантаженість.

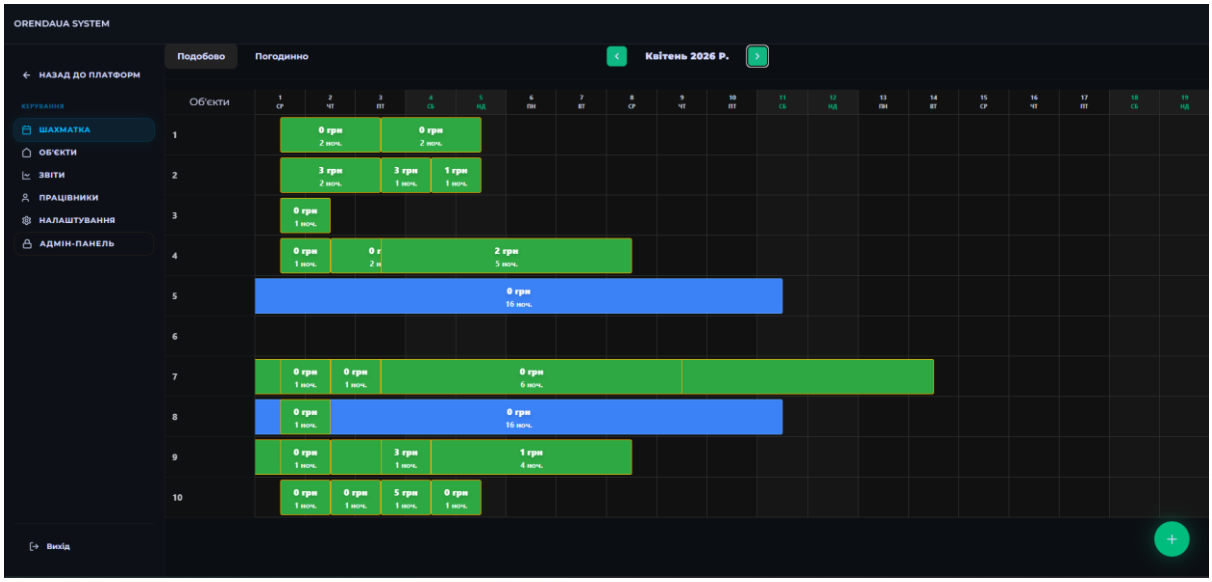

Рисунок. 3.2 – календарна сітка бронювань відображення статусів нерухомості

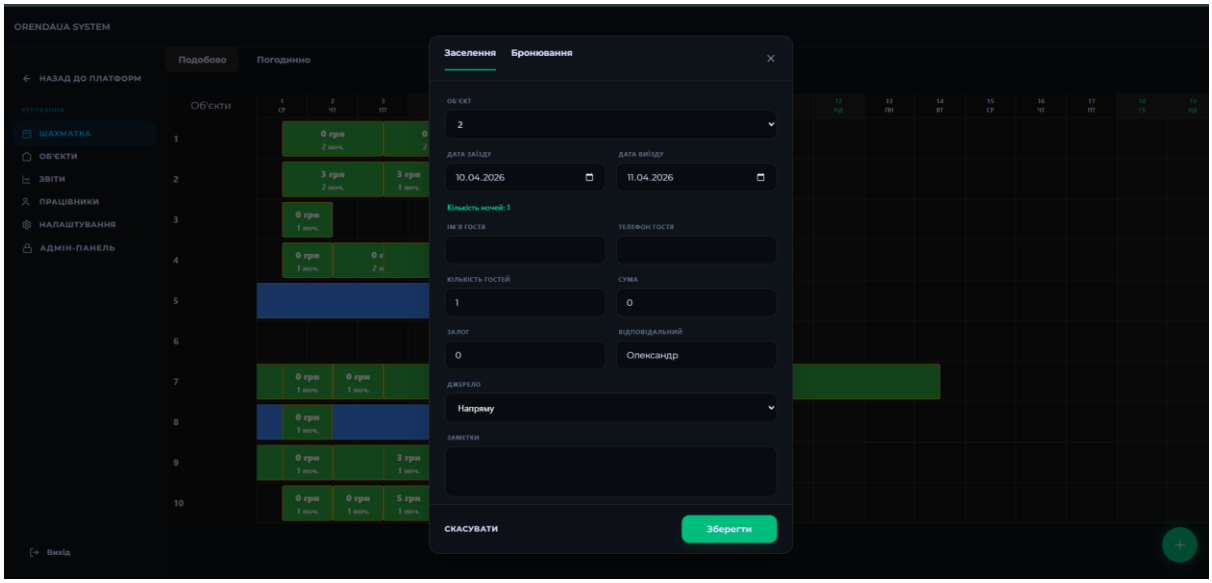

Рисунок. 3.3 – Інтерфейс створення та редагування бронювання

Користувач виділяє діапазон дат на сітці або натиснути відповідну кнопку створення, після чого відкривається модальне вікно для внесення деталей: персональних даних гостя, сум, дат заїзду/виїзду та додаткових приміток.

Перед відправкою, дані форми проходять контроль на стороні клієнта. Це зменшує кількість некоректних запитів до сервера та підвищує продуктивність і захищеність системи.

3.2. Архітектура та реалізація серверної частини (Backend)

Серверну частину розроблено на сучасному фреймворку Nest.js. Вибір цього фреймворку пояснюється його вбудованою підтримкою TypeScript та системою залежностей, що дозволяє будувати масштабовані та модульні додатки.

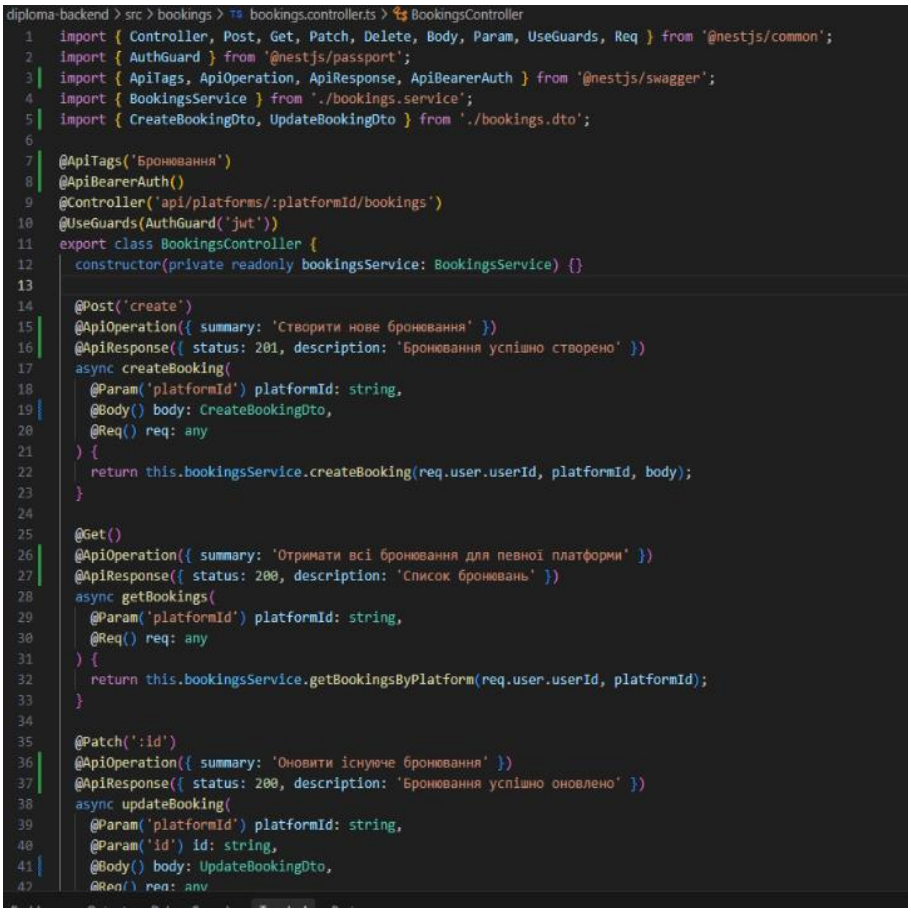

Рисунок. 3.4 – Фрагмент програмного коду контролера управління бронюваннями

Взаємодія з базою даних PostgreSQL реалізована через Prisma ORM, що забезпечує безпечний доступ до даних та автоматичні міграції схем.

3.3. Реалізація клієнтської частини на Vue.js

Клієнська частина реалізована за парадигмою Single Page Application з використанням фреймворку Vue.js та Composition API. Головною мета - створення високореактивного інтерфейсу, здатного миттєво реагувати на зміни даних, що є критично важливим для інтерактивної роботи з календарною сіткою бронювань.

3.3.1 Налаштування маршрутизації

Система використовує Vue Router для організації клієнтської маршрутизації. Було визначено два типи маршрутів:

1. Публічні маршрути, які не потребують автентифікації.

Для цих маршрутів впроваджено глобальний навігаційний захист, який перевіряє наявність та валідність JWT-токена у локальному сховищі. Якщо токен відсутній або недійсний, користувач автоматично перенаправляється на сторінку входу.

2. Реалізація інтерфейсів взаємодії з API.

Для відправки HTTP-запитів використовується бібліотека axios, налаштована як екземпляр з перехоплювачами. Кожен запит до захищених ресурсів автоматично зчитує токен зі сховища і додає його до заголовка. Це забезпечує автентифікацію для всіх операцій. Також, перехоплювачі обробляють помилки, відображаючи повідомлення користувачеві.

3.4. Розробка ключового функціоналу системи

Ключовий функціонал вебсервісу зосереджений навколо трьох взаємопов'язаних компонентів, необхідних для щоденної роботи менеджера агентства нерухомості.

Ядро системи, що забезпечує візуальний облік бронювань, реалізовано як складний компонент, оптимізований для відображення великих обсягів даних.

Завантаження даних відбувається наступнич чином: компонент викликає ендпоінт з обов'язковими параметрами, які відповідають видимому діапазону на екрані. Це дозволяє уникнути вивантаження всієї бази бронювань, знижуючи навантаження на мережу та СУБД.

Динамічне відображення реалізоване за наступним підходом: використано обчислювані властивості Vue.js для швидкого перетворення масиву записів бронювання на двовимірну матрицю для візуалізації. Кожен об'єкт нерухомості є окремим рядком, а його завантаженість відображається прямокутними блоками.

Кольорове кодування реалізовано через колірну індикацію на основі поля Booking.type та фінансового стану, в якому жовтий колір відображає попереднє бронювання, зелений колір для фактичного заселення, червоний колір для випадків, де виявлено заборгованість.

3.5. Впровадження багаторівневої системи доступу

Реалізація багаторівневої системи доступу стала ключовим етапом забезпечення безпеки системи. Впроваджена гібридна модель – RBAC на рівні маршрутів та ABAC на рівні ресурсів – забезпечує гнучке розмежування прав.

3.5.1 Створення захисту для контролерів бекенду

На рівні фреймворку Nest.js механізм Guards використовується для примусової перевірки прав доступу до REST API ендпоінтів.

AuthGuard - забезпечує обов'язкову наявність валідного JWT-токена в заголовку запиту. Якщо токен недійсний або відсутній, повертається HTTP 401 Unauthorized.

PermissionGuard - відповідає за перевірку прав доступу на рівні атрибутів. З URL-маршруту або тіла запиту витягується ідентифікатор платформи, а з декодованого JWT-токена ідентифікатор користувача. Отримані дані використовуються для звернення до таблиці у PostgreSQL, де зберігаються прапорці дозволів. У разі, коли працівник намагається виконати дію, на яку не має відповідного права - наприклад, створити об'єкт нерухомості за відсутності прапорця canManageObjects = true, Guard блокує подальшу обробку запиту та повертає клієнту статус 403 Forbidden.

Такий захист гарантує, що неавторизований користувач не зможе виконати операції, навіть якщо він якимось чином отримає доступ до структури API.

3.5.2 Динамічне відображення UI елементів на основі ролі працівника

На клієнтській частині використовується принцип динамічного рендерингу елементів інтерфейсу.

Vue-компоненти, які відповідають за критичні дії наприклад, кнопка "Видалити об'єкт", поле для редагування ціни бронювання, посилання на Фінансові звіти, обгорнуті в функцію, що перевіряє права.

Користувач бачить лише той функціонал, який йому дозволено використовувати, зменшуючи ризик помилкових дій.

3.5.3 Обмеження функцій за статусом підписки (Payment/Renew)

Реалізовано механізм обмеження функціоналу залежно від фінансового стану платформи. Ця перевірка виконується на бекенді під час обробки запиту.

Обмеження на створення ресурсів, який полягав в тому, що при спробі створити новий об'єкт, Guard додатково перевіряє, чи не перевищує кількість вже створених об'єктів ліміт.

Усі операції, що стосуються управління платформою, також перевіряються на термін дії підписки. Якщо термін вийшов, платформа переводиться у статус suspended, і більшість операцій, окрім читання та продовження підписки, блокуються.

3.5.4 Модуль управління персоналом та правами доступу

Система управління персоналом дозволяє власнику платформи гнучко керувати правами іншим співробітникам.

Інтерфейс управління працівниками являє собою таблицю, де відображаються імена та електронні пошти запрошених користувачів. Власник може динамічно змінювати права доступу кожного співробітника за допомогою системи перемикачів:

– Дозволяє створювати, редагувати та видаляти об'єкти нерухомості.

– Надає основне право на роботу з календарною сіткою, створення/редагування/скасування бронювань.

– Дозволяє переглядати фінансову аналітику та звіти.

Завдяки цьому Прибиральник матиме мінімальний набір прав, бачачи лише графік прибирань, тоді як Менеджер матиме повний доступ до бронювань та звітів.

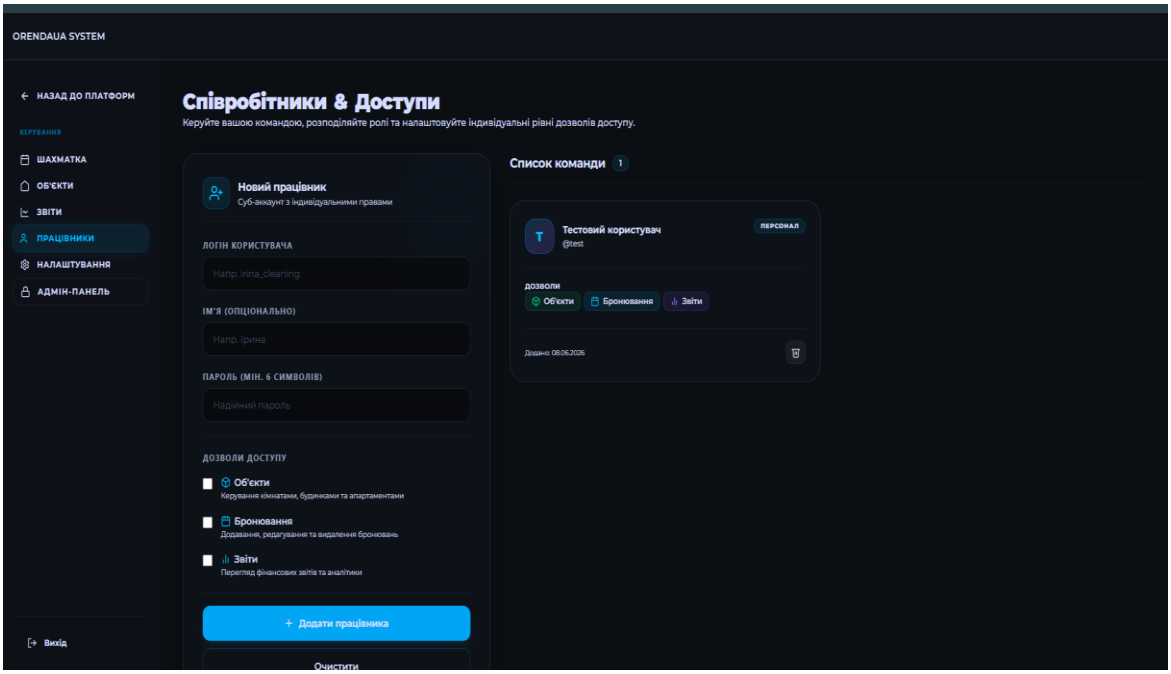

Рисунок. 3.5 – Інтерфейс управління персоналом («Працівники») з гнучким налаштуванням прав доступу

3.5.5 Панель Суперадміністратора сервісу (Admin Dashboard)

Для обслуговування самого сервісу реалізовано окрему, захищену зону – Панель Суперадміністратора. Доступ до цієї зони мають лише користувачі з глобальною роллю admin, які можуть виконувати операції, що стосуються функціонування всієї платформи.

Основний функціонал Адмін-панелі:

– Для обслуговування OrendaUA реалізовано окрему, захищену зону - Панель Суперадміністратора. Доступ до неї мають лише користувачі з глобальною роллю admin, які можуть виконувати операції, що стосуються функціонування всієї платформи.

– Опрацювання тікетів підтримки: Реалізовано список для ефективного опрацювання звернень від клієнтів, що зберігаються у таблиці Ticket. Суперадміністратор може відстежувати пріоритети звернень та управляти їхнім статусом.1

– Моніторинг системи: Зведені графіки відображають загальне навантаження на сервіс, кількість активних користувачів та кількість запитів до API.

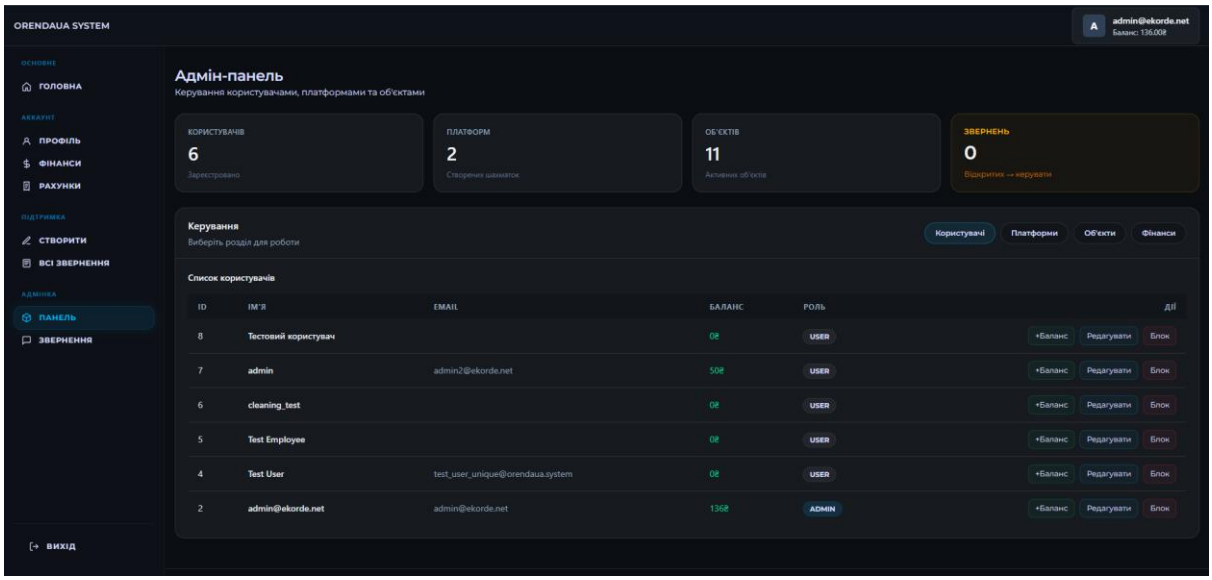

Рисунок. 3.8 – Панель Суперадміністратора сервісу OrendaUA (Admin Dashboard)

3.6. Система створення та редагування бронювань

Створення та редагування бронювань здійснюється через єдине модальне вікно, що стандартизує введення даних та спрощує взаємодію менеджера із системою.

Створення: Ініціюється кліком та виділенням вільного діапазону на календарній сітці. Клієнтський застосунок отримує objectId, startDate та endDate і передає їх у форму. Перед відправкою запиту на сервер відбувається локальна перевірка валідності заповненість полів гостя, коректність фінансових даних.

Редагування/Оновлення статусу: При кліку на існуючий блок відкривається те ж саме модальне вікно, заповнене даними бронювання. Доступна можливість зміни статусу, а також внесення коригувань у ціну та аванс.

Запобігання овербукінгу: Навіть при спробі створення броні через UI, на бекенді ініціюється транзакційна перевірка на перетин дат, що гарантує цілісність даних

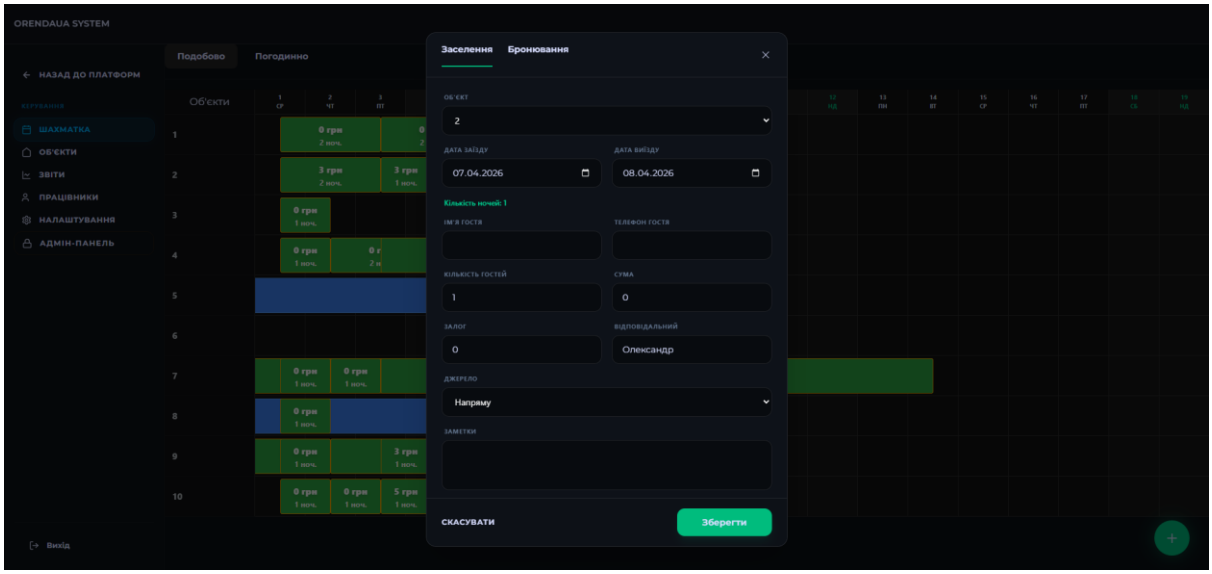

Рисунок. 3.9 – Інтерфейс модального вікна створення або редагування бронювання

3.6.1. Фільтрація нерухомості за категоріями та статусами

Для підвищення зручності роботи з великим фондом нерухомості реалізовано динамічну систему фільтрації. Фільтри розташовані на боковій панелі та дозволяють швидко сегментувати дані

За категоріями/типами: Дозволяє відображати на сітці лише об'єкти певного типу Квартири-студії або "Приватні будинки".

За статусами завантаженості: Дозволяє приховувати повністю заброньовані об'єкти або, навпаки, показувати лише ті, що потребують прибирання.

Зміна параметрів фільтрації оновлює запит до API, що дозволяє бекенду повертати лише релевантні об'єкти та їхні бронювання, забезпечуючи високу швидкість взаємодії.

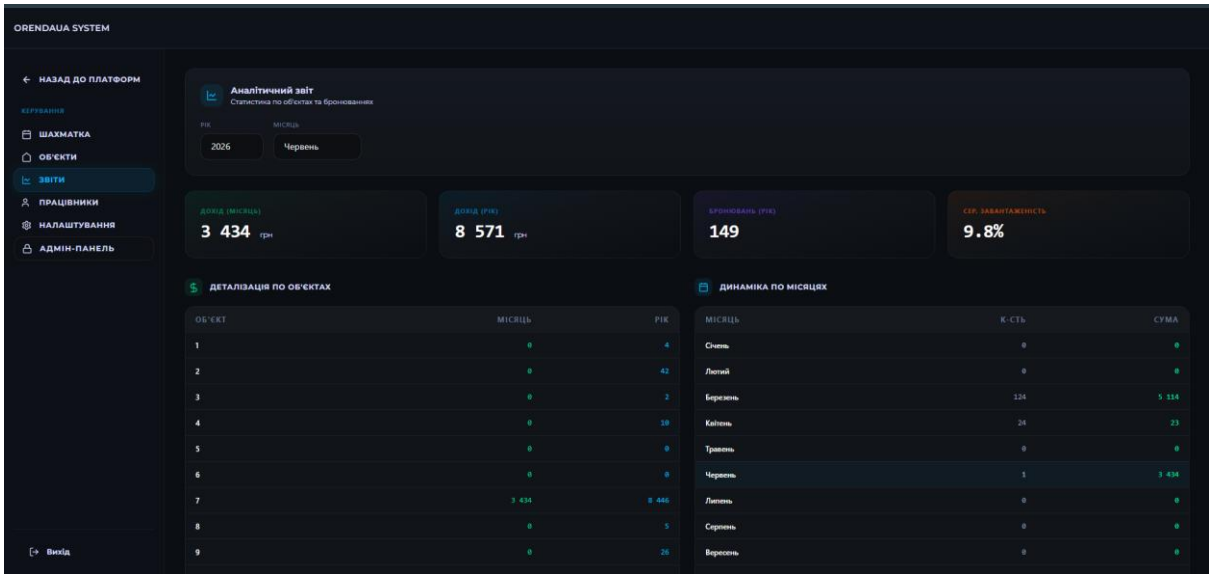

Рисунок. 3.10 – Аналітичний звіт: Статистика по об'єктах та бронюваннях

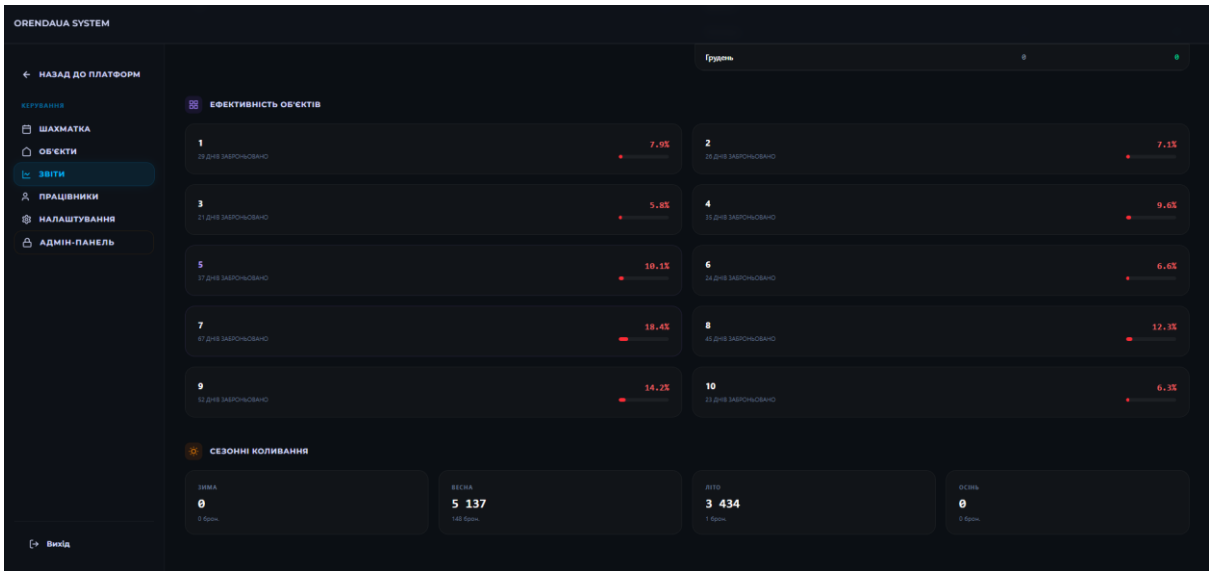

Рисунок. 3.11 – Аналітичний звіт: Статистика по об'єктах та бронюваннях

3.6.2. Особистий кабінет користувача та механізми безпеки

Особистий кабінет користувача реалізовано як інтерфейс для управління персональними даними, налаштуваннями безпеки та моніторингу фінансового стану облікового запису. На сторінці Профілю відображається ключова інформація: Email адреса, поточна роль користувача, актуальний баланс та технічні метадані, такі як дата реєстрації.

Для підвищення рівня захисту облікових даних впроваджено функціонал двофакторної автентифікації, що активується через механізм email-коду. Цей метод безпеки додає другий етап перевірки при вході в систему, вимагаючи введення одноразового коду, надісланого на електронну пошту, що ускладнює несанкціонований доступ. Інтерфейс містить кнопку Увімкніть 2FA для налаштування цієї функції. Завдяки цьому забезпечується відповідність системи сучасним стандартам інформаційної безпеки.

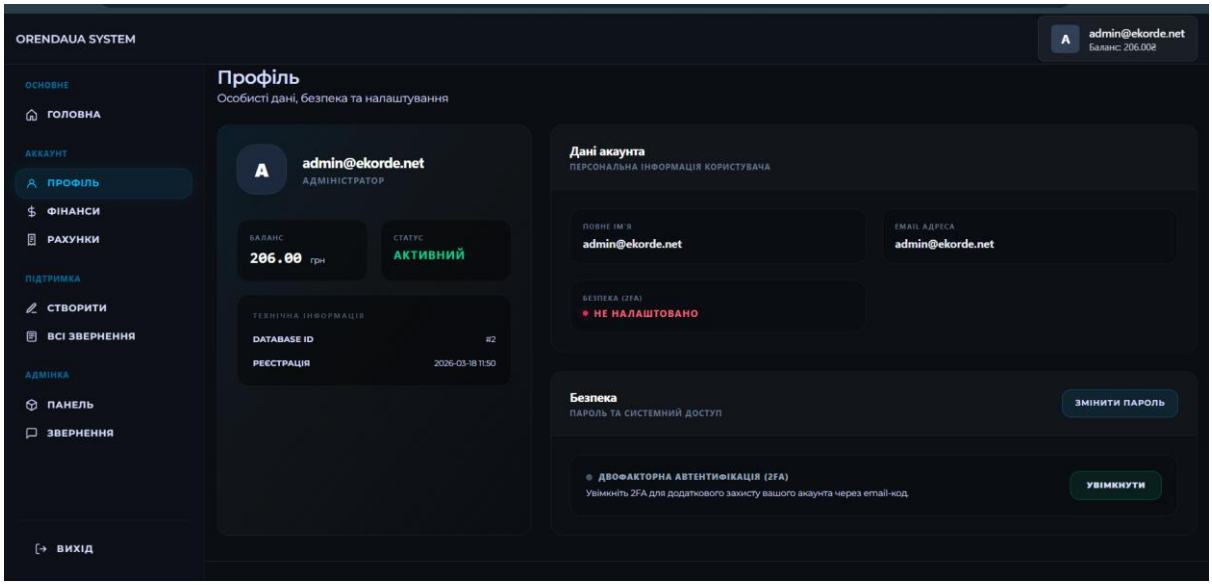

Рисунок 3.12 – Інтерфейс Особистого кабінету та налаштування двофакторної автентифікації

3.6.5. Фінансовий модуль, поповнення балансу та облік рахунків

3.6.5.1. Поповнення балансу через Monobank Jar

Фінансовий модуль забезпечує керування коштами в межах системи. Користувач може в будь-який момент переглянути поточний баланс, вартість активного тарифу та історію транзакцій.

Поповнення балансу реалізовано через інтеграцію з Monobank Jar. Користувач вводить або обирає потрібну суму та натискає кнопку поповнення, після чого відкривається модальне вікно з реквізитами для переказу та зворотним

таймером, відведеним на оплату. Після успішного переказу кошти автоматично зараховуються на баланс акаунта. Мінімальна сума поповнення становить 12 грн, комісія відсутня.

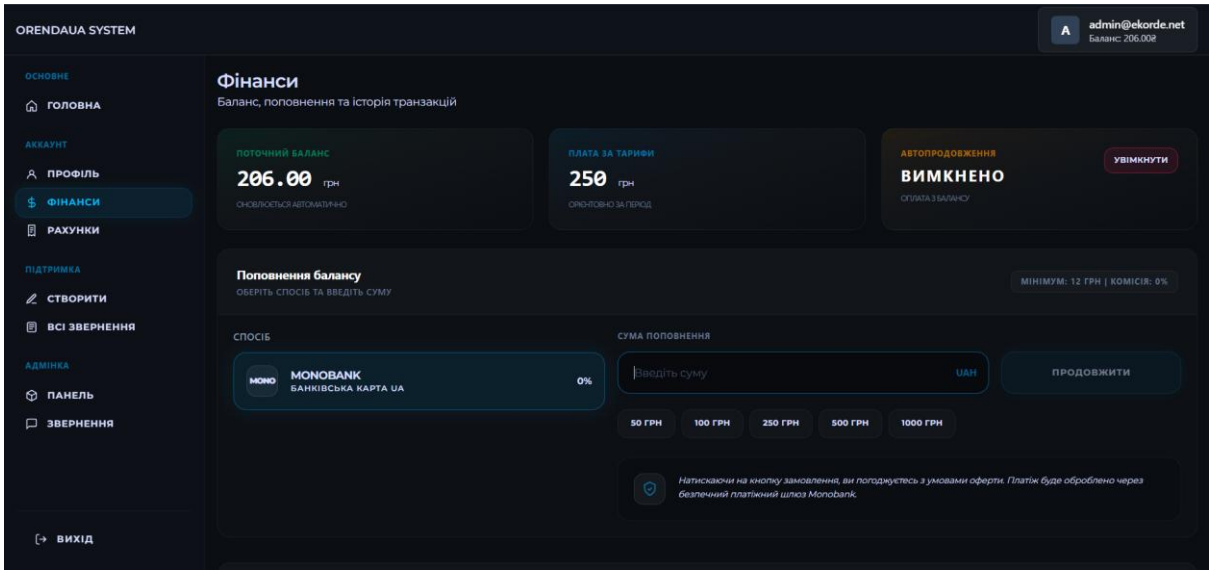

Рисунок 3.13 – Інтерфейс сторінки роботи з фінансами

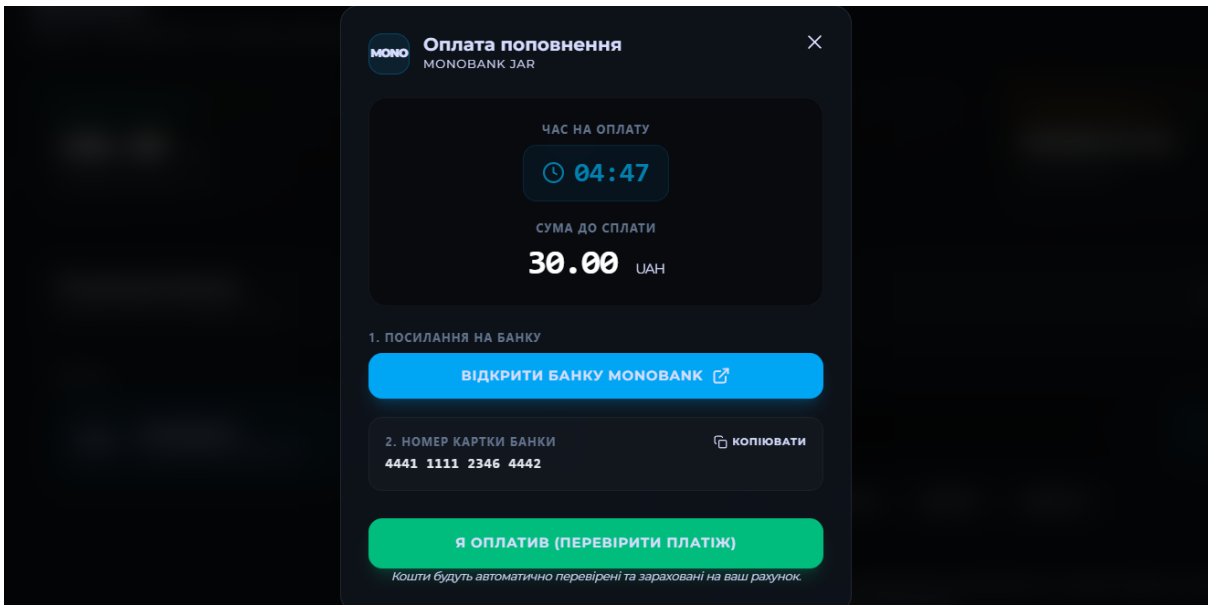

Рисунок 3.14 – Інтерфейс фінансового модуля та механізм поповнення балансу через Monobank Jar

3.6.5.2. Сторінка обліку рахунків

Для забезпечення повного фінансового контролю реалізовано окремий розділ Рахунки, де зберігається історія всіх платіжних операцій, незалежно від їхнього статусу. На цій сторінці зберігаються дані про усі успішні та неуспішні оплати та транзакції. Це дозволяє користувачеві мати повний звіт своїх фінансових взаємодій із системою.

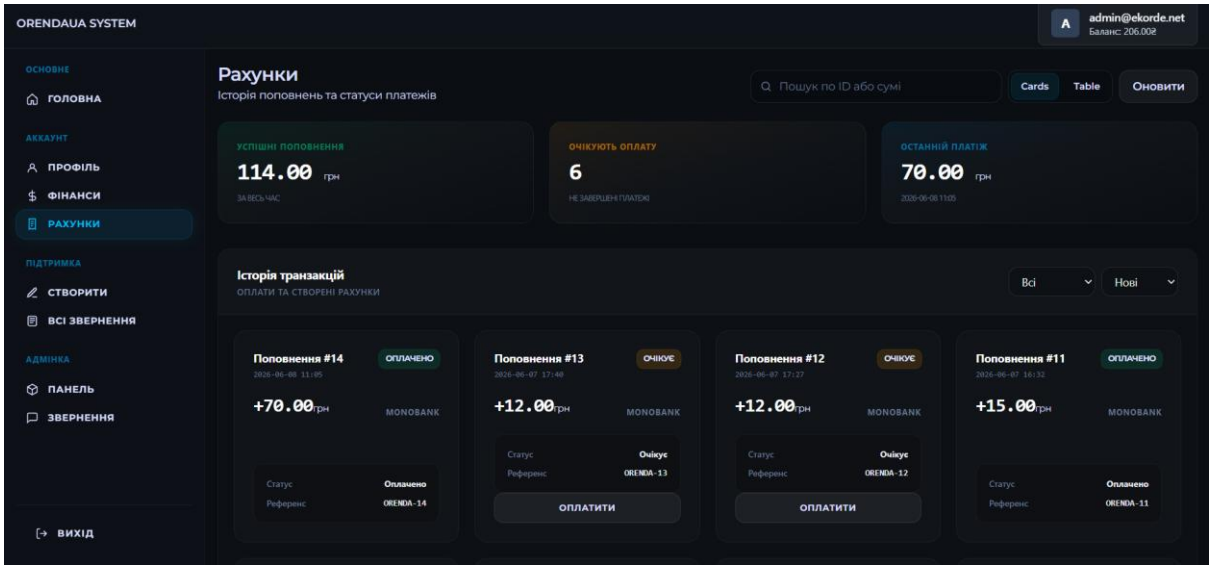

Рисунок 3.14 – Сторінка керування фінансами

3.6.6. Система тікетів технічної підтримки

Підтримка клієнтів реалізована через внутрішню систему тікетів, що дозволяє власникам платформ створювати структуровані звернення до технічного відділу сервісу. Форма для створення звернення надає детальний опис проблеми та встановлення рівня її пріоритету.

Усі створені звернення зберігаються на сторінці Всі звернення. Цей інтерфейс дозволяє користувачеві відстежувати активні та закриті тікети, використовуючи фільтри за статусом Всі, Відкриті, Закриті та пріоритетом. Кожен тікет відображається з унікальним ідентифікатором, тип, статус та дату

останнього оновлення. Наявність системи забезпечує структуровану комунікацію між клієнтом та технічною підтримкою.

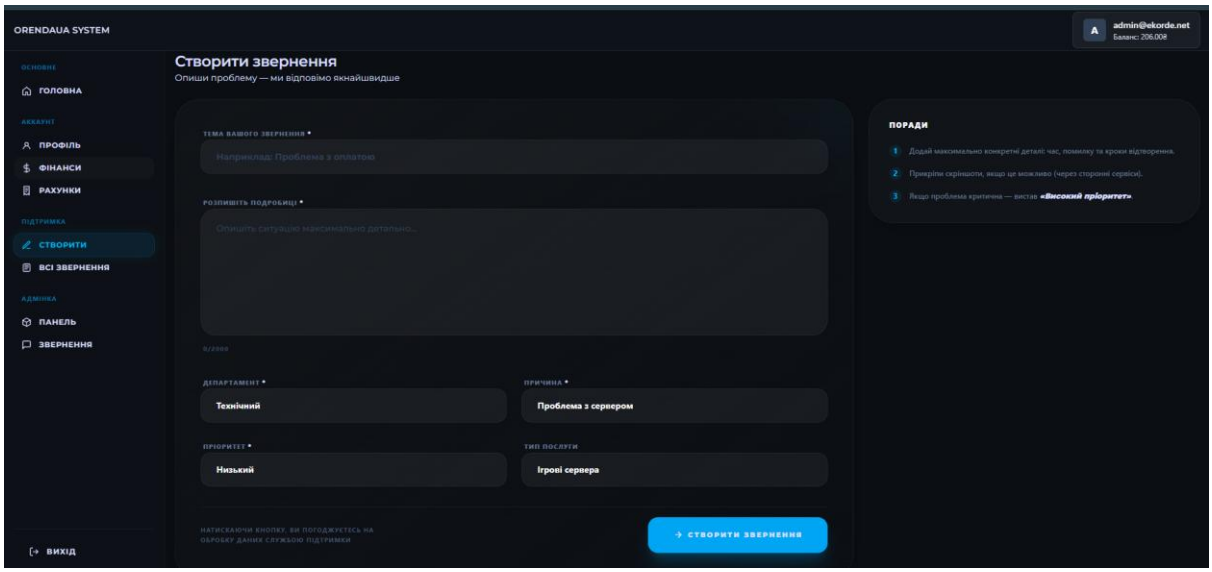

Рисунок 3.15 - Інтерфейс створення звернення до технічної підтримки

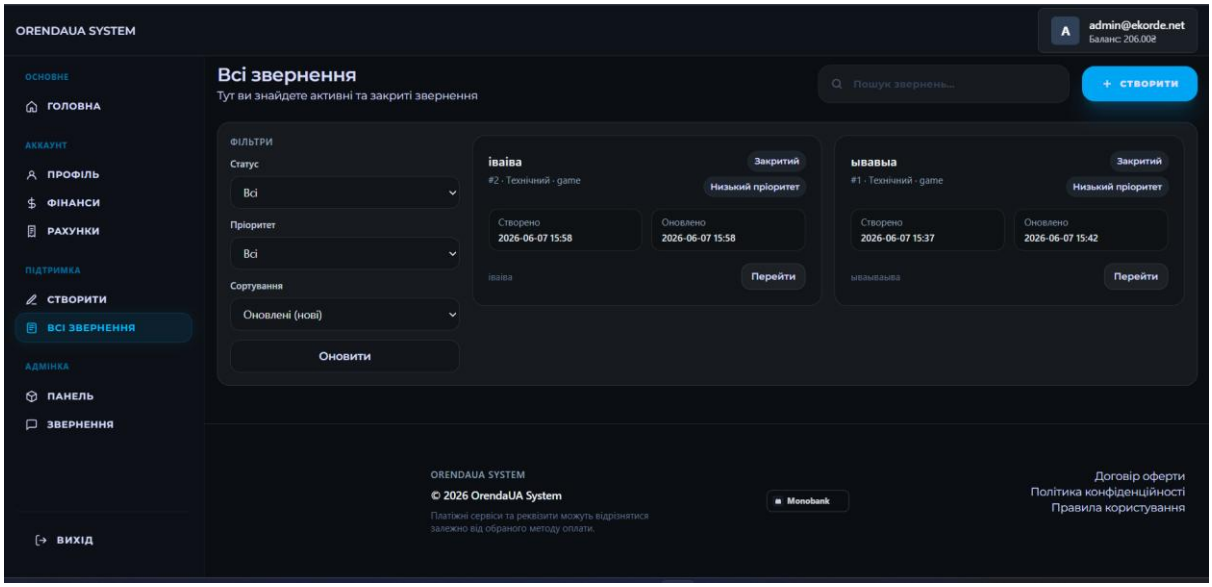

Рисунок 3.16 - Сторінка перегляду всіх звернень користувача

Висновки до розділу 3

У третьому розділі здійснено практичну реалізацію вебсервісу OrendaUA на обраному технологічному стеку. Клієнтська частина реалізована як Single Page Application з використанням Pinia для управління глобальним станом, а інтерфейс оформлено у Dark Mode з адаптивністю. Було успішно впроваджено багатокомпонентну систему безпеки для перевірки прав доступу. Реалізовано ключовий функціонал: інтерактивна календарна сітка, панель управління персоналом та налаштування їхніх прав доступу, фінансовий модуль з поповненням балансу через Monobank Jar, а також система тікетів технічної підтримки.

РОЗДІЛ 4 ТЕСТУВАННЯ ТА АНАЛІЗ ЕФЕКТИВНОСТІ ВЕБСЕРВІСУ

Тестування є важливою частиною життєвого циклу розробки програмного забезпечення, відповідність реалізованої системи сформованим функціональним та нефункціональним вимогам, зокрема вимогам надійності, безпеки та продуктивності. Верифікація вебсервісу включала три ключові аспекти: функціональну коректність валідація бізнес-логіки, тестування безпеки контроль доступу та цілісність даних а також нефункціональне тестування.

4.1 Стратегія тестування та підготовка тестового середовища

На цьому етапі була визначена загальна методологія тестування, яка базується на методах емпіричного тестування, та підготовлена необхідний інструментарій для забезпечення повної верифікації системи. Тестування проводилося відповідно до розробленого плану, що охоплював усі фукції системи.

Розробка плану тестування та написання тестових сценаріїв Test Cases

Було сформовано набір тестових сценаріїв Test Cases, які охоплюють функціональні вимоги, безпекові вимоги та вимоги до продуктивності. Кожен сценарій деталізував початкові умови, кроки виконання та очікуваний результат.

Для фіксації виявлених дефектів та контролю їх усунення Bug Reports)було обрано систему трекінгу, що дозволяє моніторити метрики якості коду та забезпечувати контроль версій.

4.2. Дослідження роботи системи та тестування

Метою тестування була перевірка стійкості системи до пікових навантажень та ефективності механізму захисту від овербукінгу. Змодельовано сценарій, у якому кілька запитів на бронювання одного об'єкта надходять одночасно на однакові дати. Тестування підтвердило, що завдяки ізольованим транзакціям PostgreSQL лише один із конкурентних запитів успішно завершується зі статусом 201 Created. Решта запитів, що намагаються зафіксувати бронювання на вже зайнятий період, коректно отримують статус 409 Conflict, що засвідчує надійність системи в умовах високого навантаження.

Змодельовано сценарій одночасного надходження кількох запитів на бронювання одного об'єкта на однакові дати. Тестування підтвердило, що механізм ізольованих транзакцій PostgreSQL забезпечує коректну обробку конкурентних запитів: лише один із них завершується успішно зі статусом 201 Created, тоді як решта запитів, що стосуються вже зайнятого періоду, отримують статус 409 Conflict. Отримані результати засвідчили високу надійність системи під час виконання критичних операцій.

Тестування перевірки прав доступу

Проведено цільове тестування підсистеми безпеки, що реалізує гібридну модель контролю доступу.

Перевірка ролей: виконано вхід під обліковими записами з різними ролями. Спроба здійснити операцію, що виходить за межі наданих повноважень, коректно блокується із поверненням статусу 403 Forbidden.

4.2.1 Тестування правильності дат та перетинів бронювань

Здійснено перевірку логіки роботи з часовими відрізками, включаючи граничні випадки:

Day-off перетин: Тестування запитів, які починаються у день виїзду іншого гостя або закінчуються у день заїзду, щоб забезпечити коректну логіку.

Валідація клієнта: Перевірено, що фронтенд-валідація запобігає відправці запиту, де дата виїзду раніша за дату заїзду.

4.3 Функціональне тестування серверної та клієнтської частин

Функціональне тестування підтвердило коректність реалізації бізнес-логіки, що є основою надійної роботи вебсервісу.

Перевірено коректність створення, читання, оновлення та видалення об'єктів нерухомості та платформ. Це включало валідацію вхідних даних та перевірку, що об'єкти, пов'язані з видаленою платформою, коректно видаляються з бази даних.

При моделюванні сценарію множинних запитів одночасна спроба забронювати один об'єкт, бекенд ізольованим транзакціям PostgreSQL , успішно фіксував лише одне бронювання, повертаючи для всіх інших спроб статус 409 Conflict

Тестування управління глобальним станом та коректності відмальовування календарної сітки бронювань

На клієнтській частині перевірено, що оновлення даних миттєво призводить до перемальовування відповідного прямокутного блоку на календарній сітці згідно з кольоровим кодуванням. Також перевірено, що фільтрація об'єктів коректно оновлює запит до API, вивантажуючи лише релевантні дані.

4.4 Тестування безпеки, авторизації та розмежування прав доступу

Перевірка механізмів безпеки є важливою для інформаційної системи, так як вона оперує конфіденційною інформацією про бронювання, фінансовими транзакціями та надає адміністративні інструменти керування середовищем.

Для перевірки захищеності системи було проведено функціональне тестування механізмів авторизації та безпеки шляхом прямої взаємодії з інтерфейсом користувача, а також через надсилання HTTP-запитів за допомогою інструментів тестування Swagger UI. Це дає можливість проаналізувати поведінку сервера при спробах доступу до захищених маршрутів з різними рівнями прав доступу або без автентифікації. Тестування безпеки охоплювало кілька ключових рівнів

Під час тестування перевірено надійність захисту облікових записів. Верифіковано, що в базі даних PostgreSQL паролі користувачів зберігаються виключно у вигляді солених хешів, згенерованих через bcrypt. Було протестовано поведінку системи при зверненні до закритих. У разі відсутності JWT-токена в заголовках запиту або при спробі передачі невалідного токена, сервер коректно повертає помилку 401 Unauthorized і блокує доступ. Також підтверджено працездатність двофакторної автентифікації - вхід у систему стає можливим лише після введення одноразового коду, надісланого на електронну пошту користувача.

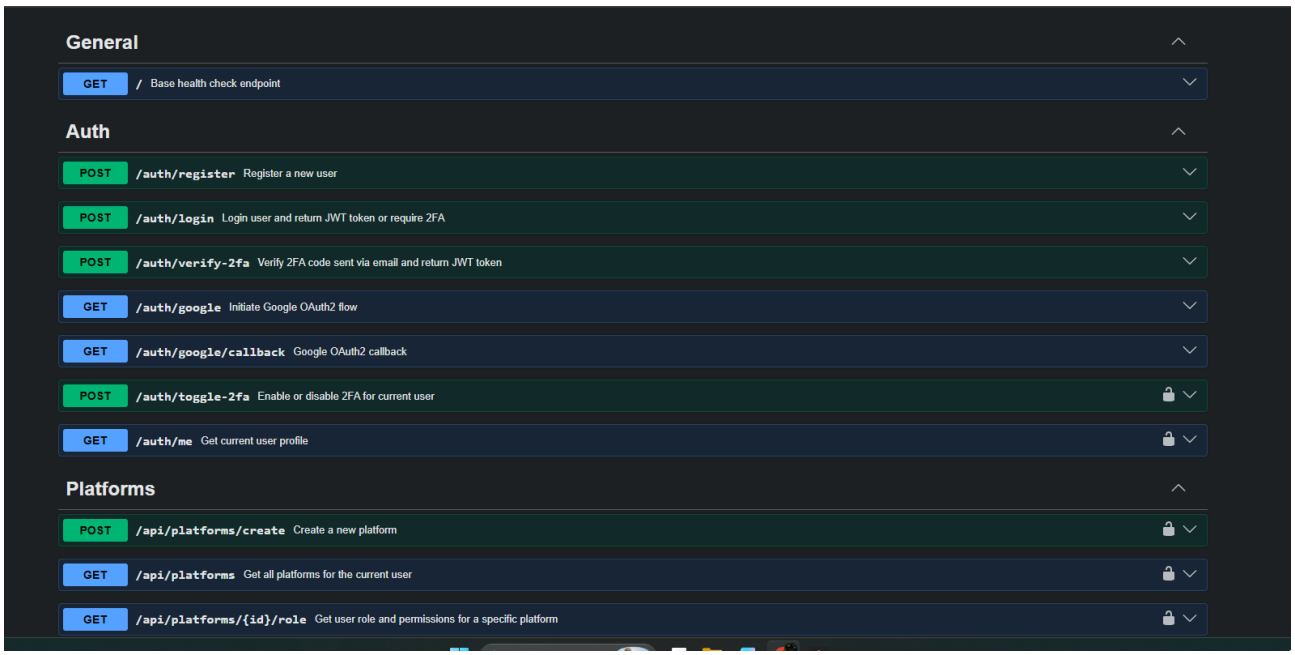

Рисунок 4.1 – Веб-інтерфейс Swagger UI для тестування API

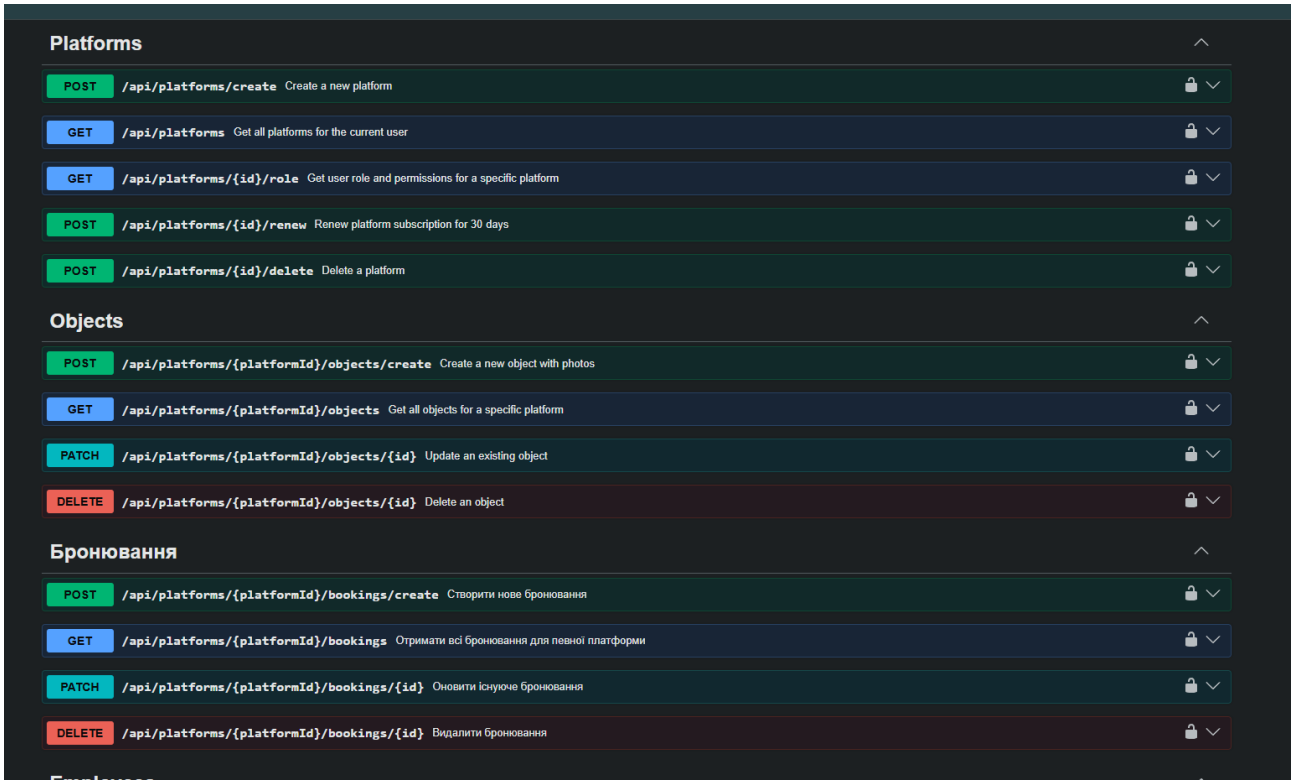

Рисунок 4.2 – Веб-інтерфейс Swagger UI для тестування API

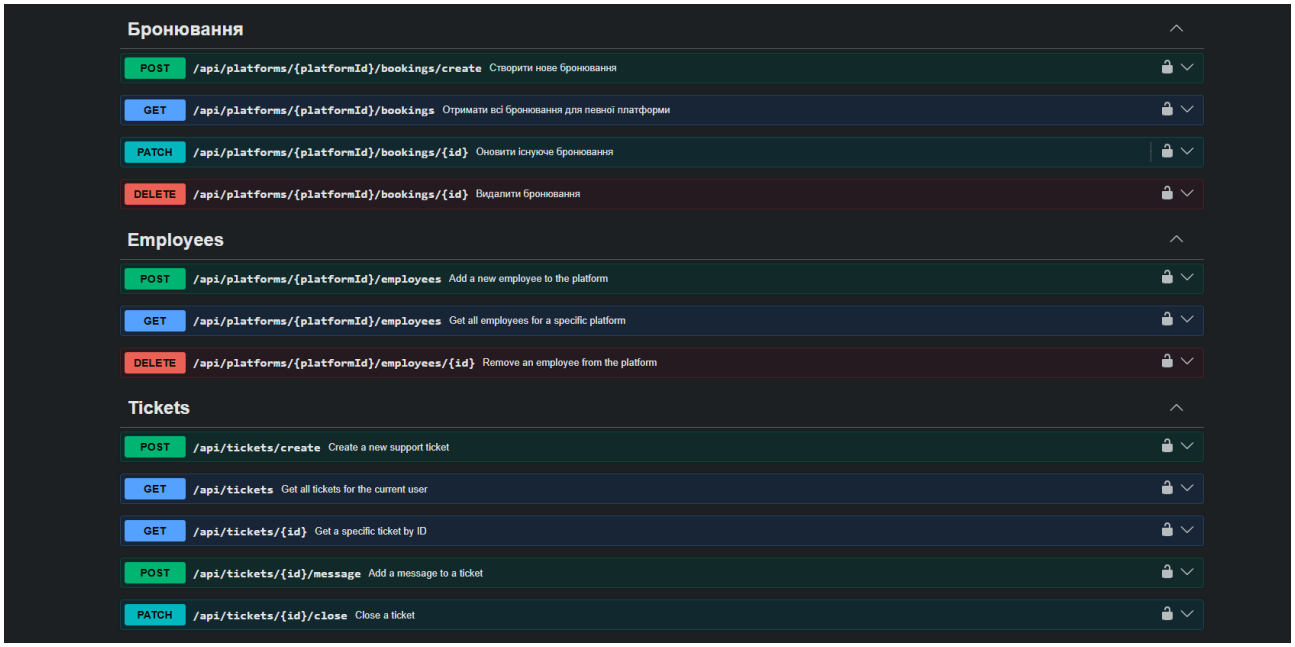

Рисунок 4.3 – Веб-інтерфейс Swagger UI для тестування API

Рисунок 4.4 – Веб-інтерфейс Swagger UI для тестування API

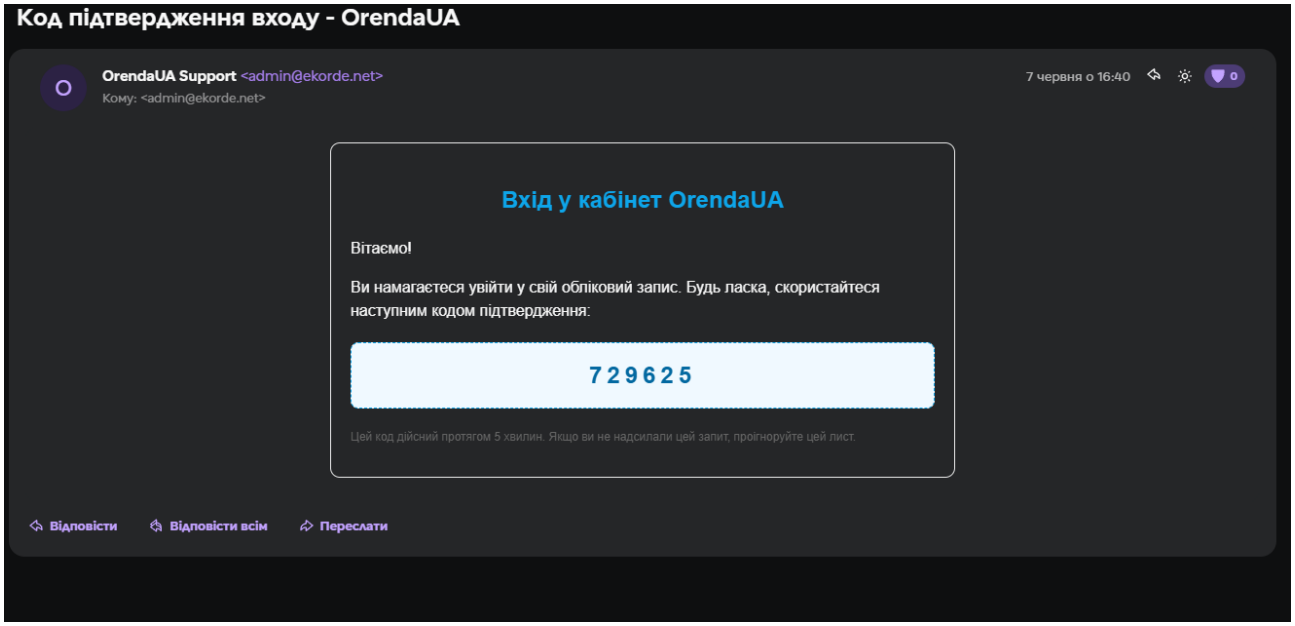

Рисунок 4.5 – Лист з кодом підтвердження при увімкненій двофакторної автентифікації

4.5 Тестова документація: Чек-листи та Детальні Тест-кейси

Цей документ містить перелік чек-листів та детальних тест-кейсів для функціонального, інтеграційного та безпекового тестування платформи управління орендою нерухомості OrendaUA

4.5.1. Тест-кейси (Test Cases)

TC-01: Реєстрація нового користувача та налаштування 2FA. Пріоритет:High. Передумови: Користувач не авторизований, електронна пошта testuser@example.com не зареєстрована в системі.

Крок	Дія	Очікуваний результат
1	Перейти на сторінку реєстрації /login .	Відображається форма реєстрації з полями: Name, Email, Password.
2	Ввести Name: Олександр, Email: testuser@example.com, Password: password123 та натиснути Зареєструватися.	Користувача створено, автоматично виконано вхід. Редирект на /dashboard.
3	Перейти на сторінку профілю /profile .	У профілі відображається тариф free, баланс 0 грн , статус 2FA: Вимкнено .
4	Натиснути кнопку "Увімкнути двофакторну автентифікацію".	Статус 2FA змінюється на Увімкнено .
5	Вийти з акаунта.	Користувача перенаправлено на сторінку входу /login .

6	Ввести email testuser@example.com, пароль password123 та натиснути Увійти .	З'являється вікно введення коду 2FA. На пошту відправлено код.
7	Ввести правильний код з пошти та підтвердити.	Вхід виконано успішно, редирект на /dashboard .

ТС-02: Створення майданчика та перевірка лімітів Free-тарифу

Крок	Дія	Очікуваний результат
1	Перейти на /dashboard та натиснути Створити платформу.	Відкривається модальне вікно створення майданчика.
2	Ввести назву: Мій перший готель та натиснути Створити.	Платформа успішно створено. Термін дії встановлено на +30 днів.
3	Спробувати створити другий майданчик.	Система блокує створення та видає помилку: Ви вичерпали ліміт сіток бронювань для вашого тарифу. (Помилка 403 від API).
4	Створити послідовно 3 об'єкти (Апартаменти 101, 102 , 103).	Усі три об'єкти успішно створюються.
5	Спробувати створити 4-й об'єкт.	Система блокує дію з помилкою:Вивичерпали ліміт об’єктів для вашого тарифу.

ТС-08: Створення тікета підтримки та відповідь адміністратора

Пріоритет:Medium Передумови: Авторизовано звичайного користувача. Є окремий акаунт адміністратора

Крок	Дія	Очікуваний результат
1	Перейти у розділ "Підтримка", створити тікет (Тема, Опис, Пріоритет, Високий).	Тікет успішно створено зі статусом open .
2	Написати додаткове повідомлення в чат тікета.	Повідомлення відображається в чаті.
3	Адміністратор входить у Адмін-панель, знаходить новий тікет.	Адміністратор бачить тікет із пріоритетом Високий.
4	Адміністратор пише відповідь: Вітаємо. Ліміт на фото становить 3 МБ.	Повідомлення адміна відображається в чаті з позначкою Адміністратор.
5	Адміністратор змінює статус тікета на Закрито.	Статус тікета змінюється на closed.
6	Користувач перевіряє тікет.	Користувач бачить відповідь адміністратора та статус closed

4.5.2. Чек-листи Checklists – Зведена Таблиця Перевірок

Модуль	Підрозділ	Чек-лист
Авторизація та Безпека	Реєстрація	Перевірка обов'язковості полів (Email, Ім'я, Пароль).
		Валідація унікальності Email (заборона повторної реєстрації).
		Валідація пароля (мінімум 6 символів).
		Автоматичне призначення тарифу free та лімітів (1 майданчик, 3 об'єкти).
	Вхід	Авторизація за коректними обліковими даними.
		Обробка невірних паролів/email (інформативні повідомлення).
	2FA	Можливість увімкнути/вимкнути двофакторну автентифікацію в профілі.
		Надсилання одноразового коду на Email під час входу.
Профіль та Тарифи	Ліміти	Перевірка автоматичного застосування обмежень для тарифів Free, Start, Business .
		Блокування створення об'єктів/майданчиків при перевищенні лімітів.
	Профіль	Блокування створення об'єктів/майданчиків при перевищенні лімітів.

Майданчики та Об'єкти	Платформа	Створення нового майданчика.
		Нарахування терміну дії на 30 днів при створенні/продовженні.
		Призупинення доступу suspended після завершення терміну дії expiresAt.
		Видалення платформи власником (видалення пов'язаних даних).
	Об'єкти	Додавання об'єкта нерухомості (назви, опис) до платформи.
		Редагування інформації та видалення об'єкта.
Календар та Бронювання	Візуалізація	Коректне відображення сітки дат (дні, тижні, місяці) та об'єктів.
		Візуальне відображення статусів бронювань різними кольорами (заселення, статус оплати).
	Управління	Створення нового бронювання (заповнення всіх полів).
		Валідація овербукінгу (блокування створення при перетині дат).
		Перетягування (Drag & Drop) або зміна розміру блоку (з автоматичною перевіркою конфліктів).
Співробітники та Доступи	Управління	Додавання співробітника за username.

		Призначення індивідуальних прав: canManageObjects, canManageBookings, canViewReports.
	Перевірка	Співробітник без прав не може додавати/редагувати бронювання (HTTP 403).
		Співробітник без права не може додавати/редагувати об'єкти (UI приховано / HTTP 403).
Фінанси та Monobank	Поповнення	Створення рахунку на поповнення балансу (перевірка мінімальної суми 12 грн).
		Генерація унікального ідентифікатора рахунку ORENDA-{id} та посилання на Банку Monobank.
		Передача правильних параметрів у URL для оплати.
	Верифікація	Зміна статусу рахунку з pending на paid після успішного підтвердження.
		Миттєве збільшення балансу користувача та запис логу транзакції.
Технічна Підтримка	Користувач	Створення тікета (Тема, Опис проблеми, Департамент, рівень Пріоритету).
		Перегляд списку власних тікетів (активні / закриті) та чат.

	Адмін-панель	Доступ до панелі адміністрування лише користувачам із роллю admin .
		Сортування, фільтрація та закриття тікетів адміністратором.

4.6. Аналіз результатів тестування та зручності користування

Результати комплексного тестування дозволили оцінити як технічні показники продуктивності системи, так і її відповідність вимогам користувацького досвіду.

Оцінка адаптивності інтерфейсу

Використовуючи utility-first фреймворку Tailwind CSS, користувацький інтерфейс виявився повністю адаптивним.

Mobile/Tablet-Friendly: Календарна сітка бронювань масштабується на мобільних пристроях, забезпечуючи зручне прокручування та відображення ключової інформації.

Зниження навантаження: Кольорове кодування статусів бронювання та розділення робочих зон, знижує когнітивне навантаження, дозволяючи менеджерам швидше приймати рішення.

4.7. Оцінка загальної ефективності та безпеки системи

На основі проведеної практичної реалізації та тестування, вебсервіс OrendaUA може бути оцінений як високопродуктивне та безпечне програмне рішення, що повністю відповідає поставленим у роботі цілям.

Функціональна повнота: Реалізовано весь ключовий функціонал, необхідний для автоматизації процесів обліку нерухомості: управління об'єктами, створення та моніторинг бронювань через календарну сітку, управління персоналом та доступом, а також база для фінансової аналітики.

Продуктивність: Обраний стек Nest.js, Vue.js, PostgreSQL забезпечив низьку затримку відгуку API та високу швидкість обробки даних.

Цілісність даних: Впровадження механізму транзакцій на рівні бази даних під час операцій бронювання ефективно запобігає виникненню подвійних бронювань, гарантуючи фінансову надійність.

Розмежування доступу: Гібридна модель забезпечує ізоляцію даних між незалежними агентствами та дотримання принципу найменших привілеїв, що було підтверджено тестуванням.

4.8 Розгортання та інфраструктура

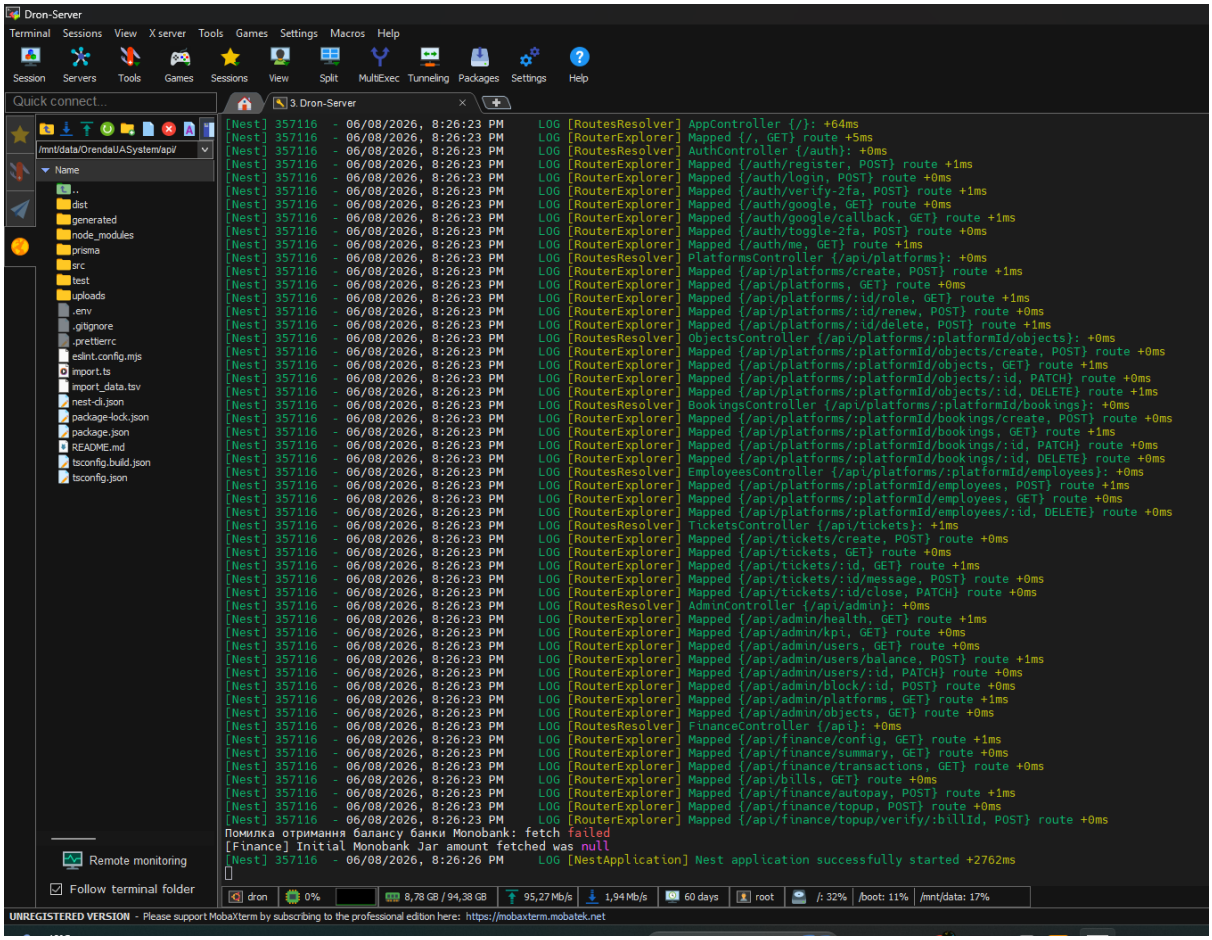

Рисунок 4.6 – Запущений Backend на сервері

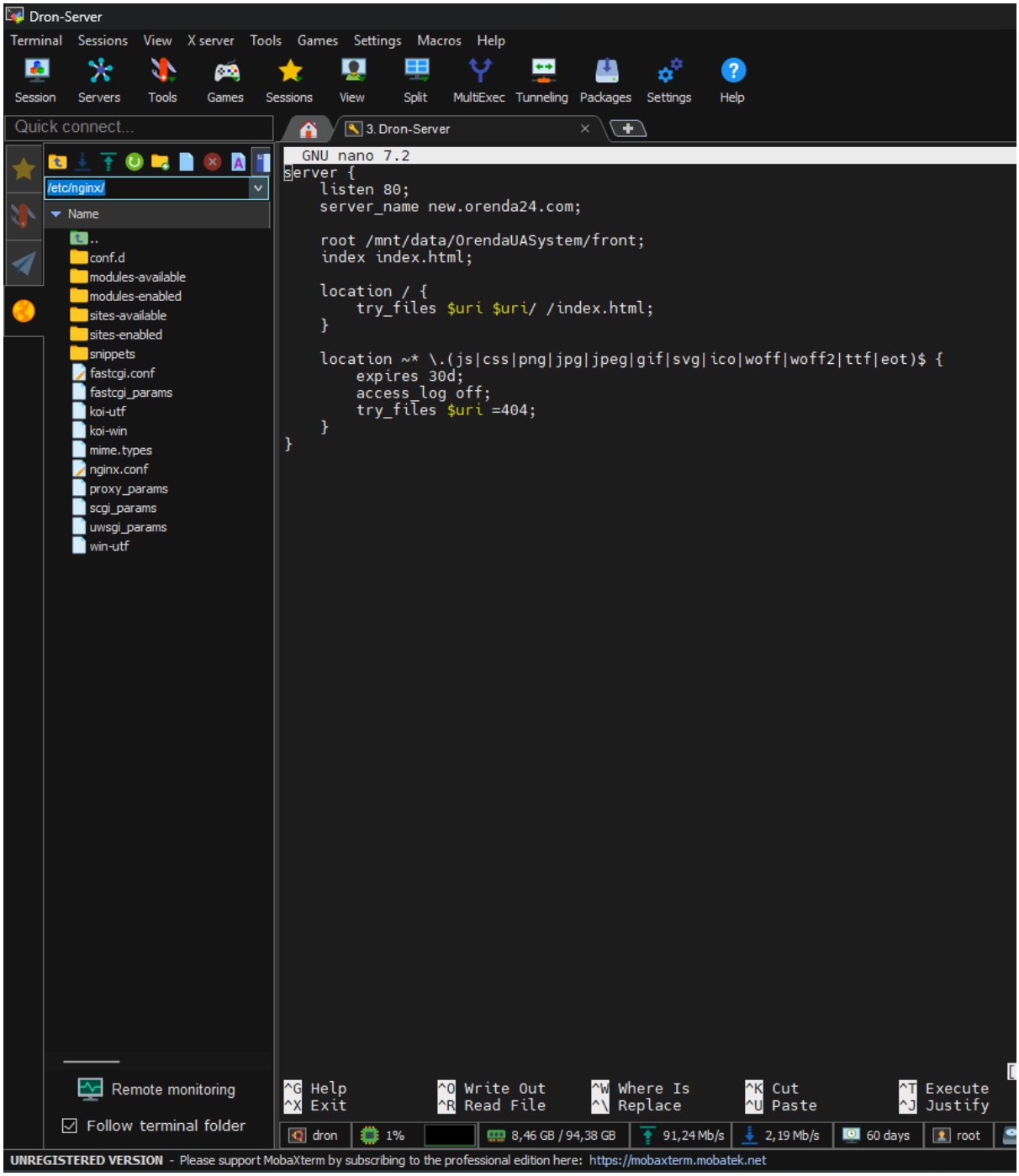

Рисунок 4.7 – Конфігурація веб-сервера nginx для роботи клієнтської частини

Рисунок 4.8 – Навантаження сервера з запущеним на ньому веб-додатком

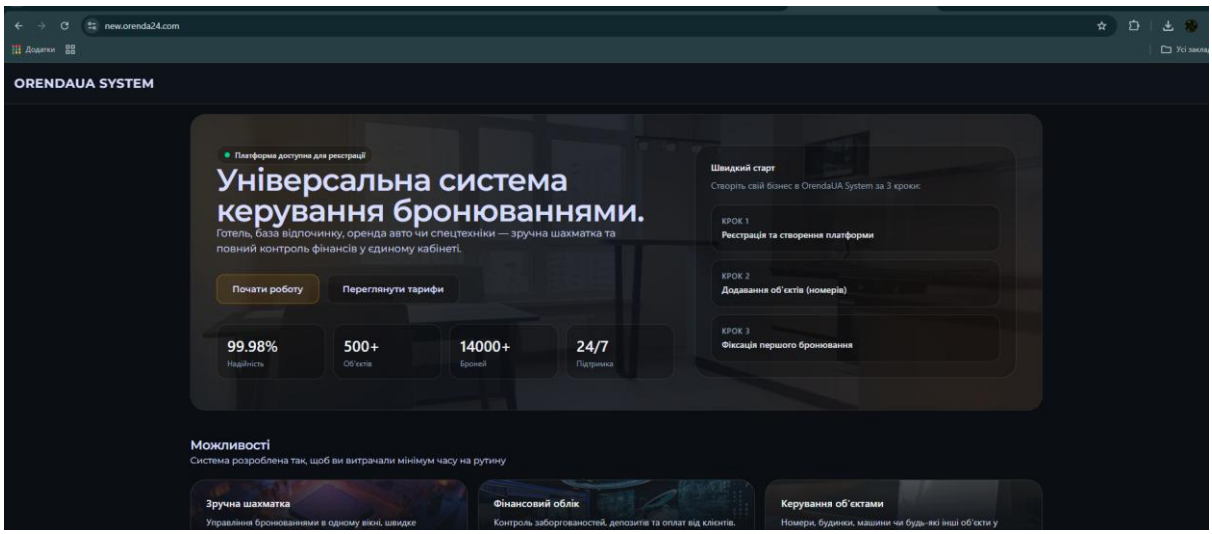

Рисунок 4.9 – Веб додаток успішно запущен на домені new.orenda24.com

Висновки до розділу 4

У четвертому розділі було підсумовано результати комплексного тестування, яке охопило функціональні, безпекові та нефункціональні аспекти вебсервісу OrendaUA. Функціональне тестування підтвердило коректність виконання всіх CRUD-операцій, а також абсолютну надійність критичної бізнес-логіки бронювань, зокрема, запобігання колізіям через використання

ізольованих транзакцій PostgreSQL. Тестування безпеки засвідчило ефективність гібридної моделі доступу, де Guards на бекенді коректно блокують несанкціоновані запити, гарантуючи повну ізоляцію даних між різними платформами. Навантажувальне тестування та аналіз продуктивності підтвердили, що середній час відгуку API відповідає нефункціональним вимогам. В цілому, результати тестування дозволили оцінити вебсервіс як високопродуктивну, надійну та безпечну систему, готову до промислової експлуатації.

ВИСНОВКИ

У кваліфікаційній роботі розроблено та досліджено вебсервіс OrendaUA - платформу для автоматизації обліку оренди нерухомості з інтерактивною візуалізацією бронювань та гнучким розмежуванням прав доступу.

У ході роботи виконано такі завдання:

Проаналізовано ринок існуючих рішень та визначено основні проблеми ручного обліку - розсинхронізацію даних, ризик овербукінгу та відсутність контролю над персоналом. На основі цього аналізу сформовано вимоги до системи та обрано технологічний стек: Nest.js для серверної частини, PostgreSQL як основна СУБД з підтримкою ACID-транзакцій, Prisma ORM для типобезпечної роботи з даними та Vue.js для клієнтської частини.

Розроблено та реалізовано ключовий функціонал системи - інтерактивну календарну сітку бронювань з кольоровим кодуванням статусів, що дозволяє менеджеру миттєво оцінювати стан фонду та створювати бронювання безпосередньо на сітці.

Впроваджено гібридну модель контролю доступу, яка забезпечує чітку ізоляцію даних між незалежними агентствами та точне дотримання принципу найменших привілеїв. Захист облікових записів реалізовано через JWT-автентифікацію, хешування паролів bcrypt та двофакторну автентифікацію.

Проведено комплексне тестування, яке підтвердило коректність бізнес-логіки, надійність механізму захисту від овербукінгу та ефективність підсистеми розмежування прав доступу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Nest.js – A progressive Node.js framework for building efficient, reliable and scalable server-side applications. 2024. URL: https://docs.nestjs.com/ (дата звернення: 10.04.2026).

2. Vue.js – The Progressive JavaScript Framework. v3.x Documentation. 2024. URL: https://vuejs.org/ (дата звернення: 15.04.2026).

3. PostgreSQL – The World's Most Advanced Open Source Relational Database. Documentation. 2025. URL: https://www.postgresql.org/docs/ (дата звернення: 20.04.2026).

4. Prisma – Next-generation ORM for Node.js and TypeScript. Documentation. 2025. URL: https://www.prisma.io/docs/ (дата звернення: 25.04.2026).

5. IETF RFC 7519: JSON Web Token (JWT). The Internet Engineering Task Force. 2015. URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 01.05.2026).

6. Sandhu R., Bhamidipati V., Munawer Q. The NIST Model for Role-Based Access Control: Toward a Unified Standard. ACM Transactions on Information and System Security. 1999. Vol. 2(1). P. 20–51.

7. IEEE Std 29148-2018. ISO/IEC/IEEE International Standard – Systems and software engineering – Life cycle processes – Requirements engineering. 2018. 106 p.

8. Open Source bcrypt implementation. 2024. URL: https://github.com/kelektiv/node.bcrypt.js (дата звернення: 05.05.2026).

9. Tailwind CSS – Utility-First CSS Framework. Documentation. 2024. URL: https://tailwindcss.com/docs

10. Fielding, R. T. Architectural Styles and the Design of Network-based Software Architectures. Ph.D. dissertation, University of California, Irvine, 2000. URL: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

11. Swagger Documentation. SmartBear Software. 2025. URL: https://swagger.io/docs/

12. MobaXterm – Terminal with SSH, X11, RDP and SFTP clients. Documentation. Mobatek. 2025. URL: https://mobaxterm.mobatek.net/documentation.html

13. Ubuntu Server documentation. Canonical Ltd. 2025. URL: https://ubuntu.com/server/docs

ДОДАТКИ

Додаток А – Посилання на репозиторій

Програмний код вебплатформи, розробленої в межах виконання кваліфікаційної роботи, розміщено у віддаленому репозиторії за адресою: https://github.com/OleksandrEkorde/csbc-diploma-project