

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ ДЕРЖАВНИЙ
УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА» ФАКУЛЬТЕТ
ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ КАФЕДРА
ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Платформа оренди авто»

Виконав студент групи ВТ-22-1

СЕРЕДА Іван Романович

Керівник роботи:

ЄФРЕМОВ Юрій Миколайович

Рецензент:

ГРОМСЬКИЙ Олександр Олександрович

Житомир - 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ ДЕРЖАВНИЙ
УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА» ФАКУЛЬТЕТ
ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ КАФЕДРА
ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна Вакалюк

«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **СЕРЕДА Іван Романович**

Керівник роботи: **ЄФРЕМОВ Юрій Миколайович**

Тема роботи: **«Платформа оренди авто»,**

Затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., №92/с**

Вихідні дані для роботи: **Об'єктом дослідження є цифрові технології для розробки та розгортання платформи оренди авто.**

Предметом дослідження є алгоритми, методи та технології проектування, розробки, розгортання та тестування платформи для оренди автомобілів, аналітичної звітності та управління автопарком.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Єфремов Ю.М.	27.02.2026р	27.02.2026р
2	Єфремов Ю.М.	27.02.2026р	27.02.2026р
3	Єфремов Ю.М.	27.02.2026р	27.02.2026р

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з пояснювальної записки та платформи для оренди авто. Пояснювальна записка містить 85 сторінок тексту, 33 рисунки, 29 таблиць та 7 сторінок додатків. Список використаних джерел складається з 30 позицій на 3 сторінках. Загальний обсяг роботи – 92 сторінки.

Метою роботи є розробка сучасної платформи для оренди автомобілів, що забезпечує клієнтам зручний пошук, бронювання та безпечну оплату транспортних засобів, а власникам автопарків - цифрове середовище для управління бізнесом, персоналом та фінансовою аналітикою.

В основу архітектури розроблюваного сервісу покладено концепцію клієнт-серверного REST API. Практичне проєктування та кодування виконано на базі такого стеку: Next.js + Redux Toolkit (клієнтська частина) та NestJS + TypeScript (серверна частина). Моделювання реляційного сховища виконано в СУБД PostgreSQL через абстракцію ORM Sequelize. Системна інфраструктура та автоматичне розгортання платформи сконфігуровані в Docker і Docker Compose під керуванням вебсервера Nginx із підтримкою CI/CD. Фінальне програмне забезпечення орієнтоване на автоматизацію обліку замовлень автопрокату, операційне управління машинами партнерів та обробку онлайн-еквайрингу за допомогою Stripe.

КЛЮЧОВІ СЛОВА: ОРЕНДА, NEXT.JS, NESTJS, POSTGRESQL, API, МОДЕЛЬ, DOCKER.

					ІПЗ.КР.Б – 121 – 26 – ПЗ						
Змн.	Арк.	№ докум.	Підпис	Дата							
Розроб.		Середа І.Р.			Платформа оренди авто			Літ.	Арк.	Аркушів	
Керівник		Єфремов Ю.М.								3	92
Рецензент		Громський О.О.			Пояснювальна записка			Житомирська політехніка, група ВТ-22-1			
Зав. каф.		Вакалюк Т.А.									

ABSTRACT

The bachelor's qualification thesis consists of an explanatory note and a car rental platform. The explanatory note contains 85 pages of text, 33 figures, 29 tables, and 7 pages of appendices. The list of references comprises 30 sources across 3 pages. The total volume of the work is 92 pages.

The aim of the work is to develop a modern car rental platform that provides clients with convenient search, booking, and secure payment for vehicles, and fleet owners with a digital environment for managing their business, staff, and financial analytics.

The architecture of the developed service is based on the concept of a client-server REST API. The practical design and coding were carried out using the following stack: Next.js + Redux Toolkit (client side) and NestJS + TypeScript (server side). The relational data storage was modeled in the PostgreSQL DBMS through the Sequelize ORM abstraction. The system infrastructure and automated deployment of the platform are configured in Docker and Docker Compose under the management of the Nginx web server with CI/CD support. The final software is aimed at automating car rental order management, operational control of partners' vehicles, and processing of online acquiring via Stripe.

KEYWORDS: RENTAL, NEXT.JS, NESTJS, POSTGRESQL, API, MODEL, DOCKER.

					ИПЗ.КР.Б – 121 – 26 – ИПЗ	Арк.
						4

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	7
ВСТУП	8
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБҐРУНТУВАННЯ ЗАСОБІВ РОЗРОБКИ ПЛАТФОРМИ ОРЕНДИ АВТОМОБІЛІВ	10
1.1. Аналіз предметної області та формулювання завдань автоматизації процесів у системі автопрокату	10
1.2. Аналіз аналогів програмного продукту	14
1.3. Вибір архітектури платформи та загальна структура майбутнього програмного продукту	21
1.4. Обґрунтування вибору інструментальних засобів	26
Висновки до першого розділу.....	30
РОЗДІЛ 2. ПРОЄКТУВАННЯ ПЛАТФОРМИ ДЛЯ ОРЕНДИ АВТО	32
2.1. Визначення варіантів використання та акторів	32
2.2. Проєктування бази даних та архітектури даних	42
2.3. Проєктування та реалізація алгоритмів роботи додатку	52
Висновки до другого розділу	62
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ПЛАТФОРМОЮ ОРЕНДИ АВТО	63
3.1. Послідовність розгортання та оптимізації параметрів платформи.....	63
3.2 Організація інтерфейсу та порядок роботи з платформою.....	67
3.3 Тестування платформи	76
Висновки до третього розділу.....	81
ВИСНОВКИ.....	82

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	83
ДОДАТКИ.....	86
Додаток А. «Діаграма бази даних».....	87
Додаток В. Лістинг сервісу автентифікації	88
Додаток С. Діаграма активностей «Алгоритм рольового розмежування доступу»	89
Додаток D. Діаграма активностей «Алгоритм багатовимірної фільтрації автомобілів»	90
Додаток Е. Лістинг алгоритму фільтрації з перевіркою доступності.....	91
Додаток F. Діаграма компонентів «Архітектура інтеграції Swagger UI у процес solo-розробки»	92

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API - Інтерфейс прикладного програмування (Application Programming Interface);

CSR - Клієнтський рендеринг сторінок (Client-Side Rendering);

JWT - Компактний стандарт токенів безпеки (JSON Web Token);

REST - Архітектурний стиль взаємодії компонентів розподіленого застосунку (Representational State Transfer);

SSR - Серверний рендеринг сторінок (Server-Side Rendering);

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						7

ВСТУП

Актуальність теми. У сучасному світі, де динаміка подій та свобода пересування стали базовими потребами людини, мобільність виходить на перший план. На фоні розвитку інтернет-технологій, люди очікують комфортних та прозорих логістичних процесів. Завдяки цифровізації спрощується чимало аспектів життєдіяльності, зокрема оренда майна. Сьогодні процес оренди авто еволюціонує до приємного, прозорого та швидкого цифрового досвіду, який, доступний кожному, хто має доступ до мережі.

Розробка цифрової системи для оренди та прокату авто, сьогодні є надзвичайно актуальною. Такі платформи дозволяють бізнесу перейти в онлайн-середовище, забезпечуючи автоматизацію та економію операційного ресурсу. Це сприяє розширенню охоплення аудиторії та дає можливість спрямовувати клієнта з рекламних каналів безпосередньо у функціональний додаток. У результаті користувач за кілька кроків отримує достовірну інформацію про умови страхування та переваги сервісу, що скорочує тривалість угоди та підвищує ефективність бізнесу.

З іншого боку, для власників автопарків це потужний інструмент звільнення від рутинної паперової роботи. Оператори можуть централізовано керувати активами: формувати звітність, аналізувати прибутки, віддалено відстежувати технічний стан транспортних засобів та планувати їх обслуговування. Проектування та впровадження таких комплексних рішень є ключовим етапом модернізації галузі. Це дозволяє замінити застарілу ручну працю інтелектуальними алгоритмами управління, створюючи фундамент для стабільного розвитку в умовах глобальної цифровізації.

Метою роботи є створення платформи для оренди авто «Drivefy». Ключове призначення системи - розробка зручного та функціонального додатку, який дозволить користувачам легко орендувати авто, а власникам автопарку - ефективно відстежувати стан транспортних засобів,

автоматизувати управління замовленнями та оптимізувати операційні витрати за допомогою централізованої панелі адміністрування.

Завдання роботи. Для досягнення поставленої мети необхідно виконати такі завдання:

- здійснити аналіз предметної області та існуючих рішень у галузі надання автомобілів в оренду;
- сформулювати вимоги до розроблюваної системи;
- визначити архітектуру, узагальнити структуру та ключові компоненти додатку;
- обґрунтувати вибір засобів реалізації програмного продукту;
- спроектувати структуру баз даних для зберігання даних про автомобілі, користувачів, компанії, історію оренди та платежі;
- створити платформу із застосуванням обраного інструментарію на основі визначених вимог;
- розробити зручний інтерфейс як для клієнтів, так і для власників та адміністраторів автопарків;
- розгорнути платформу на віддаленому сервері;
- протестувати розроблений програмний продукт для перевірки його працездатності та відповідності заявленим вимогам.

Об'єктом дослідження є цифрові технології для розробки та розгортання платформи оренди авто.

Предметом дослідження є алгоритми, методи та технології проектування, розробки, розгортання та тестування платформи для оренди автомобілів, аналітичної звітності та управління автопарком.

Практичне значення одержаних результатів полягає у створенні повноцінного, готового до впровадження програмного продукту, який здатний оптимізувати бізнес-процеси реальних компаній з прокату автомобілів.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБҐРУНТУВАННЯ ЗАСОБІВ РОЗРОБКИ ПЛАТФОРМИ ОРЕНДИ АВТОМОБІЛІВ

1.1. Аналіз предметної області та формулювання завдань автоматизації процесів у системі автопрокату

Сучасний етап розвитку суспільства характеризується стрімкою зміною парадигми мобільності. Володіння власним транспортним засобом поступово перестає бути єдиним чи навіть найзручнішим способом пересування, поступаючись місцем концепції «автомобіль як послуга» (Car-as-a-Service). На тлі загальної цифровізації економіки та зростання вимог споживачів до швидкості отримання послуг, ринок оренди автомобілів потребує глибокої технологічної трансформації. Традиційні моделі прокату, обтяжені паперовою тяганиною, тривалими процедурами перевірки та непрозорим ціноутворенням, більше не відповідають очікуванням сучасного користувача. Відтак, виникає гостра необхідність у розробці комплексних цифрових рішень, які б об'єднали клієнтів та надавачів послуг у єдиній, зручній та безпечній екосистемі.

Головною задачею цього проєкту є розробка багатофункціональної платформи для оренди автомобілів «Drivefy», яка працюватиме як у сегменті B2C (бізнес для споживача), так і в B2B (бізнес для бізнесу). Програмне рішення має функціонувати як SaaS-продукт (програмне забезпечення як послуга), надаючи користувачам не лише інструменти для пошуку та бронювання авто, але й просунуте середовище для створення та управління власним орендним бізнесом.

Для забезпечення ефективної взаємодії всіх учасників процесу, система повинна підтримувати чітку рольову модель доступу (RBAC), яка включає п'ять основних типів акторів: «Гість» (Guest), «Користувач» (Client), «Власник компанії» (Owner), «Адміністратор» (Admin) та «Співробітник» (Employee). Кожна з ролей вимагає розробки специфічного функціоналу та унікального користувацького інтерфейсу.

На початковому етапі взаємодії платформа має надавати базовий функціонал для неавторизованої ролі «Гість». Такий відвідувач повинен мати безперешкодний доступ до перегляду публічного каталогу автомобілів, можливість використовувати розширену систему фільтрації для пошуку вільних авто на потрібні дати, а також ознайомлюватися з інформацією про компанії-орендодавці та їхні пункти видачі. Проте функціонал бронювання автомобіля для цієї ролі є заблокованим, що спонукає гостя до створення облікового запису.

Для отримання доступу до повного циклу послуг необхідна роль авторизованого клієнта - «Користувач». На цьому рівні платформа має забезпечувати зрозумілий процес від моменту реєстрації та автентифікації до оформлення і завершення поїздки. Критично важливою вимогою є інтеграція надійного платіжного шлюзу (зокрема, Stripe) для безпечного проведення транзакцій, збереження платіжних методів та формування прозорої історії фінансових витрат користувача. Крім того, система повинна підтримувати соціальну складову - можливість залишати та редагувати відгуки після завершення оренди, що формуватиме рейтинг довіри на платформі.

Важливою функціональною вимогою до платформи є можливість безшовної трансформації звичайного користувача у бізнес-партнера. Створивши власну компанію, користувач отримує роль «Власника» та доступ до спеціалізованого середовища - панелі управління автопарком. У цьому контексті задача розширюється до створення повноцінної ERP-системи (управління ресурсами підприємства) для автопарку. Власник повинен мати інструменти для масштабування бізнесу: додавання нових пунктів видачі, управління реєстром автомобілів, а також ведення суворого технічного обліку.

Особливу увагу слід приділити розробці алгоритмів автоматичного блокування доступності транспортних засобів. Система має самостійно виключати автомобіль із пошукової видачі та унеможливити його бронювання, якщо термін дії його страхового поліса вичерпано, або транспортний засіб перебуває на плановому технічному обслуговуванні. Це

мінімізує юридичні ризики та людський фактор. Також робочий простір власника має включати аналітичну панель для моніторингу ключових бізнес-показників: загального обороту, чистого доходу, структури витрат (пальне, клінінг, ремонт тощо), а також візуалізації цих даних за допомогою інтерактивних графіків.

Для забезпечення операційної діяльності компаній, система повинна передбачати ієрархію делегування повноважень шляхом створення ролей «Адміністратора» та «Співробітника». Адміністратори відповідатимуть за тактичне управління: розширення автопарку, найм співробітників, внесення даних про страхування та технічному обслуговуванні. Лінійні співробітники, своєю чергою, потребують швидких та безвідмовних інструментів для фізичної видачі та прийому автомобілів. З метою прискорення цих логістичних операцій, система має генерувати та зчитувати унікальні QR-коди або ідентифікатори для кожного бронювання. Це дозволить оператору за лічені секунди ідентифікувати замовлення, передати авто клієнту, а під час повернення - оперативно зафіксувати стан машини, додати супутні витрати (наприклад, послуги мийки) або виписати штраф у разі порушення умов договору.

Процес реалізації такого комплексного застосунку вимагає проходження низки структурованих інженерних етапів.

На першому етапі необхідно спроектувати загальну архітектуру системи. Оскільки платформа розрахована на високі навантаження та вимагає відмінної пошукової оптимізації (SEO) для сторінок каталогу, доцільним є використання фреймворку Next.js для клієнтської частини, що забезпечить серверний рендеринг - SSR [1][2][3]. Для серверної частини (backend) необхідно застосувати NestJS. Прогресивний фреймворк, базований на Node.js, який дозволить побудувати масштабоване, модульне та легко тестоване API.

Другий етап включає проєктування реляційної бази даних. Структура повинна охоплювати велику кількість взаємопов'язаних сутностей: профілі

користувачів, дані компаній, характеристики транспортних засобів, локації пунктів видачі, лог бронювань, фінансові транзакції, графіки технічного обслуговування та історію відгуків. Архітектура бази даних має забезпечувати цілісність інформації та швидкість виконання складних запитів, особливо під час фільтрації вільних автомобілів за заданими датами.

Третій етап полягає у розробці інтерфейсу користувача (UI/UX). Візуальне рішення має бути сучасним, мінімалістичним та адаптивним, забезпечуючи однаково високий рівень комфорту як при роботі зі смартфона під час швидкого бронювання авто, так і при роботі з десктопними моніторами під час аналізу фінансових графіків власником автопарку.

На четвертому етапі здійснюється програмна реалізація бізнес-логіки, налаштування маршрутизації, інтеграція сторонніх сервісів (Stripe для платежів, хмарних сховищ для завантаження фотографій користувачів та авто) та імплементація механізмів генерації QR-кодів [4].

Завершальним етапом є комплексне тестування розробленої системи, яке поєднує автоматизовані та ручні підходи для забезпечення надійності програмного продукту. Насамперед застосовується модульне (unit) тестування, що дозволяє ізольовано перевірити коректність виконання базової бізнес-логіки та роботу окремих компонентів платформи на рівні коду. Наступним кроком є ретельне ручне (manual) тестування, під час якого крок за кроком відтворюються реальні користувацькі сценарії [5]. Особлива увага під час цього процесу приділяється перевірці коректності роботи рольової моделі - здійснюється строгий контроль меж доступу для кожної з п'ятих передбачених ролей системи, щоб повністю унеможливити несанкціоновані дії та гарантувати безпечну взаємодію користувачів із платформою перед її фінальним релізом.

Підсумовуючи, результатом виконання поставленої задачі має стати розгортання високотехнологічної, безпечної та масштабованої платформи «Drivefy». Цей програмний продукт має повністю покрити потреби обох цільових аудиторій: надати кінцевим споживачам прозорий інструмент для

забезпечення особистої мобільності, а власникам бізнесу - ефективний цифровий інструмент для автоматизації рутини, зниження операційних витрат та максимізації прибутковості автопарку.

1.2. Аналіз аналогів програмного продукту

Створення будь-якого інноваційного цифрового рішення вимагає глибокого розуміння поточного стану ринку та детального вивчення вже існуючих продуктів. Аналіз конкурентів є фундаментальним етапом, який дозволяє не лише оцінити рівень технологічного розвитку галузі, а й виявити концептуальні обмеження та реальні бізнес-переваги в наявних моделях.

Сучасний український ринок оренди автомобілів розвивається у кількох паралельних напрямках: високотехнологічний каршерінг, класичний багатопрофільний прокат та інноваційні P2P-платформи (peer-to-peer). Для комплексного дослідження було обрано чотири репрезентативні сервіси: Getmancar, NarsCars, 7Cars та «Їду сам». Вивчення специфіки цих платформ дає змогу чітко ідентифікувати незадоволені потреби як орендарів, так і власників бізнесу, щоб закласти їх вирішення в основу нової архітектури. Їхній аналіз проведено з фокусом не лише на клієнтський досвід (B2C), а й на бізнес-орієнтовані можливості, функціональну архітектуру та операційну ефективність, щоб сформулювати вимоги до майбутнього продукту.

Сервіс Getmancar є першопрохідцем та найбільшим гравцем на українському ринку каршерінгу [6]. На відміну від класичного прокату, бізнес-модель компанії базується на концепції короткострокового спільного використання автомобілів та глибокій інтеграції програмного забезпечення з апаратним рівнем (IoT-телематикою). Хоча такий підхід забезпечує високий рівень автоматизації, повністю закрита корпоративна екосистема робить цю модель непридатною для масштабування та використання сторонніми автопарками у форматі SaaS. Кожен транспортний засіб оснащений спеціальними модулями, що дозволяють клієнтам віддалено розблокувати двері через Bluetooth або інтернет-з'єднання, а самій компанії - здійснювати GPS-трекінг та контроль рівня пального в реальному часі. Взаємодія з

					ІІЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						14

користувачем доведена до повної автоматизації: від онлайн-верифікації документів до безпосереднього початку поїздки, що виключає необхідність залучення менеджерів на локації (див. рис. 1.1).

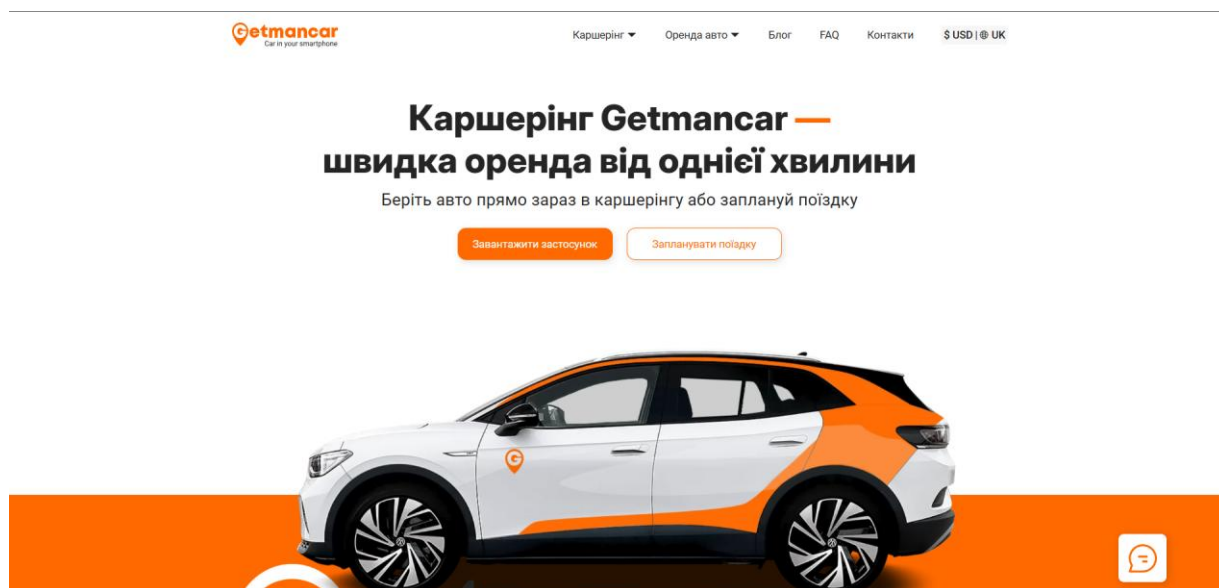


Рис. 1.1 - Інтерфейс Getmancar

Незважаючи на високий технологічний рівень, система має суворі архітектурні обмеження. Це повністю закрита корпоративна B2C-екосистема, яка обслуговує виключно власний автопарк компанії і не надає стороннім підприємцям інструментів для реєстрації власного бізнесу на їхній базі. Такий монопольний підхід унеможливорює розвиток партнерської мережі та перетворює платформу на інструмент обслуговування виключно однієї компанії, а не повноцінний маркетплейс. Більше того, фокус на мобільному додатку позбавляє користувачів можливості зручного пошуку та бронювання через повноцінну вебверсію, що є критичним для клієнтів, які планують довготривалі подорожі з десктопних пристроїв. Масштабування такої моделі також ускладнене високою вартістю закупівлі та інтеграції телематичного обладнання для кожного окремого автомобіля.

Сервіс NarsCars є яскравим представником класичної моделі компанії з прокату автомобілів [7]. Платформа функціонує як розгорнута цифрова вітрина власного масивного автопарку, пропонуючи автомобілі у різних цінових сегментах - від бюджетних міських варіантів до преміум класу.

Вебсайт дозволяє користувачам ознайомитися з характеристиками транспорту та забронювати модель на термін від кількох діб до місяців (див. рис. 1.2).

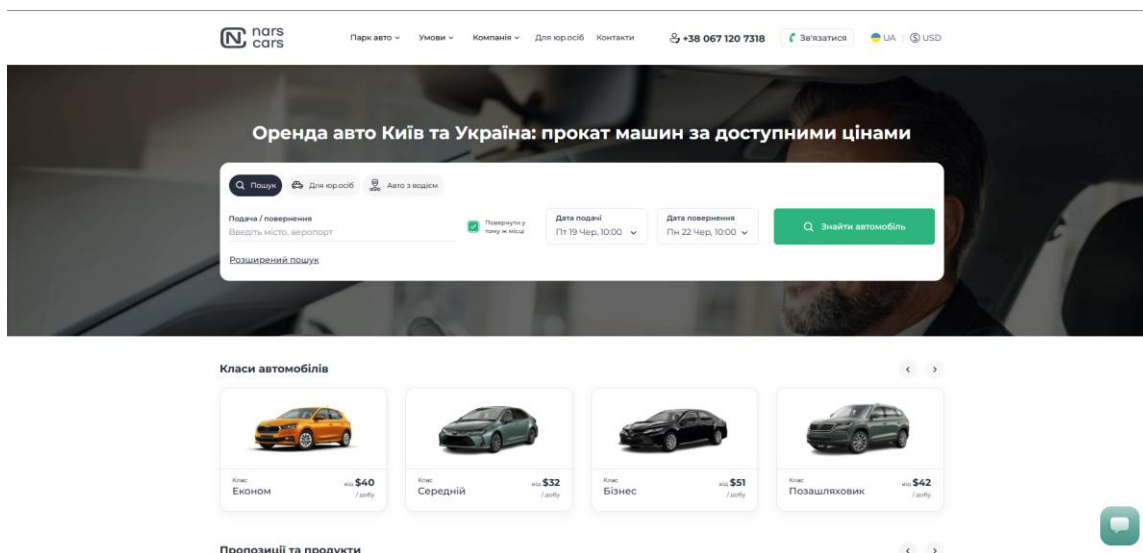


Рис. 1.2 - Інтерфейс NarsCars

Значним технологічним кроком компанії стала інтеграція з державним застосунком «Дія», що суттєво оптимізувало процедуру ідентифікації клієнтів, забезпечивши швидку та юридично безпечну верифікацію водійських документів онлайн. Однак, рівень цифровізації фінансових та операційних процесів залишається частковим. Платформа не підтримує завершеного циклу онлайн-оплати: користувачі не можуть повністю оплатити оренду та заставу безпосередньо на сайті, що часто вимагає подальшої комунікації з кол-центром або розрахунку на місці. З позиції бізнес-архітектури, програмне забезпечення NarsCars вирішує виключно внутрішні завдання компанії. Воно не побудоване за принципом маркетплейсу, де клієнт міг би порівняти пропозиції від різних орендодавців, і не має відкритої SaaS-інфраструктури, яку могли б використовувати сторонні власники автопарків для управління своїми активами.

Платформа 7Cars займає специфічну нішу комплексних корпоративних та VIP-послуг у сфері транспорту [8]. Компанія вийшла за межі класичного прокату без водія, диверсифікувавши свої пропозиції: від обслуговування делегацій та організації трансферів до надання автомобілів у довгостроковий оперативний лізинг для юридичних осіб (B2B). Вебсайт компанії

оптимізований під детальне представлення цих послуг, проте оформлення угоди є здебільшого консервативним (див. рис. 1.3).

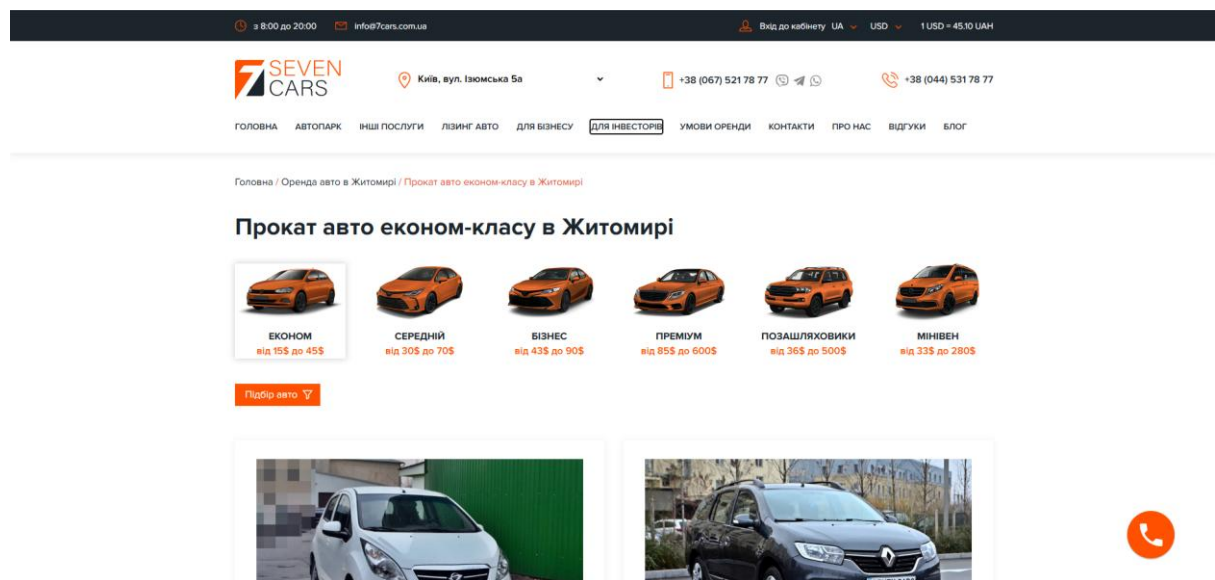


Рис. 1.3 - Інтерфейс 7Cars

Обов'язкова фізична ідентифікація орендаря співробітником та підписання паперових договорів гарантують максимальний рівень безпеки угод, що є критично важливим для автомобілів преміумкласу. Зворотною стороною такого консервативного підходу є низький рівень диджиталізації внутрішньої рутини. Відсутність автоматизованих ERP-систем змушує менеджерів компанії опрацьовувати значний обсяг паперової та адміністративної роботи вручну, що уповільнює процес видачі авто. Як і NarsCars, 7Cars є закритою системою, що просуває пропозиції лише однієї компанії, обмежуючи користувача в можливості об'єктивного цінового порівняння на єдиному майданчику.

Проект «Їду сам» функціонує за децентралізованою P2P-моделлю (peer-to-peer), нагадуючи за своєю бізнес-логікою сервіс Airbnb у вертикалі автопрокату [9]. Не володіючи власним автопарком, система виступає як цифровий маркетплейс, що забезпечує пряму взаємодію між фізичними особами. У межах цієї екосистеми приватні автовласники можуть гнучко розміщувати машини в каталозі, самостійно формувати тарифи й отримувати пасивний дохід, тоді як орендарі отримують доступ до диверсифікованого асортименту авто (див. рис. 1.4).

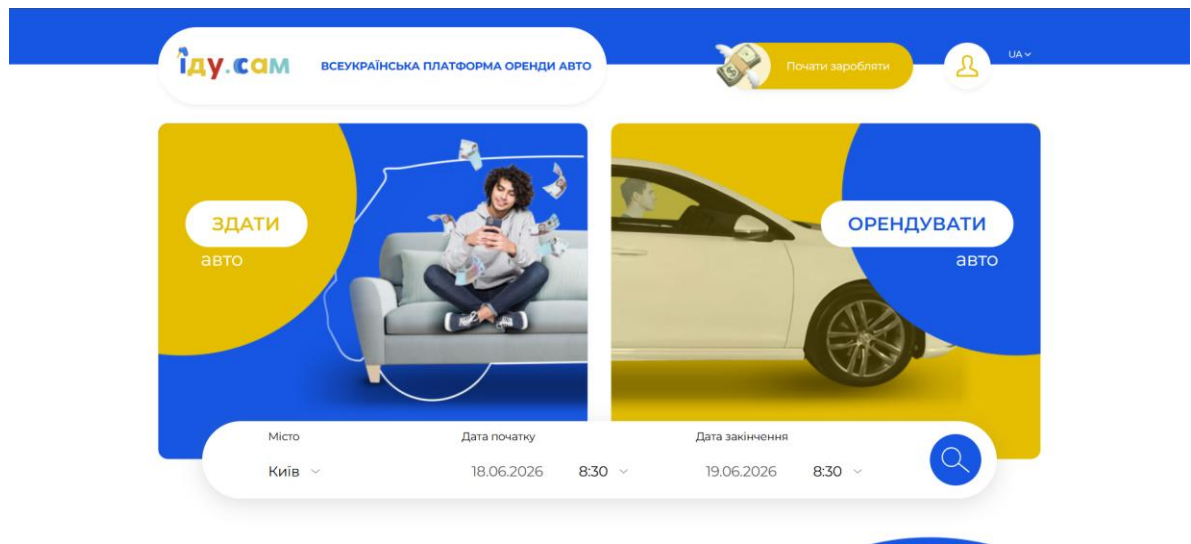


Рис. 1.4 - Інтерфейс «Їду сам»

Такий підхід забезпечує величезну різноманітність авто, широке географічне покриття та природну цінову конкуренцію. Баланс довіри в екосистемі підтримується завдяки вбудованій системі взаємних рейтингів та відгуків. Попри очевидні переваги для приватних осіб, платформа має суттєві обмеження для корпоративного використання. Її архітектура розрахована виключно на модель C2C (споживач для споживача) і не містить професійних інструментів для середнього чи великого бізнесу. Компанії-орендодавці не мають можливості створювати ієрархічні структури персоналу, додавати пункти видачі, вести фінансову аналітику чи автоматизовувати технічний контроль. Більше того, децентралізація призводить до відсутності єдиних стандартів якості: технічний стан автомобіля та загальний рівень сервісу повністю залежать від сумлінності конкретного власника.

Таблиця 1.1

Переваги та недоліки існуючих сервісів оренди автомобілів

Сервіс	Основні переваги	Основні недоліки
Getmancar	Інтеграція IoT-телематики; автоматизована верифікація; похвилинна тарифікація.	Відсутність повноцінної вебверсії; закритість платформи для інших бізнесів; вартість масштабування.
NarsCars	Інтеграція із застосунком «Дія»; широкий асортимент автопарку; калькулятор ціни.	Неможливість повністю оплатити послуги онлайн; відсутність формату маркетплейсу; замкнута архітектура.

продовж. табл. 1.1

Сервіс	Основні переваги	Основні недоліки
7Cars	Диверсифікація (оренда з водієм, лізинг); високий рівень безпеки угод; орієнтація на B2B.	Висока частка ручної роботи; відсутність ERP-системи; відсутність конкурентів на сайті.
«Їду сам»	Формат маркетплейсу; можливість доходу для фізичних осіб; система рейтингів.	Відсутність B2B-інструментів для автопарків; неможливість гарантувати єдиний стандарт якості.

Проведений порівняльний аналіз чинних рішень на ринку автопрокату дозволяє виявити ключові тенденції та технологічні обмеження сучасних систем (див. табл. 1.1). Більшість представлених сервісів мають закриту архітектуру і спрямовані на управління власним автопарком, що унеможлиблює масштабування бізнесу або інтеграцію сторонніх компаній. Водночас наявні P2P-платформи, хоч і пропонують формат маркетплейсу, не мають необхідних B2B-інструментів та засобів автоматизації для повноцінного ведення бізнесу, що підтверджує актуальність розробки гнучкої SaaS-платформи «Drivefy».

Для більш глибокого розуміння майбутнього позиціонування розроблюваного продукту на ринку, було сформовано матрицю функціональних та бізнес-орієнтованих можливостей (див. табл. 1.2).

Таблиця 1.2

Порівняння бізнес-орієнтованих можливостей систем оренди авто

Функціональна можливість / Сервіс	Getmancar	NarsCars	7Cars	«Їду сам»	Розроблювана платформа
Інтеграція IoT-телематики (віддалене відкриття авто)	+	-	-	-	-
Спрощена онлайн-верифікація (у т.ч. через застосунок «Дія»)	+	+	-	-	-
Повноцінна онлайн-оплата бронювання	Тільки у додатку	-	Частково	+	+

Функціональна можливість / Сервіс	Getmancar	NarsCars	7Cars	«Їду сам»	Розроблювана платформа
Модель маркетплейсу (доступ до авто від різних компаній)	-	-	-	+	+
SaaS-ERP для бізнесу (робочий простір: доходи, витрати)	-	-	-	-	+
Розширена ієрархія прав для компаній (Власник, Адмін)	-	-	-	-	+
Цифровізація логістики (видача авто через сканування QR)	-	-	-	-	+
Автоматичне блокування доступності (при протермінованому ТО)	-	-	Невідомо	-	+
Облік витрат на рівні кожного автомобіля	-	Невідомо	-	-	+

Проведений аналіз демонструє, що український ринок цифрових рішень для оренди авто є суттєво фрагментованим, де кожен із сильних гравців фокусується на вузькій ніші. Getmancar інвестує в телематику та мобільний досвід короткострокових поїздок. NarsCars пропонує якісний класичний сервіс із передовими державними інтеграціями, проте обмежує користувачів у повноцінній онлайн-оплаті. 7Cars фокусується на преміальному обслуговуванні та корпоративному секторі. Платформа «Їду сам» успішно реалізовує P2P-підхід, але не має функціоналу для професійних компаній-орендодавців.

Розроблювана платформа оренди авто не матиме на меті конкурувати з каршерінгом у сфері IoT-апаратури чи замінити класичні лізингові послуги. Замість цього вона займе унікальну B2B2C-нішу. З одного боку, клієнти отримають повноцінний вебмаркетплейс із безпечною онлайн-оплатою, де

можна зручно порівнювати ціни від різних постачальників послуг. З іншого боку, система слугуватиме як потужне, готове SaaS-рішення (ERP-система) для малого та середнього бізнесу у сфері автопрокату.

Майбутня платформа свідомо збереже етап фізичної перевірки водійських прав співробітником та підписання договору для мінімізації юридичних ризиків власників. Проте цей процес буде кардинально прискорений завдяки впровадженню системи QR-кодів для логістики (швидкої ідентифікації та видачі авто). Відокремлені робочі простори, чітка ієрархія доступу для персоналу та автоматичні алгоритми блокування транспорту за статусом технічного обслуговування перетворять розроблювану платформу на повноцінний цифровий офіс, що дозволить підприємцям ефективно управляти власним автопарком.

1.3. Вибір архітектури платформи та загальна структура майбутнього програмного продукту

Вибір архітектури програмного продукту є стратегічним етапом, який визначає не лише поточну продуктивність розроблюваної платформи оренди авто, але й її здатність до масштабування, відмовостійкість та легкість подальшої підтримки. Враховуючи високі вимоги до інтерактивності інтерфейсу, необхідність роботи з фінансовими транзакціями та наявність складної рольової ієрархії, для реалізації проєкту було обрано клієнт-серверну архітектуру з використанням парадигми REST API (Representational State Transfer) [10].

Такий підхід передбачає повне розділення відповідальності між фронтендом (інтерфейсом користувача) та бекендом (серверною частиною обробки даних). Клієнтська та серверна частини взаємодіятимуть шляхом обміну даними у легкому форматі JSON через захищений протокол HTTP [11]. Це дозволяє розробляти та розгортати ці два компоненти незалежно один від одного, що суттєво знижує ризик виникнення критичних збоїв. Крім того, використання REST-архітектури без збереження стану оптимізує навантаження на сервер, оскільки кожен запит обробляється незалежно, що є

The diagram illustrates a Docker-based application architecture. At the top, a box labeled "Користувач / Браузер" (User / Browser) connects to a "Nginx Reverse Proxy" box within a "Docker Internal Network" container. The connection is labeled "HTTPS :443". The "Nginx Reverse Proxy" also receives "Асинхронні CSR запити" (Asynchronous CSR requests) from "Certbot / Let's Encrypt" (a dashed box). The proxy routes traffic to two main services: "Next.js Frontend SSR/CSR" and "NestJS Backend API". The "Next.js Frontend" receives "Proxy /" and "SSR API Запити" (SSR API requests). The "NestJS Backend API" receives "Proxy /api" and "Proxy /uploads". The "NestJS Backend API" is connected to "PostgreSQL" (via "SQL / Sequelize") and "Uploads Volume" (via "Filesystem"). Additionally, the "NestJS Backend API" sends "Платежі" (Payments) to the "Stripe API" (a dashed box). "SSL Сертифікати" (SSL Certificates) are provided by "Certbot / Let's Encrypt" to the "Nginx Reverse Proxy".

Фронтенд-частина розроблюваної платформи базуватиметься на фреймворку Next.js. Вибір цього інструменту зумовлений його здатністю гнучко поєднувати різні стратегії рендерингу. Для публічних сторінок (наприклад, каталогу автомобілів та сторінок компаній) буде застосовано серверний рендеринг (SSR). Це критично важливо для пошукової оптимізації (SEO), оскільки пошукові роботи отримуватимуть повністю сформований HTML-документ. З іншого боку, для приватних розділів системи (єдиного інтерактивного середовища Drivefy Workspace, доступ до модулів якого динамічно регулюється відповідно до ролі: власник, адміністратор чи співробітник) використовуватиметься клієнтський рендеринг (CSR) з подальшою гідратацією (Hydration). Це перетворить застосунок на Single Page

Application (SPA) у межах робочого простору, забезпечуючи миттєвий відгук інтерфейсу без повного перезавантаження сторінок.

Для ефективного управління глобальним станом застосунку та оптимізації мережевих запитів планується використання сучасного стеку Redux Toolkit у поєднанні з RTK Query [12][13]. Це дозволить автоматизувати процес кешування відповідей від сервера, значно зменшуючи кількість повторних запитів до API. Крім того, застосування механізмів персистенції (Redux Persist) забезпечить збереження нечутливих налаштувань інтерфейсу в локальній пам'яті браузера, зберігаючи контекст користувача навіть після закриття вкладки.

Серверна архітектура та бізнес-логіка розроблятиметься з використанням прогресивного Node.js фреймворку NestJS. Головною перевагою NestJS є його сувора типізація (завдяки TypeScript) та вбудована орієнтація на модульну, багат шарову архітектуру (MVC) [14][15][16]. Такий строгий розподіл обов'язків у поєднанні з механізмом впровадження залежностей (Dependency Injection) суттєво спрощує ізольоване тестування кожного окремого компонента. Крім того, це гарантує високу підтримуваність системи, оскільки зміни в одному рівні інфраструктури не вимагають переписування загальної бізнес-логіки застосунку. Увесь код сервера буде логічно розділено на незалежні модулі (модуль користувачів, модуль автомобілів, модуль бронювань тощо).

Архітектура серверної частини базується на чіткому розподілі обов'язків між трьома основними шарами, що забезпечує послідовну обробку даних (див. рис. 1.6). На першому етапі взаємодії контролери (Controllers) приймають вхідні HTTP-запити, здійснюють первинну валідацію отриманих даних та відповідають за маршрутизацію. Очищені та перевірені дані з контролерів передаються на рівень сервісів (Services), де зосереджено ядро бізнес-логіки системи - від алгоритмів перевірки доступності автомобілів за датами до обчислення складної фінансової статистики. Оскільки сервіси повністю ізольовані від деталей роботи з мережею, для безпосередньої комунікації з

базою даних вони звертаються до ORM-моделей (Data Access Layer). Цей шар об'єктно-реляційного відображення абстрагує складні SQL-запити до рівня методів класів, що дозволяє сервісам ефективно маніпулювати даними та забезпечує оптимізований інформаційний потік у межах платформи.

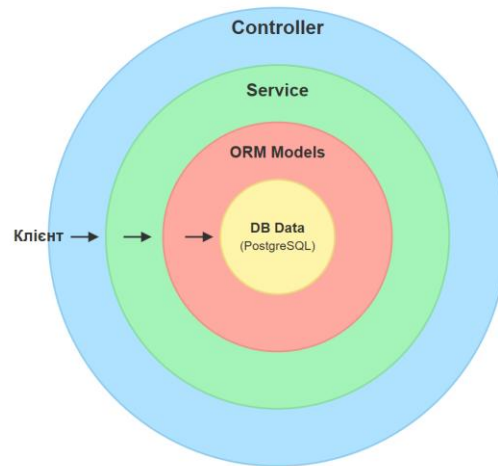


Рис. 1.6 - Модульна багатошарова архітектура серверної частини на базі NestJS

Основою для зберігання даних виступатиме реляційна система управління базами даних PostgreSQL [17]. Вибір реляційної моделі є безальтернативним для систем, що працюють з фінансами (інтеграція зі Stripe), розкладами бронювань та складною структурою компаній, оскільки PostgreSQL гарантує строгу консистентність даних (відповідність принципам ACID) та дозволяє будувати надійні зв'язки між таблицями.

Система безпеки розроблюваної платформи базуватиметься на стандартизованому механізмі JSON Web Tokens (JWT) у поєднанні з моделлю RBAC (Role-Based Access Control)[18][19]. Оскільки платформа передбачає використання єдиного простору Drivefy Workspace, RBAC дозволить на рівні серверних Middleware перевіряти права доступу до кожного конкретного ендпоінту API та динамічно обмежувати інтерфейс залежно від того, хто авторизувався (Користувач, Власник, Адміністратор чи Співробітник).

З метою протидії поширеним вебвразливостям, зокрема атакам типу XSS (міжсайтовий скриптинг), архітектура передбачає відмову від зберігання JWT-токенів у відкритому локальному сховищі браузера (LocalStorage)[20].

Натомість токени доступу передаватимуться сервером та зберігатимуться виключно у захищених файлах cookie з прапорцями HttpOnly та Secure, що унеможливить доступ до них з боку шкідливих JavaScript-скриптів на клієнті. Для забезпечення стабільності життєвого циклу програмного продукту вся архітектура буде побудована на принципах контейнеризації з використанням Docker, де кожен компонент системи запускатиметься у власному ізолюваному контейнері. Це повністю вирішує проблему «локальної працездатності» - продукт працюватиме ідентично на машині розробника, на тестовому сервері та у робочому (продакшн) середовищі.

Як точка входу для всього зовнішнього трафіку буде розгорнуто вебсервер Nginx, який виконуватиме роль зворотного проксі-сервера (Reverse Proxy) [20]. Nginx відповідатиме за термінацію SSL-сертифікатів (шифрування трафіку через HTTPS), балансування навантаження та перенаправлення запитів на відповідні Docker-контейнери (Next.js для клієнтських сторінок, NestJS для API). Оновлення коду та його доставка на сервери відбуватиметься автоматизовано за допомогою налаштованих конвеєрів CI/CD (безперервна інтеграція та розгортання), що дозволить оперативно випускати оновлення без переривання роботи сервісу[21][22](див. табл. 1.3).

Таблиця 1.3

Обґрунтування архітектурних рішень розроблюваної платформи

Компонент системи	Обрана технологія / Концепція	Технічне обґрунтування та переваги
Клієнтська архітектура	Next.js (SSR + CSR з Hydration)	Забезпечує SEO-оптимізацію для публічного каталогу та реактивність (SPA) для єдиного простору компанії (Workspace).
Керування станом	Redux Toolkit + RTK Query (із Persist)	Оптимізує кешування даних, суттєво зменшує кількість мережових запитів та зберігає критичний стан сесії користувача.
Архітектура сервера	NestJS (Модульна багаторівнева архітектура)	Чіткий розподіл на Контролери, Сервіси та ORM-моделі інкапсулює бізнес-логіку та полегшує підтримку кодової бази.

Компонент системи	Обрана технологія / Концепція	Технічне обґрунтування та переваги
Протокол взаємодії	REST API (JSON)	Створює універсальний, стандартизований інтерфейс для швидкої передачі даних між клієнтом та сервером.
База даних	PostgreSQL	Гарантує транзакційну цілісність, надійність та суворе дотримання реляційних зв'язків (оренда - авто - витрати - платежі).
Безпека та авторизація	JWT (HttpOnly Cookies) + RBAC	Захищає від XSS-атак та дозволяє гнучко керувати доступами всередині єдиного Drivefy Workspace.
Інфраструктура	Docker + Nginx + CI/CD	Забезпечує ізоляцію середовищ, безпечне маршрутизування трафіку (Reverse Proxy) та автоматизацію релізів.

Підсумовуючи, обрана архітектура закладає міцний фундамент для розвитку проєкту. Вона дозволить легко розширювати платформу та додавати нові функції, зберігаючи її швидкою та безпечною. Таке поєднання технологій гарантує створення сучасного та якісного продукту.

1.4. Обґрунтування вибору інструментальних засобів

Для реалізації розроблюваної платформи оренди автомобілів було сформовано сучасний технологічний стек, де вибір кожного інструменту здійснювався на основі ретельного порівняльного аналізу з наявними на ринку альтернативами з урахуванням специфіки бізнес-вимог системи.

Формування архітектури логічно розпочинається з клієнтської частини, основу якої складає екосистема React [23]. Проте базовий React використовує виключно клієнтський рендеринг (CSR), що є критичним недоліком для платформи формату маркетплейсу, де необхідна якісна індексація пошуковими роботами для залучення органічного трафіку. З огляду на це, було обрано фреймворк Next.js, який вирішує цю проблему на архітектурному рівні.

Таблиця 1.4

Обґрунтування вибору технологій клієнтської частини

Технологія / Інструмент	Альтернатива	Переваги обраного рішення
Next.js	React (класичний SPA)	Забезпечує серверний рендеринг (SSR) для публічного каталогу авто (критично для SEO), зберігаючи реактивність SPA для закритих панелей (в нашому випадку <i>Drivefy Workspace</i>).
Ant Design (AntD)	Bootstrap, Tailwind CSS, Material UI	Корпоративна UI-бібліотека, яка містить готові складні компоненти (таблиці, графіки, форми), що пришвидшує розробку панелей керування.

Оцінка альтернативних рішень дозволила обрати інструменти, що оптимально задовольняють вимоги як публічного каталогу, так і корпоративного середовища керування автопарком (див. табл. 1.4).

Для забезпечення безперебійної роботи багатого клієнтського інтерфейсу серверну частину пропонується реалізувати на базі Node.js із використанням фреймворку NestJS. Цей вибір, серед низки потужних альтернатив, зумовлений можливістю реалізувати концепцію ізоморфного коду - використання мови TypeScript на всіх рівнях застосунку.

Таблиця 1.5

Порівняльний аналіз серверних технологій

Технологія	Основні характеристики	Причина відхилення / вибору
PHP (Laravel)	Надійна класика, велика екосистема, MVC патерн.	Вимагає постійного перемикання контексту між JavaScript на фронтенді та PHP на бекенді.
Java (Spring Boot)	Потужне Enterprise-рішення, висока безпека.	Занадто громіздке, має надлишковий рівень абстракцій (boilerplate) для SaaS-продукту середнього масштабу.
Go (Golang)	Найвища продуктивність та швидкість компіляції.	Менш розвинена екосистема готових інструментів для швидкої реалізації бізнес-логіки.

Технологія	Основні характеристики	Причина відхилення / вибору
Node.js (NestJS)	Суворі типізація, модульність, Fullstack TypeScript.	Обрано: дозволяє перевикористовувати типи даних (DTO) між клієнтом та сервером, зменшуючи кількість помилок інтеграції.

Таким чином, на основі проведеного порівняльного аналізу (див. табл. 1.5), як базову технологію для реалізації серверної частини було обрано фреймворк NestJS (на базі Node.js). Попри високу продуктивність рішень на Go чи зрілість екосистем Java та PHP, саме NestJS забезпечує оптимальний баланс між швидкістю розробки бізнес-логіки та надійністю архітектури завдяки суворій типізації мови TypeScript. Використання єдиного технологічного стеку (Fullstack TypeScript) для клієнтської та серверної частин платформи «Drivefy» дозволяє безшовно перевикористовувати об'єкти передачі даних (DTO), мінімізувати ризик виникнення помилок під час інтеграції API та значно спрощує подальше масштабування системи.

Оскільки обрана серверна технологія має працювати зі складними сутностями, постало питання вибору оптимальної системи зберігання даних. Аналізувалися як реляційні (SQL), так і нереляційні (NoSQL) моделі. Враховуючи, що система оренди авто передбачає виконання складних транзакцій та підтримання жорстких зв'язків (наприклад, послідовної залежності між компанією, автомобілем, фактом бронювання та фінансовим звітом), використання SQL з повною підтримкою властивостей ACID виявилось безальтернативним.

Таблиця 1.6

Порівняльний аналіз систем управління базами даних

СУБД	Тип	Особливості та обґрунтування
MongoDB	NoSQL	Гнучка схема, швидка розробка. Відхилено: брак суворі структури та реляційних зв'язків.

СУБД	Тип	Особливості та обґрунтування
MySQL	SQL (Реляційна)	Стандартне реляційне рішення. Відхилено: менш гнучка робота з напівструктурованими даними порівняно з аналогами.
PostgreSQL	SQL (Реляційна)	Обрано: повна підтримка ACID, паралельне виконання транзакцій (MVCC) та нативний тип JSONB для зберігання динамічних характеристик авто.

Для обґрунтування архітектурного вибору системи управління базами даних (СУБД) для платформи оренди автомобілів було проведено порівняльний аналіз реляційних та нереляційних альтернатив (див. табл. 1.6). Головними критеріями оцінювання виступили забезпечення цілісності інформації, підтримка складних реляційних зв'язків та можливість ефективного збереження мінливих специфікацій транспортних засобів. За результатами зіставлення нереляційну СУБД MongoDB було відхилено через брак суворої структурованості, а реляційну MySQL - через обмежену гнучкість під час роботи з напівструктурованими об'єктами. Оптимальним рішенням для проєкту було обрано СУБД PostgreSQL, яка повністю задовольняє вимогам високої надійності транзакцій завдяки підтримці ACID та MVCC, а також дозволяє ефективно зберігати динамічні характеристики автомобілів у нативному форматі JSONB.

Надійна база даних PostgreSQL у поєднанні з ORM Sequelize є фундаментом для інтеграції платіжного шлюзу Stripe, що дозволяє безпечно зберігати платіжні токени (PCI DSS) для безшовної оплати та стягнення додаткових витрат після завершення оренди [24]. Якість продукту забезпечується комбінованим підходом: клієнтський інтерфейс покривається модульними тестами у Vitest, а бізнес-логіка та контракти API перевіряються ручним тестуванням і через Postman [25]. Завершальним етапом є доставка

коду на сервери з використанням сучасного DevOps-підходу для гарантування безперебійних оновлень.

Таблиця 1.7

Порівняння методів розгортання та інфраструктури

Метод розгортання	Характеристика процесу	Недоліки / Переваги
Ручне (FTP / SSH)	Передача файлів вручну, налаштування середовища безпосередньо на сервері.	Відхилено: високий ризик людської помилки, проблема "на моєму комп'ютері працює", тривалий час простою (downtime).
Docker + GitLab CI/CD	Контейнеризація всіх сервісів; автоматичний запуск тестів та збірка образів після злиття коду.	Обрано: ідентичність середовищ виконання, відсутність простоїв під час оновлення, максимальна надійність релізів.

З метою обґрунтування вибору стратегії релізу та архітектури управління інфраструктурою було проведено порівняльний аналіз методів розгортання вебплатформи (див. табл. 1.7). Основними критеріями оцінювання виступили рівень автоматизації процесів, стабільність роботи системи під час оновлень та мінімізація ризиків виникнення помилок. За результатами зіставлення традиційний підхід із ручним налаштуванням сервера через FTP/SSH було відхилено через високу ймовірність людського фактора, тривалий час простою системи (downtime) та проблему неузгодженості конфігурацій середовища розробника й хостингу. Оптимальним технологічним рішенням для проекту було обрано автоматизований конвеєр на базі технології контейнеризації Docker та інструментів GitLab CI/CD. Такий підхід гарантує повну ідентичність середовищ виконання на всіх етапах розробки, забезпечує оновлення платформи з нульовим часом простою та максимізує надійність кожного релізу.

Висновки до першого розділу

У першому розділі було виконано постановку задачі на розробку багатофункціональної платформи для оренди автомобілів. Проведений аналіз ринку та існуючих рішень (каршерінгових сервісів, класичного прокату та

P2P-платформ) дозволив виявити їхні обмеження та чітко визначити нішу розроблюваного продукту - створення гібридного B2B2C-маркетплейсу, що поєднує клієнтську вітрину з повноцінним SaaS-середовищем для управління автопарками.

На основі дослідження сформовано ключові вимоги до системи: підтримка багаторівневої рольової моделі, цифровізація логістичних процесів за допомогою QR-кодів, автоматизація контролю доступності транспортних засобів та безпечна обробка онлайн-платежів. Для задоволення цих потреб обґрунтовано використання клієнт-серверної архітектури на базі REST API, що гарантує ізолюваність процесів, відмовостійкість та здатність системи до масштабування.

Для практичної реалізації обрано сучасний стек технологій. Клієнтську частину спроектовано на базі фреймворку Next.js та корпоративної UI-бібліотеки Ant Design, що забезпечить оптимальний баланс між SEO-індексацією публічних сторінок та реактивністю закритих панелей керування. Серверну логіку буде імплементовано за допомогою модульного Node.js-фреймворку NestJS. Надійною основою для збереження складних реляційних зв'язків та транзакцій виступатиме СУБД PostgreSQL у поєднанні з ORM Sequelize. Для забезпечення фінансового циклу інтегрується платіжний шлюз Stripe, а процеси розгортання та тестування спиратимуться на Docker, GitLab CI/CD, Vitest та Postman. Такий комплексний підхід дозволить усунути ключові недоліки існуючих на ринку монопольних систем, відкриваючи гнучкі можливості для реєстрації сторонніх автопарків. Для надійного функціонування такої відкритої екосистеми закладені інфраструктурні рішення гарантують високий рівень безпеки фінансових даних та створюють передумови для безшовного масштабування платформи під зростаючі навантаження. Загалом, сформовані бізнес-вимоги та обґрунтований комплекс інструментальних засобів створюють міцний інженерний фундамент для переходу до наступного етапу - проектування структури бази даних та безпосередньої програмної реалізації платформи.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ПЛАТФОРМИ ДЛЯ ОРЕНДИ АВТО

2.1. Визначення варіантів використання та акторів

Платформу Прибрав одне найменш суттєве речення - останнє в абзаці про мультитенантність (про динамічний перерахунок доступності, бо ця думка вже частково розкрита раніше). Нижче повний текст:

Платформу оренди автомобілів реалізовано як багаторівневу B2B2C-систему з ієрархічним поділом функціональності залежно від ролі користувача та його належності до конкретної компанії. Її основу становлять модулі пошуку й бронювання транспортних засобів, безпечної обробки платежів та комплексного адміністрування автопарків (Drivefy Workspace). Уся бізнес-логіка спирається на реляційну базу даних, яка забезпечує транзакційну цілісність під час одночасного доступу до спільних ресурсів.

Перш ніж перейти до програмної реалізації, необхідно формалізувати поведінкові вимоги до системи через визначення кола її користувачів та доступних їм сценаріїв. Саме чітке окреслення акторів і притаманних кожному з них повноважень є відправною точкою проєктування, оскільки від коректності цього поділу залежить як архітектура рівня безпеки, так і логіка розмежування доступу до даних. Подвійна природа платформи, що одночасно обслуговує приватних клієнтів і бізнес-орендодавців, зумовлює потребу в гнучкій рольовій моделі, здатній враховувати не лише тип користувача, а й контекст його належності до конкретного автопарку.

Функціональні можливості розподілено між п'ятьма основними типами акторів та одним системним фоновим процесом, де кожен наступний рівень розширює права попереднього. Початковою точкою взаємодії є гість - неавторизований відвідувач, якому доступний виключно публічний функціонал: перегляд каталогу, розширена фільтрація (за класом, трансмісією, ціною) та ознайомлення з умовами компаній-орендодавців і геолокацією пунктів видачі. Після автентифікації відвідувач трансформується у користувача - авторизованого клієнта, який отримує доступ до особистого

кабінету, оформлення бронювань, безпечного збереження платіжних даних та керування відгуками після поїздки.

Наступний рівень представляє співробітник - лінійний працівник компанії, зосереджений на логістиці: скануванні QR-перепусток, фіксації технічного стану авто під час видачі й повернення та обліку супутніх витрат. Його розширені права діють виключно в межах робочого простору, тоді як в інших компаніях він ідентифікується як звичайний користувач. Операційне управління забезпечує адміністратор, наділений правами редагування локацій, створення карток авто, встановлення тарифів, ведення технічного обліку (моніторинг ТО і страхових полісів) та керування складом співробітників. Найвищою ланкою є власник - ініціатор створення робочого простору, який акумулює права всіх нижчих рівнів, а його ключова функція зміщена в бік стратегічного планування на основі аналітичних дашбордів.

Реалізація такої рольової моделі вимагає застосування принципів мультитенантності (multi-tenancy), де кожен бізнес-об'єкт чітко асоційований з ізольованим простором конкретного власника [26]. Це гарантує безпеку комерційних даних кожного автопарку та створює передумови для горизонтального масштабування: кількість компаній-партнерів може зростати без деградації продуктивності центрального ядра API. Кожна операція в межах Drivefy Workspace супроводжується перевіркою контексту (tenant context validation) на рівні middleware, що виключає несанкціонований доступ до ресурсів іншого підприємства. Додатково система забезпечує відмовостійкість завдяки фоновим процесам і планувальникам (Cron jobs), які в асинхронному режимі валідують стан автопарку незалежно від активності користувачів.

Функціональні межі системи, сценарії взаємодії та зв'язки між акторами змодельовано за допомогою діаграми варіантів використання (Use Case Diagram) відповідно до стандартів UML (див. рис. 2.1). Така модель дозволяє простежити ієрархічне успадкування повноважень, залежності «extend» та «include» і контекстозалежність ролей. Вона слугує архітектурним фундаментом для подальшої реалізації, де кожен прецедент відображається на

відповідний API-маршрут, сервісний шар і контролер, забезпечуючи наскрізний контроль безпеки.



Рис. 2.1 - Діаграма варіантів використання платформи для оренди авто

Запропонована модель взаємодії забезпечує однозначний розподіл доступу до функцій залежно від ролі та належності до компанії, гарантуючи ізоляцію даних різних орендодавців у межах єдиної платформи. На основі цієї моделі сформовано набір функціональних і нефункціональних вимог, які лягли в основу розробки.

Функціональні вимоги охоплюють ключові можливості платформи та визначаються переліком сценаріїв, доступних кожному актору:

- автентифікація та авторизація з рольовою моделлю доступу (RBAC) у межах мультитенантної архітектури;
- публічний каталог транспортних засобів із розширеною фільтрацією за класом, трансмісією, ціною та локацією пунктів видачі;
- оформлення бронювання з автоматичним розрахунком вартості та інтеграцією платіжного шлюзу (Stripe) з токенизацією картки клієнта;
- особистий кабінет користувача з історією поїздок та керуванням відгуками;
- робочий простір компанії: видача й повернення авто, сканування QR-перепусток, фіксація технічного стану та супутніх витрат;
- адміністрування автопарку: керування картками авто, тарифами, локаціями видачі та технічним обліком (ТО, страхові поліси);
- аналітичні дашборди власника для моніторингу оборотів, чистого доходу та витрат;
- автоматичні фонові процеси контролю придатності транспортних засобів та блокування їх доступності.

Щоб система працювала стабільно та вирішувала поставлені бізнес-задачі, визначено набір нефункціональних вимог. Платформа повинна витримувати високі навантаження під час пікових сезонів оренди та зберігати коректність обробки запитів за одночасного доступу багатьох користувачів. Використання серверного рендерингу гарантує швидкість завантаження публічних сторінок каталогу до 2 секунд, що є критичним для SEO-індексації. Клієнтська частина має бути повністю адаптивною, оскільки Співробітники здійснюють сканування QR-кодів переважно з мобільних пристроїв безпосередньо на паркувальних майданчиках. Окрему категорію складають вимоги безпеки: платіжні дані обробляються виключно через токенизацію на боці Stripe, а дані різних компаній логічно ізольовані в межах мультитенантного середовища.

Загальна модель налічує понад десяток варіантів використання, проте для розкриття внутрішньої логіки платформи доцільно детально

					ІІЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						35

специфікувати ключові сценарії, які послідовно охоплюють повний життєвий цикл взаємодії - від першого знайомства гостя з каталогом до автоматизованого контролю придатності автопарку.

Таблиця 2.1

Сценарій UC-01: Перегляд і фільтрація каталогу автомобілів

Назва сценарію	Перегляд і фільтрація каталогу автомобілів
Ідентифікатор	UC-01
Головний актор	Гість (Guest)
Мета	Знайти відповідний транспортний засіб серед доступних пропозицій
Передумова	Відсутні; функціонал доступний без автентифікації
Основний потік	1. Гість відкриває публічний каталог і застосовує фільтри пошуку. 2. Система обробляє запит та повертає перелік доступних автомобілів за заданими критеріями. 3. Гість відкриває картку обраного авто для ознайомлення з умовами орендодавця та геолокацією пункту видачі.
Розширення	Якщо за заданими критеріями немає доступних авто, система виводить порожній результат із пропозицією змінити фільтри.
Результат	Гість отримує перелік релевантних транспортних засобів та детальну інформацію про обраний автомобіль.

Сценарій UC-01 виступає першою точкою дотику між потенційним клієнтом та платформою (див. табл. 2.1). Оскільки користувач на цьому етапі лише оцінює пропозиції, можливість анонімного пошуку є критично важливою для залучення аудиторії та зниження порогу входу. Гнучка система фільтрації мінімізує когнітивне навантаження, дозволяючи миттєво відсіяти нерелевантні варіанти за ціною, класом чи локацією. Знайшовши оптимальний автомобіль, гість переходить до етапу оформлення, що вимагає перетворення його на авторизованого користувача із захищеним персональним середовищем. Для цього в системі реалізовано безшовний перехід до екранних форм автентифікації та реєстрації нового облікового запису. Таке рішення дозволяє зберегти параметри поточного вибору клієнта та автоматично перенести їх у новостворену сесію для фіналізації бронювання.

Таблиця 2.2

Сценарій UC-02: Реєстрація та автентифікація користувача

Назва сценарію	Реєстрація та автентифікація користувача
Ідентифікатор	UC-02
Головний актор	Гість (Guest)
Мета	Отримати персональний доступ до функцій авторизованого клієнта
Передумова	Відсутні
Основний потік	1. Гість відкриває форму реєстрації та вводить контактні й облікові дані. 2. Система валідує дані та створює обліковий запис. 3. Для подальших сеансів користувач виконує вхід, вводячи облікові дані. 4. Система перевіряє дані та відкриває доступ до особистого кабінету.
Розширення	Якщо введено некоректні або вже зайняті дані (наприклад, невалідну пошту), система повертає повідомлення про помилку.
Результат	Гість отримує підтверджений обліковий запис та авторизований доступ до сервісу.

Сценарій UC-02 формує стійкий персональний контекст клієнта в системі (див. табл. 2.2). Процес реєстрації не лише створює цифровий профіль для збереження історії бронювань, але й є необхідним кроком для забезпечення безпеки та юридичної відповідальності. Авторизація відкриває доступ до конфіденційних операцій, зокрема до збереження платіжних реквізитів у системі, що є обов'язковою умовою для переходу до безпосереднього оформлення угоди.

Таблиця 2.3

Сценарій UC-03: Бронювання транспортного засобу та токенизація картки

Назва сценарію	Бронювання транспортного засобу та токенизація картки
Ідентифікатор	UC-03
Головний актор	Користувач (Client)
Мета	Оформити бронювання обраного автомобіля та безпечно зберегти платіжні дані клієнта для подальших розрахунків
Передумова	Користувач авторизований; обраний автомобіль має статус «Доступний» на вказані дати

Назва сценарію	Бронювання транспортного засобу та токенизація картки
Основний потік	1. Користувач обирає дати оренди та пункт видачі/повернення. 2. Система розраховує фінальну вартість. 3. Користувач вводить платіжні реквізити через інтегрований віджет Stripe. 4. Система ініціює оплату. 5. Створюється запис бронювання; клієнт отримує цифровий QR-код.
Розширення	Якщо транзакцію відхилено банком, користувач отримує повідомлення про помилку з пропозицією використати іншу картку. Бронювання не створюється.
Результат	Автомобіль зарезервовано на вказані дати; його статус доступності змінюється у глобальному пошуку.

Сценарій UC-03 становить центральне комерційне ядро платформи для клієнтської частини (B2C) (див. табл. 2.3). Інтеграція токенизації через Stripe вирішує одразу два завдання: забезпечує швидку і безпечну передоплату та гарантує орендодавцю можливість автоматичного стягнення коштів у разі майбутніх штрафів чи пошкоджень майна. Цей рівень фінансового захисту є критично важливим аргументом для залучення на платформу бізнес-партнерів, процес приєднання яких регулюється окремим потоком.

Таблиця 2.4

Сценарій UC-06: Реєстрація бізнесу

Назва сценарію	Реєстрація бізнесу
Ідентифікатор	UC-06
Головний актор	Користувач (із переходом у роль Власника)
Мета	Створити ізольоване мультитенантне середовище (Tenant) для управління автопарком
Передумова	Актор має зареєстрований клієнтський обліковий запис
Основний потік	1. Користувач заповнює реєстраційну форму компанії 2. Система валідує вхідні дані та генерує унікальний ідентифікатор компанії (Company ID). 3. Користувач автоматично отримує найвищу роль Owner та повний доступ до адміністративних дашбордів.
Розширення	Якщо реєстраційні дані не пройшли валідацію, система повертає повідомлення про помилку, і робочий простір не створюється.

Назва сценарію	Реєстрація бізнесу
Результат	Створено ізольоване мультитенантне середовище для управління автопарком.

Сценарій UC-06 кардинально змінює парадигму використання системи, розкриваючи її B2B-потенціал (див. табл. 2.4). Перехід користувача в статус Власника ініціює створення ізольованого Tenant-середовища. Це дозволяє незалежним прокатним компаніям вести свою діяльність на базі єдиної SaaS-платформи без ризику витоку даних до конкурентів. Отримавши такий простір, компанія може делегувати операційні задачі своєму персоналу, який безпосередньо взаємодіятиме з клієнтами на локаціях.

Таблиця 2.5

Сценарій UC-08: Ідентифікація замовлення та фізична передача авто

Назва сценарію	Ідентифікація замовлення та фізична передача авто
Ідентифікатор	UC-08
Головний актор	Співробітник (Employee)
Мета	Передати автомобіль клієнту з фіксацією його технічного стану на момент видачі
Передумова	Співробітник авторизований та прив'язаний до поточної локації видачі. Клієнт прибув за автомобілем
Основний потік	1. Співробітник використовує сканер у застосунку для зчитування QR-коду з пристрою клієнта. 2. Система миттєво знаходить відповідне бронювання та виводить інформацію на екран. 3. Співробітник проводить візуальний огляд авто та фіксує поточний стан (пробіг, рівень пального) у системі. 4. Співробітник підтверджує видачу автомобіля.
Розширення	Якщо QR-код не зчитується, співробітник може виконати ручний пошук за номером бронювання або прізвищем клієнта.
Результат	Статус бронювання змінюється на «В оренді» (Rented). Починається відлік часу фактичного використання.

Сценарій UC-08 переводить цифрову угоду у фізичну площину (див. табл. 2.5). Використання QR-коду значно пришвидшує ідентифікацію клієнта, усуваючи потребу в тривалому оформленні паперових документів. Суворі фіксації стартових показників (пробігу та пального) співробітником у

застосунку створює беззаперечну доказову базу, яка слугуватиме еталоном під час зворотного прийому транспортного засобу.

Таблиця 2.6

Сценарій UC-09: Повернення автомобіля та фіксація фінального стану

Назва сценарію	Повернення автомобіля та фіксація фінального стану
Ідентифікатор	UC-09
Головний актор	Співробітник (Employee)
Мета	Прийняти автомобіль від клієнта та зафіксувати його стан після завершення поїздки
Передумова	Бронювання має статус «В оренді»; клієнт повернув авто на локацію
Основний потік	1. Співробітник відкриває відповідне бронювання. 2. Проводить огляд авто та фіксує фінальні показники (пробіг, рівень пального, наявність пошкоджень). 3. Система порівнює стан із зафіксованим під час видачі. 4. Співробітник підтверджує повернення.
Розширення	У разі виявлення пошкоджень або порушень умов співробітник ініціює облік супутніх витрат (штрафи, клінінг).
Результат	Статус бронювання змінюється на «Завершено»; автомобіль повертається до пулу доступних.

Сценарій UC-09 логічно замикає життєвий цикл конкретної оренди (див. табл. 2.6). Автоматизоване порівняння фінальних показників із тими, що були зафіксовані під час видачі, нівелює людський фактор та дозволяє системі миттєво розрахувати суми переплат за перевищення пробігу чи нестачу пального. Проте безперебійність таких операцій безпосередньо залежить від наявності коректно налаштованої бази транспортних засобів, за формування якої відповідає адміністративний персонал.

Таблиця 2.7

Сценарій UC-11: Створення та редагування картки автомобіля

Назва сценарію	Створення та редагування картки автомобіля
Ідентифікатор	UC-11
Головний актор	Адміністратор (Admin)
Мета	Додати новий транспортний засіб до автопарку або оновити його дані

Назва сценарію	Створення та редагування картки автомобіля
Передумова	Адміністратор авторизований у межах робочого простору компанії
Основний потік	1. Адміністратор відкриває розділ автопарку та ініціює створення картки. 2. Вводить характеристики авто (марка, модель, клас, трансмісія, фото, локація). 3. Система валідує дані та зберігає картку. 4. Автомобіль з'являється в каталозі компанії.
Розширення	Якщо обов'язкові поля заповнено некоректно, система повертає повідомлення про помилку валідації.
Результат	Створено або оновлено картку автомобіля; авто доступне для подальшого налаштування й бронювання.

Сценарій UC-11 відповідає за наповнення системи ключовим комерційним активом - автомобілями (див. табл. 2.7). Від того, наскільки детально та точно адміністратор внесе інформацію (включно з якісними медіафайлами та коректним ціноутворенням), залежить конверсія показів авто в реальні бронювання на клієнтській вітрині. Водночас додавання автомобіля вимагає не лише комерційного опису, а й суворого моніторингу його технічного та юридичного статусу.

Таблиця 2.8

Сценарій UC-12: Планування ТО та контроль страхових полісів

Назва сценарію	Планування ТО та контроль страхових полісів
Ідентифікатор	UC-12
Головний актор	Адміністратор (Admin)
Мета	Внести та актуалізувати технічні дані автомобіля (дати ТО, термін дії страховки)
Передумова	У системі зареєстровано щонайменше один транспортний засіб
Основний потік	1. Адміністратор відкриває картку конкретного автомобіля. 2. Переходить у розділ «Технічний стан». 3. Вносить кінцеву дату дії страхового поліса та орієнтовну дату/пробіг наступного планового ТО. 4. Система зберігає дані та оновлює графік доступності автомобіля.

Назва сценарію	Планування ТО та контроль страхових полісів
Розширення	Якщо введено некоректну дату (наприклад, у минулому), система повертає повідомлення про помилку валідації.
Результат	Технічні дані авто актуалізовано. Сформовано базис для роботи автоматичних системних алгоритмів блокування.

Сценарій Сценарій UC-12 виконує роль превентивного механізму безпеки (див. табл. 2.8): задані адміністратором дедлайни ТО та терміни дії полісів дозволяють системі автоматично знімати авто з публікації в критичні дати, унеможливлуючи бронювання транспорту, що не відповідає стандартам безпеки.

Розглянуті сценарії в сукупності описують наскрізний потік цінності платформи та взаємну узгодженість ролей. Їх аналіз дозволив виокремити ключові сутності предметної області - користувачів, компанії, ролі та права доступу, автомобілі, локації видачі, бронювання, платіжні записи, відгуки й записи технічного обліку, - на основі яких сформовано модель даних системи.

2.2. Проєктування бази даних та архітектури даних

Ефективне функціонування будь-якого сучасного вебзастосунку, особливо у сфері послуг оренди та B2B-взаємодії, неможливе без надійної, швидкодіючої та добре структурованої бази даних. Для платформи Drivefy, яка об'єднує клієнтів, власників автопарків та адміністраторів, база даних виступає не просто пасивним сховищем інформації, а ядром усієї бізнес-логіки. Вона повинна забезпечувати безперебійну обробку сотень одночасних запитів, гарантувати консистентність фінансових транзакцій, стійкість до збоїв та підтримувати складні ієрархічні зв'язки між сутностями системи. Для збереження всієї інформації, що обробляється сервісом, було обрано PostgreSQL - одну з найнадійніших і найпотужніших систем керування базами даних з відкритим вихідним кодом. Вона забезпечує високу продуктивність, підтримку транзакцій, складних запитів і реляційних зв'язків, а також ефективну роботу з розширеними типами даних, що є критично важливим під

час роботи з геоданими, фінансовими розрахунками та складними ієрархіями компаній.

PostgreSQL не поступається комерційним аналогам за функціональністю, є стабільною та активно підтримується спільнотою, а також має зручну інтеграцію з ORM Sequelize, що є суттєвою перевагою для розробки на сучасному технологічному стеку. Використання ORM дозволяє працювати з таблицями як з об'єктами та їхніми взаємозв'язками, значно спрощуючи написання запитів. Це забезпечує абстракцію над SQL-запитами, автоматично реалізує зв'язки між моделями, робить роботу з базою даних більш зручною та безпечною, а також дозволяє використовувати механізми міграцій для ведення версій структури БД.

Жорсткі обмеження цілісності (foreign key, not null, unique) гарантують консистентність даних, а продумана система індексів забезпечує ефективну роботу аналітичних запитів. Завдяки нормалізації структури даних та відсутності надлишкової інформації платформа отримує надійну, продуктивну та масштабовану основу для збирання й обробки даних. Проєкт бази даних передбачає використання двадцяти реляційних таблиць, які охоплюють увесь спектр функціональних вимог системи: від обліку користувачів до управління технічним станом автопарку та фінансової аналітики. Така структура дозволяє ефективно працювати з пов'язаними даними, забезпечує високий рівень безпеки та підтримку всього необхідного функціоналу сервісу.

Логічна модель даних спроектована таким чином, щоб забезпечити повну ізоляцію даних у межах різних автопарків (мультитенантність). Це досягається через зовнішні ключі, що зв'язують кожну сутність із відповідним ідентифікатором компанії. Центральне місце в моделі посідають сутності User, Company, Car та Rental, що формують каркас бізнес-логіки платформи. Для реалізації зв'язків багато-до-багатьох, зокрема при формуванні списку обраних автомобілів користувача, використовуються спеціальні pivot-таблиці (наприклад, Favorite), які забезпечують цілісність даних та відсутність дублювання. Кожен запис у системі супроводжується системними мітками

часу, що дозволяє відстежувати історію змін та проводити аудит дій користувачів.

Для цілісного сприйняття спроектованої структури всі двадцять сутностей разом із атрибутами та зв'язками між ними зведено в єдину ER-діаграму бази даних (див. Додаток А). На ній наочно простежуються ключові реляційні зв'язки: агрегація автомобілів і користувачів навколо сутності Company, прив'язка орендних угод до клієнтів та транспортних засобів, а також ієрархія довідникових таблиць, що описують характеристики авто.

Оскільки повна логічна модель налічує двадцять таблиць, наведення детальної специфікації кожної з них у тексті є надлишковим. Тому далі розглянуто структуру лише ключових сутностей, що формують каркас бізнес-логіки платформи Drivefy та найповніше розкривають принципи організації даних, тоді як решта довідкових і допоміжних таблиць відображені на загальній ER-діаграмі (див. Додаток А). Для кожної з обраних сутностей наведено опис її полів, типів даних і функціональної ролі в системі.

Таблиця 2.9

Специфікація сутності User (Користувачі)

Назва поля	Тип даних	PK	FK	Опис
id	Integer	+	-	Унікальний ідентифікатор користувача
email	String	-	-	Унікальна електронна пошта для авторизації
password	String	-	-	Хешований пароль
firstName	String	-	-	Ім'я користувача
lastName	String	-	-	Прізвище користувача
role	Enum	-	-	Системна роль (CLIENT, EMPLOYEE, OWNER, ADMIN)
companyId	Integer	-	+	Прив'язка до компанії (для власників та співробітників)
avatarUrl	String	-	-	Посилання на зображення профілю
stripeCustomerId	String	-	-	Ідентифікатор клієнта в системі Stripe
createdAt / updatedAt	Date	-	-	Системні мітки часу створення та оновлення

Фундаментом системи, з якого розпочинається взаємодія актора з платформою, є сутність User (див. табл. 2.9). Вона зберігає облікові дані, розмежовує рівні доступу та є центральним вузлом, з яким пов'язані історія оренд, платежі, відгуки й належність до компаній. Поле ідентифікатора клієнта Stripe інтегрує користувача у фінансову екосистему платформи.

Таблиця 2.10

Специфікація сутності Company (Компанії-партнери)

Назва поля	Тип даних	PK	FK	Опис
id	Integer	+	-	Унікальний ідентифікатор компанії
name	String	-	-	Назва компанії або бренду
description	Text	-	-	Опис діяльності та умов прокату
payoutCard	String	-	-	Номер картки для отримання виплат
payoutIban	String	-	-	Банківський рахунок у форматі IBAN
isActive	Boolean	-	-	Статус активності компанії на платформі
createdAt / updatedAt	Date	-	-	Системні мітки часу

Оскільки у Drivefy власники автопарків пропонують власні транспортні засоби, виникла потреба в ізолюваному управлінні бізнесом (див. табл. 2.10). Сутність Company виконує роль цифрового представництва партнера на платформі: вона агрегує всі автомобілі орендодавця, його співробітників та зберігає банківські реквізити для виплат прибутку (IBAN, номери карток).

Таблиця 2.11

Специфікація сутності City (Міста)

Назва поля	Тип даних	PK	FK	Опис
id	Integer	+	-	Унікальний ідентифікатор міста
name	String	-	-	Назва населеного пункту
viewsCount	Integer	-	-	Лічильник популярності напрямку
createdAt / updatedAt	Date	-	-	Системні мітки часу

Географічна доступність сервісу реалізована через ієрархію локацій, першою ланкою якої є сутність City (див. табл. 2.11). Вона структурує

географію присутності платформи та містить лічильник переглядів, що дозволяє виводити найпопулярніші напрямки на головній сторінці вебсайту.

Таблиця 2.12

Специфікація сутності PickupPoint (Точки видачі авто)

Назва поля	Тип даних	PK	FK	Опис
id	Integer	+	-	Унікальний ідентифікатор точки
cityId	Integer	-	+	Посилання на місто розташування
address	String	-	-	Фізична адреса локації
latitude	Decimal	-	-	Географічна широта
longitude	Decimal	-	-	Географічна довгота
isActive	Boolean	-	-	Доступність точки для бронювання
createdAt / updatedAt	Date	-	-	Системні мітки часу

Конкретні місця, де клієнт може отримати або повернути автомобіль, описано сутністю PickupPoint (див. табл. 2.12). Вона пов'язана з містами і зберігає точні географічні координати та фізичні адреси, що дозволяє реалізувати відображення точок на інтерактивній мапі в клієнтському застосунку.

Таблиця 2.13

Специфікація сутності Car (Автомобілі)

Назва поля	Тип даних	PK	FK	Опис
id	Integer	+	-	Унікальний ідентифікатор автомобіля
companyId	Integer	-	+	Власник автомобіля (Компанія)
brandId	Integer	-	+	Марка авто
modelId	Integer	-	+	Модель авто
typeId	Integer	-	+	Тип кузова
fuelId	Integer	-	+	Тип пального
transmissionId	Integer	-	+	Тип трансмісії
pickupPointId	Integer	-	+	Поточна локація автомобіля
year	Integer	-	-	Рік випуску
mileage	Integer	-	-	Пробіг автомобіля
averageRating	Decimal	-	-	Середній рейтинг автомобіля

Назва поля	Тип даних	PK	FK	Опис
averageRating	Decimal	-	-	Середній рейтинг автомобіля
seats	Integer	-	-	Кількість пасажирських місць
color	String	-	-	Колір кузова
pricePerDay	Decimal	-	-	Вартість оренди за одну добу
images	Array	-	-	Масив URL-посилань на фотографії
ordersCount	Integer	-	-	Загальна кількість успішних оренд
manualConfirmation	Boolean	-	-	Прапорець необхідності ручного дозволу оренди

Центральним вузлом, що об'єднує компанії та локації, є сутність Car (див. табл. 2.13). Вона представляє транспортний засіб, готовий до оренди, і містить набір атрибутів - від експлуатаційних (пробіг, місця, колір) до економічних (ціна за день) та рейтингових. Поле ручного підтвердження дозволяє преміум-власникам перевіряти клієнтів перед угодою.

Таблиця 2.14

Специфікація сутності Rental (Орендні угоди)

Назва поля	Тип даних	PK	FK	Опис
id	Integer	+	-	Унікальний ідентифікатор угоди
userId	Integer	-	+	Клієнт, який бронює авто
carId	Integer	-	+	Орендований автомобіль
pickupPointId	Integer	-	+	Місце передачі транспортного засобу
startDate	Date	-	-	Дата та час початку оренди
endDate	Date	-	-	Дата та час завершення оренди
totalPrice	Decimal	-	-	Загальна вартість угоди
status	Enum	-	-	Статус (PENDING, ACTIVE, COMPLETED, CANCELLED)
isPaid	Boolean	-	-	Стан розрахунку
qrCode	String	-	-	Токен/URL для підтвердження видачі
confirmationStatus	Enum	-	-	Стан ручного підтвердження менеджером

Бізнес-логіку прокату фіксує сутність Rental (див. табл. 2.14). Вона описує передачу авто клієнту, контролює часові рамки оренди, зберігає QR-код для сканування та відстежує життєвий цикл заявки.

Таблиця 2.15

Специфікація сутності Payment (Платежі)

Назва поля	Тип даних	PK	FK	Опис
id	Integer	+	-	Унікальний ідентифікатор платежу
rentalId	Integer	-	+	Оренда, за яку здійснюється платіж
userId	Integer	-	+	Користувач-платник
pickupPointId	Integer	-	+	Ідентифікатор пункту видачі автомобіля
user_payment_method_id	Integer	-	+	Ідентифікатор збереженого платіжного методу користувача
amount	Decimal	-	-	Сума операції
application_fee_amount	Decimal	-	-	Сума комісії платформи
commission_amount	Decimal	-	-	Сума комісійної винагороди
currency	String	-	-	Валюта платежу (наприклад, UAH)
createdAt	Date	-	-	Дата та час створення запису
description	String	-	-	Детальний опис призначення платежу
externalId	String	-	-	Зовнішній ідентифікатор транзакції
method	Enum	-	-	Метод оплати
orderId	String	-	-	Ідентифікатор пов'язаного замовлення
paid_at	Date	-	-	Фактична дата та час успішної оплати
payment_date	Date	-	-	Розрахункова або запланована дата платежу
paymentDetails	JSON	-	-	Додаткові деталі платежу
status	String	-	-	Поточний статус платежу
stripe_destination_account	String	-	-	Ідентифікатор цільового акаунту Stripe орендодавця

продовж. табл. 2.15

Назва поля	Тип даних	PK	FK	Опис
stripe_charge_id	String	-	-	Ідентифікатор списання (Charge) у системі Stripe
stripe_payment_intent_id	String	-	-	Ідентифікатор наміру платежу (Payment Intent) у Stripe
stripe_transfer_id	String	-	-	Ідентифікатор переказу (Transfer) у системі Stripe
subtotal_amount	Decimal	-	-	Проміжна сума
total_amount	Decimal	-	-	Загальна підсумкова сума платежу
updatedAt	Date	-	-	Дата та час останнього оновлення запису

Жоден сучасний сервіс не може існувати без системи білінгу (див. табл. 2.15). Сутність Payment документує кожну фінансову транзакцію, що проходить через платформу. Вона розроблена з урахуванням можливості використання кількох платіжних провайдерів (Stripe, LiqPay) і гарантує збереження ідентифікаторів зовнішніх транзакцій для аудиту та вирішення спірних питань.

Таблиця 2.16

Специфікація сутності UserPaymentMethod (Збережені методи оплати)

Назва поля	Тип даних	PK	FK	Опис
id	Integer	+	-	Унікальний ідентифікатор
userId	Integer	-	+	Власник картки
stripePaymentMethodId	String	-	-	Токен картки в системі Stripe
last4	String	-	-	Останні 4 цифри картки для відображення
brand	String	-	-	Платіжна система (Visa, MasterCard)
isDefault	Boolean	-	-	Основна картка за замовчуванням

Для покращення користувацького досвіду (UX) впроваджено сутність UserPaymentMethod (див. табл. 2.16). Вона дозволяє клієнтам безпечно зберігати свої платіжні інструменти (у вигляді токенізованих даних Stripe) для майбутніх оренд, усуваючи потребу щоразу вводити номер картки.

Таблиця 2.17

Специфікація сутності Favorite (Обрані авто)

Назва поля	Тип даних	PK	FK	Опис
id	Integer	+	-	Унікальний ідентифікатор запису
userId	Integer	-	+	Користувач, який зберіг авто
carId	Integer	-	+	Збережений автомобіль
createdAt	Date	-	-	Дата додавання до обраного

Аби стимулювати повернення клієнтів на платформу, створено сутність Favorite (див. табл. 2.17). Вона виконує роль списку бажань (wishlist), дозволяючи користувачам зберігати автомобілі, які їм сподобалися, для швидкого доступу в майбутньому. Саме ця сутність є прикладом pivot-таблиці зв'язку «користувач - автомобіль», згаданої вище.

Таблиця 2.18

Специфікація сутності Review (Відгуки)

Назва поля	Тип даних	PK	FK	Опис
id	Integer	+	-	Унікальний ідентифікатор відгуку
userId	Integer	-	+	Автор відгуку
carId	Integer	-	+	Автомобіль, якого стосується відгук
rentalId	Integer	-	+	Угода, в рамках якої залишено відгук
rating	Integer	-	-	Оцінка від 1 до 5
comment	Text	-	-	Текстове враження від оренди

Довіра до платформи базується на прозорості відгуків (див. табл. 2.18). Сутність Review надає клієнтам можливість оцінити транспортний засіб після завершення оренди. Ці дані є ключовими для формування автоматичного рейтингу автомобіля. Наявність зовнішніх ключів userId та carId забезпечує сувору реляційну пов'язаність відгуку з конкретним автором та об'єктом прокату. Крім того, обов'язковий зв'язок із сутністю укладеної угоди (rentalId) повністю блокує можливість генерації фіктивних оцінок, гарантуючи, що відгук може бути залишений виключно за фактом реального користування послугою.

Таблиця 2.19

Специфікація сутності Insurance (Страхування)

Назва поля	Тип даних	PK	FK	Опис
id	Integer	+	-	Унікальний ідентифікатор поліса
carId	Integer	-	+	Застрахований автомобіль
provider	String	-	-	Назва страхової компанії
policyNumber	String	-	-	Номер страхового документа
startDate	Date	-	-	Дата початку дії поліса
endDate	Date	-	-	Дата закінчення дії поліса

Комерційне використання автомобілів вимагає жорсткого контролю ризиків, що вирішується сутністю Insurance (див. табл. 2.19). Вона зберігає деталі страхових полісів, терміни їх дії та покриття, що допомагає компаніям уникати видачі авто з простроченим захистом.

Таблиця 2.20

Специфікація сутності MaintenanceRecord (Історія обслуговування)

Назва поля	Тип даних	PK	FK	Опис
id	Integer	+	-	Унікальний ідентифікатор запису
carId	Integer	-	+	Автомобіль, що пройшов ТО
description	Text	-	-	Опис виконаних робіт
cost	Decimal	-	-	Вартість обслуговування
date	Date	-	-	Дата проведення робіт

Для забезпечення технічної справності автопарку в системі передбачено сутність MaintenanceRecord (див. табл. 2.20). Вона формує цифрову сервісну книжку для кожного автомобіля, фіксуючи витрати на ремонт, планові ТО та заміну запчастин. Завдяки використанню зовнішнього ключа carId кожен запис про технічні роботи суворо прив'язується до конкретної транспортної одиниці, що дозволяє відстежувати повну історію її експлуатації. Накопичена інформація про вартість (cost) та характер робіт є базою для аналітичного модуля платформи, який допомагає партнерам оцінювати рентабельність утримання кожної машини та оптимізувати операційні витрати.

Запроєктована таким чином логічна схема бази даних охоплює всі аспекти бізнес-моделі Drivefy - від обліку користувачів і компаній до фінансової аналітики та контролю технічного стану автопарку.

2.3. Проєктування та реалізація алгоритмів роботи додатку

На етапі проєктування системи управління орендою «Drivefy» критично важливим є формалізований опис бізнес-логіки. Для забезпечення стабільності, безпеки та інтерактивності спроєктовано низку ключових алгоритмів. Варто зауважити, що наведено не вичерпний перелік усіх алгоритмів додатку, проте саме ці механізми є фундаментальними, і їхнього опису достатньо для розуміння загальної архітектури, руху даних та базової логіки функціонування більшої частини продукту. Графічне представлення їхньої роботи продемонстровано за допомогою діаграм активностей.

Систему побудовано за клієнт-сервальною архітектурою на основі фреймворків NestJS (сервер) та Next.js (клієнт). Взаємодія між рівнями здійснюється через RESTful API, де кожна відповідь проходить через глобальний інтерцептор TransformInterceptor, що уніфікує структуру даних незалежно від джерела. Серверна бізнес-логіка зосереджена в модулях (auth, fleet, rentals, favorites тощо), а клієнтська взаємодія реалізована через RTK Query з кешуванням та автоматичною інвалідацією тегів. Такий підхід забезпечує чітке розмежування відповідальностей між рівнями та спрощує їх незалежне масштабування.

Початковою точкою взаємодії користувача із системою є алгоритм автентифікації, що формує базовий рівень безпеки. Його проєктування зосереджено на надійній ідентифікації клієнта та встановленні сесії без збереження стану на сервері (stateless-сесії) - сучасному стандарті масштабованих архітектур. Це оптимізує використання серверних ресурсів, оскільки зникає потреба підтримувати активні сесії в пам'яті чи базі даних: уся контекстна інформація про користувача інкапсулюється безпосередньо в токени, а кожен наступний запит обробляється як самодостатній процес.

Алгоритм починається з прийому облікових даних (електронна пошта та пароль) через захищений POST-ендпоінт. Першим кроком є пошук запису в базі за адресою пошти через UsersService. Якщо запис знайдено, виконується криптографічна перевірка пароля бібліотекою bcrypt, яка порівнює введений пароль із хешованим значенням [27]. Bcrypt реалізує хешування на основі функції Blowfish з параметром вартості 10, що робить підбір методом грубої сили практично неможливим. У разі невідповідності алгоритм повертає HTTP-статус 401 Unauthorized без деталізації причини, запобігаючи витoku інформації про наявність облікового запису.

У разі успішної валідації сервер переходить до генерації JSON Web Token. У payload токена інкапсулюються ключові атрибути: ідентифікатор користувача (sub), пошта, роль (role), ім'я, прізвище, номер телефону та ідентифікатор компанії (companyId). Наявність цих даних у токени дозволяє бекенду миттєво розпізнавати клієнта при кожному запиті, уникаючи звернень до бази. Токен підписується секретним ключем за алгоритмом HS256 та передається у вигляді HTTP-only cookie, що виключає доступ до нього через JavaScript і знижує ризик XSS-атак.

Стратегію вилучення токена (JwtStrategy) реалізовано з подвійним механізмом: спочатку система зчитує токен з HTTP-only cookie, і лише за його відсутності перевіряє заголовок Authorization у форматі Bearer Token. Це дозволяє одночасно обслуговувати веббраузери та зовнішніх API-клієнтів (мобільні додатки, Swagger) [28]. Після верифікації метод validate повертає об'єкт користувача, який NestJS прикріплює до запиту як req.user і робить доступним у контролерах через декоратор CurrentUser(). Така архітектура поєднує захист від CSRF-атак через ізольовані cookies з універсальністю REST API для сторонніх інтеграцій. Відповідна діаграма (див. рис. 2.2) демонструє послідовність процесів: ініціалізацію запиту, пошук у базі, перевірку хешу пароля, генерацію токена та передачу його клієнту.

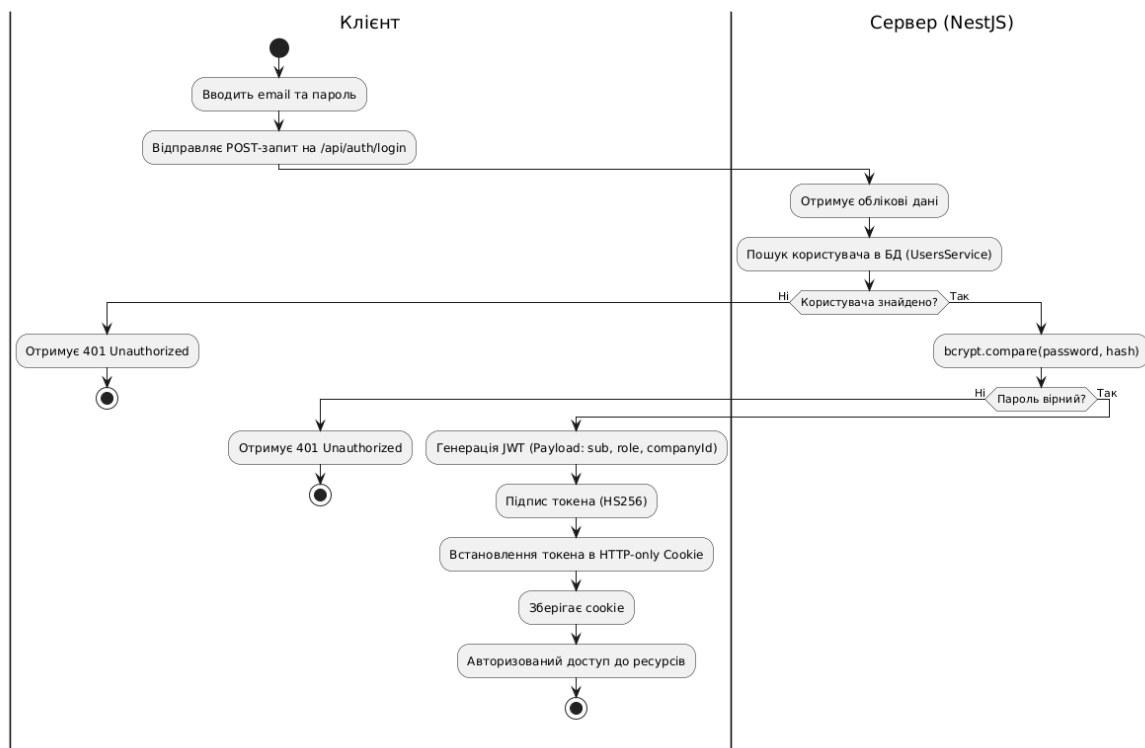


Рис. 2.2 - Діаграма активностей «Алгоритм автентифікації (Login)»

Наведено лістинг коду, що реалізує основну логіку автентифікаційного сервісу та стратегію вилучення токена для встановлення захищеної сесії (див. Додаток В). Метод `validateUser` перевіряє наявність користувача та валідує пароль, обов'язково вилучаючи його хеш перед поверненням результату. Далі метод `login` формує JWT-токен із стандартизованим payload (зокрема полем `sub`). За вилучення та перевірку токена відповідає `JwtStrategy`: він пріоритетно шукає його в HTTP-only cookie, а за відсутності - у заголовку `Authorization`, після чого перевіряє термін дії та зберігає верифіковані дані в `req.user`.

Після успішного входу та встановлення сесії в дію вступає алгоритм рольового розмежування доступу (RBAC). Система розрізняє п'ять типів користувачів (`guest`, `client`, `employee`, `admin`, `owner`), гарантуючи строгу контекстну безпеку на рівні кожного маршруту API. Ключовою особливістю цього RBAC є декларативний підхід: декоратор `Roles()` через функцію `SetMetadata` прикріплює масив дозволених ролей безпосередньо до обробників маршрутів або контролерів, що спрощує читабельність без створення централізованих конфігів. Безпосередню перевірку прав виконує клас `RolesGuard` (реалізація `CanActivate`): він перехоплює запит, через сервіс

Reflector зчитує метадані та порівнює їх із роллю користувача з об'єкта req.user. Якщо роль не відповідає дозволеним, NestJS автоматично генерує виняток ForbiddenException зі статусом HTTP 403. Головною перевагою такої реалізації є незалежність від бази даних під час перевірки прав, оскільки вся ідентифікаційна інформація вже закладена у JWT-токені. RolesGuard застосовується глобально і працює у строгій послідовності після JwtAuthGuard, який першим верифікує токен, що унеможливлює обхід авторизації. Покрокову логіку роботи алгоритму проілюстровано на діаграмі активностей (див. Додаток С). Нижче наведено лістинг коду, що реалізує алгоритм рольового розмежування доступу:

```
export class RolesGuard implements CanActivate {
  constructor(private reflector: Reflector) {}
  canActivate(context: ExecutionContext): boolean {
    const requiredRoles = this.reflector.getAllAndOverride<UserRole[]>(ROLES_KEY,
      [context.getHandler(), context.getClass()]);
    if (!requiredRoles || requiredRoles.length === 0) return true;
    const { user } = context.switchToHttp().getRequest();
    if (!user || !requiredRoles.includes(user.role)) {
      throw new ForbiddenException('Недостатньо прав для виконання цієї дії');
    }
    return true;
  }
}
```

Представлений механізм RBAC реалізований через декоратор Roles (використовує SetMetadata) та сервіс Reflector, який зчитує метадані для перевірки прав. Завдяки синхронній перевірці невеликого масиву ролей (за час $O(n)$) та відсутності звернень до бази даних цей stateless-механізм забезпечує нульові накладні витрати на авторизацію запитів.

Центральним алгоритмом пошуку є механізм багатовимірної динамічної фільтрації у методі findAll сервісу CarsService. Він обробляє до тринадцяти параметрів, формуючи оптимізований SQL-запит лише з явно вказаними критеріями для ефективного використання індексів PostgreSQL. Алгоритм автоматично нормалізує ID-фільтри через функцію getFilterIds, перетворюючи

одиничні чи множинні параметри на умови «=» або IN без дублікатів, та обробляє числові діапазони для ціни й об'єму двигуна.

Найскладнішою частиною пошуку є перевірка часової доступності автомобіля. За наявності дат оренди формується SQL-підзапит із оператором UNION, який через умову NOT IN виключає автомобілі в активній оренді або на технічному обслуговуванні. Завершується формування запиту валідацією терміну дії страховки (через Op.or бібліотеки Sequelize), підключенням асоційованих таблиць (JOIN), сортуванням та пагінацією, що робить метод універсальним інструментом для запитів будь-якої складності. Послідовність етапів проілюстровано на діаграмі активностей (див. Додаток D), а фрагмент коду наведено в лістингу (див. Додаток E). Безпеку запиту забезпечує застосування іменованих параметрів замість конкатенації рядків, що захищає від SQL-ін'єкцій.

Отримавши список доступних автомобілів, користувач має змогу застосовувати складні багатокритеріальні фільтри. За синхронізацію їхнього стану між React-компонентом та URL-адресою браузера відповідає алгоритм, реалізований у вигляді хука useCarFilters. Збереження стану фільтрів в URL дозволяє надсилати посилання з уже застосованими фільтрами, забезпечує коректну роботу кнопок «назад» і «вперед», а також уможливорює серверний рендеринг у Next.js із врахуванням фільтрів - причому стан зберігається після оновлення сторінки без додаткового сховища даних.

Алгоритм ініціалізується під час першого монтування хука через useState, де виконується парсинг URL-параметрів за допомогою useSearchParams() з Next.js App Router. Для полів-масивів реалізовано функцію getArrayParam, яка обробляє одночасно два формати, а параметр lockedFilters дозволяє фіксувати місто або пункт видачі при перегляді авто конкретного офісу.

Двостороння синхронізація реалізована через хук useEffect, що запускається при кожній зміні об'єкта filters і визначає, чи змінилось значення поля порівняно з URL-параметрами. Для порівняння масивів застосовано

канонічне сортування через `JSON.stringify`, а метод `router.replace` (а не `push`) оновлює URL без додавання запису в стек навігації. Для перемикавання масивових фільтрів реалізовано функцію `toggleArrayFilter` за принципом ідемпотентного перемикача, а прапор `{ scroll: false }` запобігає прокрутці сторінки при кожній зміні фільтра.

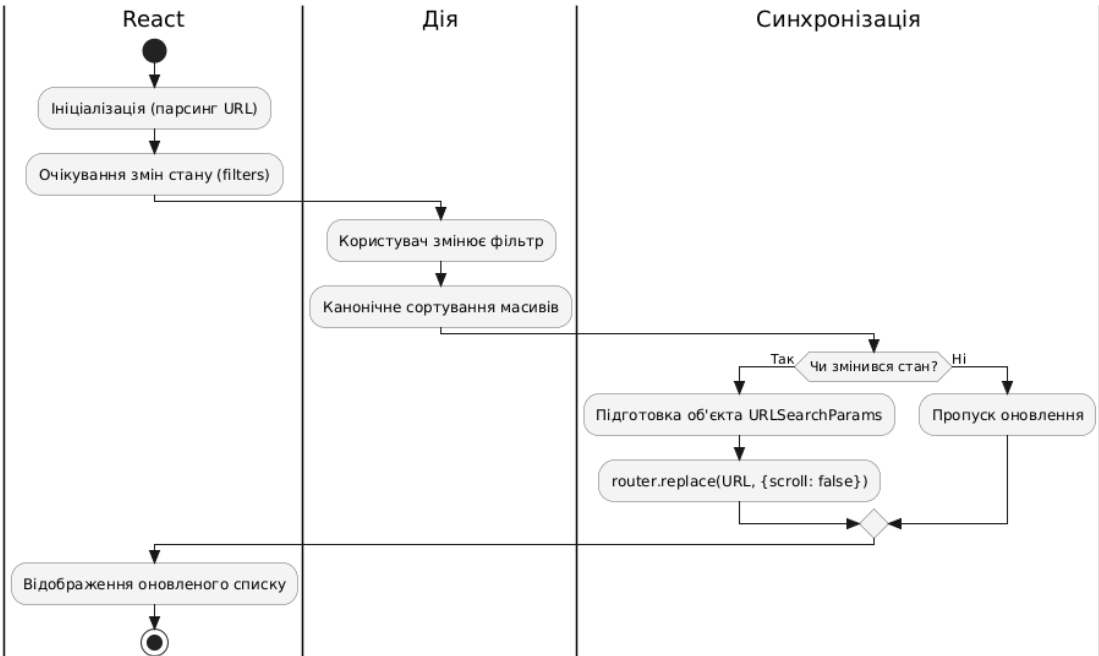


Рис. 2.3 - Діаграма активностей «Алгоритм синхронізації стану фільтрів з URL»

Нижче наведено ключовий лістинг алгоритму двосторонньої синхронізації стану фільтрів з URL:

```

useEffect(() => {
  const params = new URLSearchParams(searchParams.toString());
  let hasChanges = false;
  Object.entries(filters).forEach(([key, value]) => {
    if (lockedFilters?.includes(key as keyof FilterState)) return;
    if (Array.isArray(value)) {
      const currentVals = params.getAll(key);
      const nextVals    = value.map(String);
      if (JSON.stringify([...currentVals].sort()) !==
          JSON.stringify([...nextVals].sort())) {
        params.delete(key);
        nextVals.forEach(v => params.append(key, v));
      }
    }
  });
  if (hasChanges) {
    router.replace(`${location.pathname}?${params.toString()}`, {scroll: false});
  }
});

```

```

        hasChanges = true;
    }
} else if (value !== undefined && value !== null && value !== '') {
    if (params.get(key) !== value.toString()) {
        params.set(key, value.toString()); hasChanges = true;
    }
} else if (params.has(key)) {
    params.delete(key); hasChanges = true;
}
});

```

Представлений фрагмент коду реалізує двосторонній алгоритм синхронізації стану між React та браузерним URL. Функція `getArrayParam` демонструє толерантний парсинг: метод `flatMap` розплющує масив значень у єдиний список, а ланцюжок `map/filter` гарантує числовий тип і відфільтровує некоректні значення. Всередині `useEffect` клонується поточний `URLSearchParams` замість прямої мутації, а прапорець `hasChanges` дозволяє виконати `router.replace` лише при реальних змінах, запобігаючи нескінченному циклу рендерів. Перерахунок відбувається реактивно при кожній зміні `filters` завдяки масиву залежностей хука.

Для забезпечення зручного доступу до улюблених автомобілів у системі реалізовано дворівневий алгоритм управління списком обраного. Перший рівень - клієнтський `Redux-slice` з персистенцією в `localStorage` - забезпечує миттєву відповідь інтерфейсу та роботу функції навіть без авторизації (гостьовий режим). Другий рівень - серверна синхронізація через API - гарантує збереження та узгодженість обраного між різними пристроями та сесіями авторизованого користувача. Взаємодія між рівнями реалізована через алгоритм об'єднуючої синхронізації (`union merge`), що виключає втрату даних незалежно від порядку операцій та кількості пристроїв.

На клієнтській стороні алгоритм керується через `Redux Toolkit` із застосуванням механізмів персистенції, що дозволяє автоматично відновлювати збережений стан із `localStorage` під час перезавантаження сторінки. Базові операції додавання чи видалення авто (`toggleFavorite`)

використовують пряму мутацію масиву через Immer-обгортку, забезпечуючи миттєву реакцію інтерфейсу, а сувора перевірка типів перед кожною операцією гарантує коректну роботу компонентів навіть за некоректного стану кешу.

Серверна частина алгоритму реалізована у вигляді методу syncFavorites у FavoritesService, який викликається автоматично після успішного входу й приймає локальний список carId з клієнтського Redux-стану. Спершу виконується валідація переданих ідентифікаторів через запит до бази, що відфільтровує невалідні або видалені авто. Далі зчитуються поточні записи обраного для userId з використанням Set для O(1)-пошуку, після чого нові carId створюються через bulkCreate - масовий INSERT за один SQL-запит замість N окремих. Фінальним кроком формується об'єднаний список з обох джерел через конструктор Set, який замінює локальний стан на повний серверний. Повну послідовність кроків проілюстровано на діаграмі активностей (див. рис. 2.4).

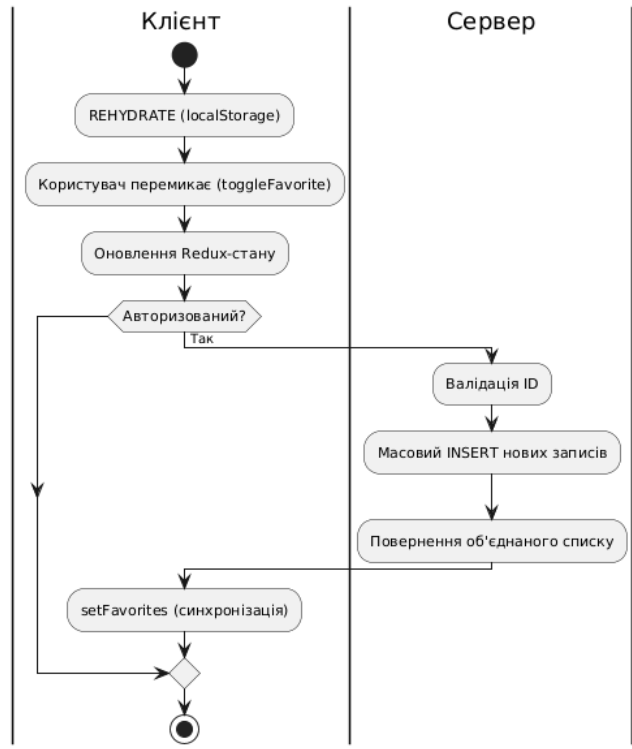


Рис. 2.4 - Діаграма активностей «Алгоритм синхронізації списку обраних автомобілів»

Практичну реалізацію описаного алгоритму на стороні наведено у наступному фрагменті коду:

```
const favoritesSlice = createSlice({
  ...
  reducers: {
    toggleFavorite: (state, action: PayloadAction<string>) => {
      const carId = action.payload;
      if (!Array.isArray(state.items)) { state.items = [carId]; return; }
      const index = state.items.indexOf(carId);
      if (index !== -1) state.items.splice(index, 1);
      else state.items.push(carId);
    },
    setFavorites: (state, action: PayloadAction<string[]>) => {
      state.items = Array.isArray(action.payload) ? action.payload : [];
    },
  },
});
```

Зв'язуючою ланкою між локальним клієнтським станом та серверною бізнес-логікою виступає механізм відкладеної мережевої синхронізації, який автоматично ініціюється в момент зміни статусу авторизації користувача. Поки клієнт перебуває в статусі гостя, усі додані автомобілі зберігаються виключно в пам'яті браузера. Однак, щойно відбувається успішний вхід у систему (реєстрація або авторизація), архітектура застосунку фіксує появу валідного сесійного токена та миттєво ініціює асинхронний HTTP-запит до REST API. Цей запит акумулює весь накопичений локально масив ідентифікаторів обраних автомобілів, упаковує його в оптимізований JSON-формат та відправляє на захищений серверний ендпоінт.

Процес прийому цих даних на бекенді супроводжується суворим контролем цілісності та безпеки на рівні контролерів NestJS. Перш ніж переданий масив ідентифікаторів досягне ядра алгоритму для безпосереднього об'єднання, він проходить через глобальні механізми типізації та валідації (ValidationPipe). На цьому етапі система порівнює вхідне корисне навантаження (payload) із заздалегідь визначеним об'єктом передачі даних (DTO), відсіюючи будь-які некоректні типи, дублікати або спроби шкідливих

ін'єкцій. Лише після підтвердження абсолютної валідності структури даних запит делегується безпосередньо до сервісного шару платформи. Програмну реалізацію цього серверного функціоналу наведено у наступному фрагменті коду:

```
async syncFavorites(userId: number, localCarIds: string[]): Promise<string[]> {
  let validCarIds: string[] = [];
  if (localCarIds?.length > 0) {
    const validCars = await this.carModel.findAll({
      where: { id: localCarIds }, attributes: ['id'],
    });
    validCarIds = validCars.map(car => car.id);
  }
  const existing = await this.favoriteModel.findAll({
    where: { userId }, attributes: ['carId'],
  });
  const existingIds = new Set(existing.map(f => f.carId));
  const newFavorites = validCarIds
    .filter(carId => !existingIds.has(carId))
    .map(carId => ({ userId, carId }));
  if (newFavorites.length > 0)
    await this.favoriteModel.bulkCreate(newFavorites);
  return Array.from(new Set([...existingIds, ...validCarIds]));
}
```

Представлений код демонструє двошарову архітектуру алгоритму персистенції обраного. У клієнтській частині обробник REHYDRATE виконується раніше будь-яких UI-рендерів завдяки позиції в extraReducers, що гарантує заповнення стану перед першим малюванням компонентів. Пряма мутація через splice в Immer-context є ефективнішою за фільтрацію через filter(), оскільки уникає створення нового масиву. У серверній частині ключова оптимізація - використання Set для existingIds замість звичайного масиву: операція has() виконується за O(1) замість O(n), що суттєво прискорює алгоритм при великих списках обраного. Метод bulkCreate виконує всі INSERT в рамках однієї транзакції, гарантуючи атомарність операції: або всі нові записи зберігаються успішно, або жоден. Використання Array.from(new

Set(...existingIds, ...validCarIds) для фінального списку є ідіоматичним JavaScript-способом обчислення об'єднання двох множин без дублікатів, що відповідає математичній моделі union merge та забезпечує ідемпотентність операції синхронізації незалежно від кількості повторних викликів.

Висновки до другого розділу

У другому розділі сформовано архітектурну основу багаторівневої B2B2C-платформи оренди автомобілів Drivefy з мультитенантною організацією та логічною ізоляцією даних. На основі моделі варіантів використання окреслено функціональні межі системи для п'яти типів акторів (Гість, Користувач, Співробітник, Адміністратор, Власник) та фонового процесу контролю автопарку. Деталізовано й формалізовано дев'ять ключових сценаріїв - від анонімного пошуку й бронювання з токенизацією картки через Stripe до видачі авто й аналітики. Визначено функціональні й нефункціональні вимоги, включаючи рольовий доступ (RBAC) та серверний рендеринг публічних сторінок.

Обґрунтовано вибір реляційної СУБД PostgreSQL та ORM Sequelize, на базі яких спроектовано логічну схему з двадцятьма специфікованими таблицями. Центральними вузлами структури виступають сутності User, Company, Car та Rental, а мультитенантність і зв'язки багато-до-багатьох реалізовано через систему зовнішніх ключів та pivot-таблиці.

У межах клієнт-серверної архітектури (NestJS та Next.js) формалізовано п'ять базових алгоритмів: stateless-автентифікацію на JWT із хешуванням bcrypt; рольове розмежування доступу через RolesGuard; багатовимірну динамічну фільтрацію авто за доступністю через SQL-підзапити; двосторонню синхронізацію фільтрів з URL; а також дворівневе керування списком обраного з об'єднанням Redux-стану та сервера. Розроблені рішення формують цілісне технічне підґрунтя для подальшої реалізації масштабованої та безпечної архітектури платформи.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ПЛАТФОРМОЮ ОРЕНДИ АВТО

3.1. Послідовність розгортання та оптимізації параметрів платформи

Процес розгортання та конфігурування розроблюваної платформи оренди автомобілів на віддаленому сервері є критичним етапом, який гарантує її доступність, безпеку та здатність витримувати реальні навантаження. З огляду на модульну архітектуру системи, яка включає клієнтську частину на Next.js, серверну частину на NestJS та реляційну базу даних PostgreSQL, стратегія деплою базується на принципах повної ізоляції середовищ. Для реалізації цього підходу застосовано технологію контейнеризації Docker. Кожен компонент платформи упаковується у власний ізольований контейнер, що повністю вирішує проблему сумісності програмного забезпечення та операційної системи сервера. Для оптимізації процесу створення образів застосовано багатоетапне збирання, під час якого на першому кроці компілюється вихідний код та завантажуються всі інструменти розробки, а на фінальному етапі до робочого контейнера переносяться лише скомпільовані файли та мінімальний набір продуктових залежностей. Управління життєвим циклом та мережевою взаємодією цих контейнерів здійснюється за допомогою інструменту оркестрації Docker Compose, який об'єднує фронтенд, бекенд та базу даних у захищену віртуальну мережу, обмежуючи прямий доступ ззовні до критичних вузлів. Окрім цього, для надійного збереження інформації навіть після перезапуску середовища налаштовано постійні сховища Docker Volumes, що гарантує цілісність реляційної бази даних. Така конфігурація дозволяє легко переносити розгорнуту інфраструктуру між різними хостингами без ризику виникнення конфліктів залежностей чи втрати поточних налаштувань. У підсумку, автоматизоване керування контейнерами закладає міцний фундамент для стабільної роботи системи під час високої активності користувачів та значно спрощує її подальше адміністрування (див. рис. 3.1).

Схема взаємодії ізольованих Docker-контейнерів платформи

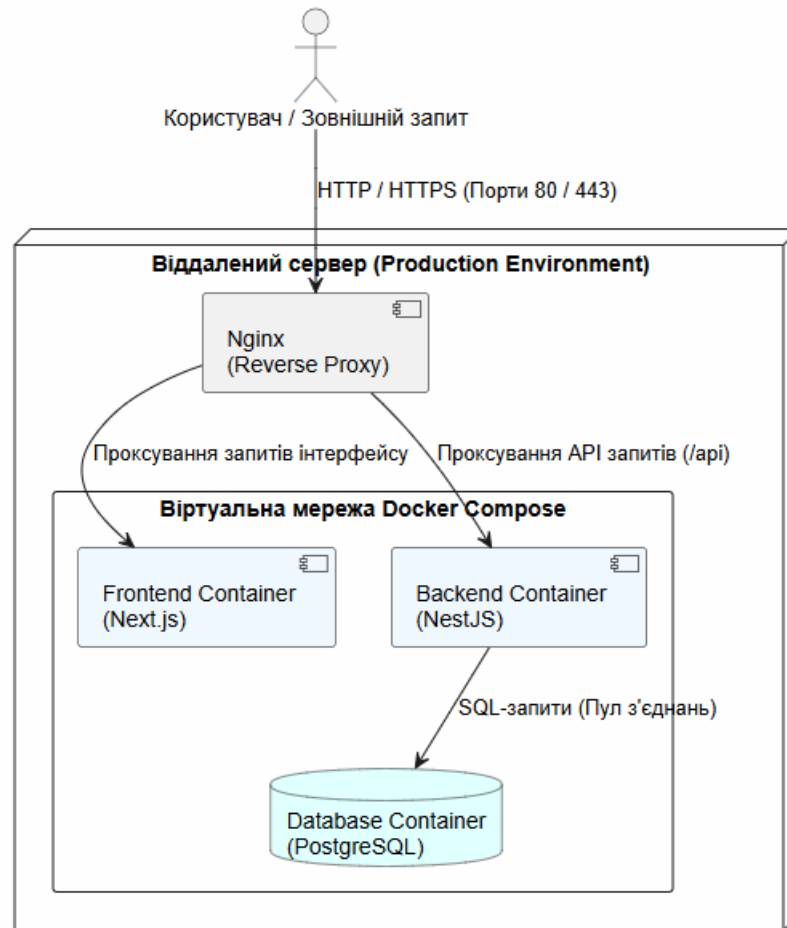


Рис. 3.1 - Схема взаємодії ізольованих Docker-контейнерів платформи

Єдиною точкою входу для всього зовнішнього трафіку виступає вебсервер Nginx, який налаштований у режимі зворотного проксі-сервера. Його головне завдання полягає у прийомі клієнтських запитів та їх інтелектуальному перенаправленні до відповідних Docker-контейнерів залежно від маршруту. Окрім балансування навантаження, Nginx виконує функцію термінації SSL-сертифікатів, що є обов'язковою умовою для безпечної передачі конфіденційних даних та роботи платіжних шлюзів. За допомогою інструментарію від центру сертифікації Let's Encrypt для доменного імені платформи згенеровано криптографічні сертифікати, що дозволило перевести всю взаємодію на захищений протокол HTTPS. Для додаткового посилення захисту від поширених вебвразливостей у конфігурації Nginx активовано строгі політики безпеки, а також налаштовано

обов'язкове перенаправлення всього незахищеного HTTP-трафіку на безпечний порт (див. рис. 3.2).

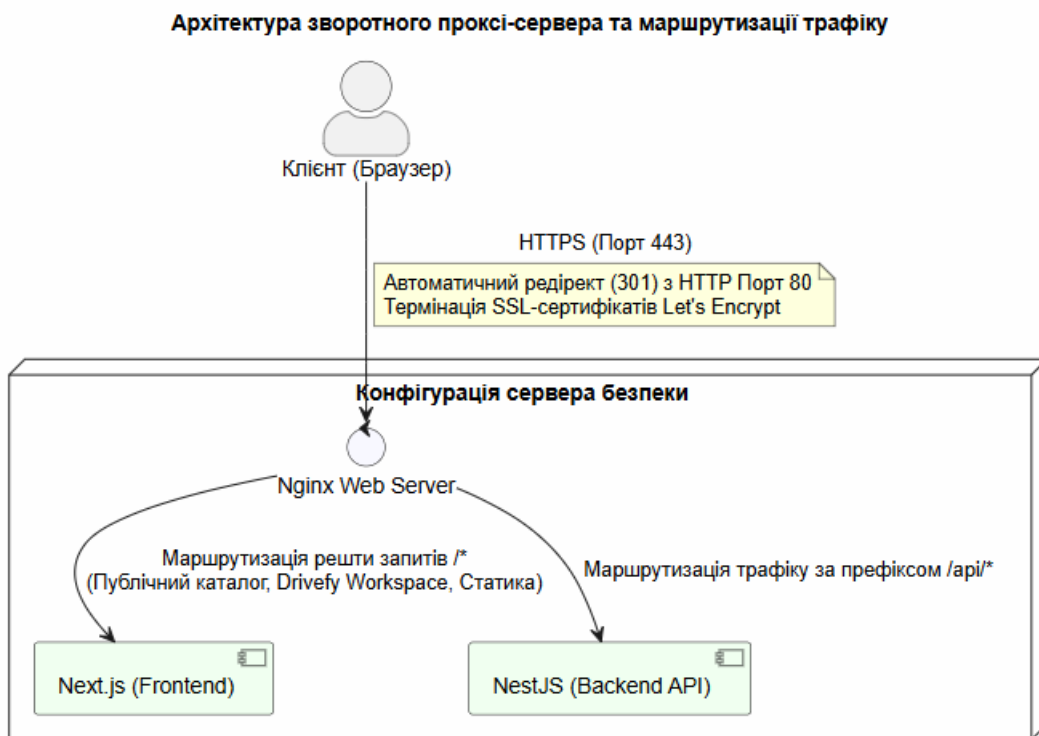


Рис. 3.2 - Архітектура зворотного проксі-сервера та маршрутизації трафіку

Для забезпечення безперервного процесу оновлення програмного продукту без зупинки його функціонування було спроектовано та впроваджено конвеєр безперервної інтеграції та розгортання (CI/CD). Цей автоматизований механізм ініціюється при кожному оновленні кодової бази у репозиторію. Спочатку код проходить етап статичного аналізу для виявлення синтаксичних відхилень, після чого запускається критично важливий етап автоматизованого тестування. Цей етап охоплює як модульне тестування бізнес-логіки серверної частини, так і UI-тестування клієнтського інтерфейсу, яке виконується у безголовому режимі браузера для повної імітації користувацьких сценаріїв, таких як процес бронювання авто або робота з панеллю управління автопарком [29]. У разі успішного проходження всіх тестів система автоматично збирає нові Docker-образи, завантажує їх на віддалений сервер та ініціює фоновий перезапуск контейнерів, забезпечуючи оновлення платформи з нульовим часом простою (див. рис. 3.3).

Етапи конвеєра безперервної інтеграції та розгортання (CI/CD)

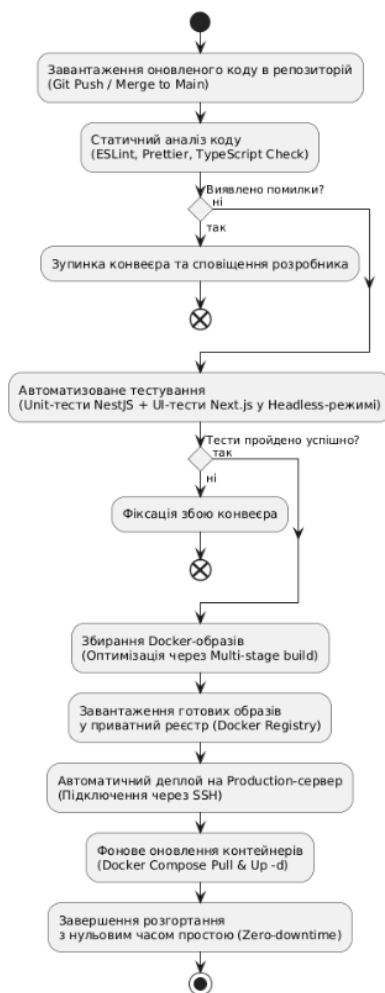


Рис. 3.3 - Етапи конвеєра безперервної інтеграції та розгортання

Після налаштування інфраструктури розгортання здійснюється глибока оптимізація параметрів самої вебсистеми для досягнення максимальної продуктивності. На рівні клієнтського додатка Next.js застосовуються механізми серверного рендерингу та інкрементної статичної регенерації, що дозволяє миттєво віддавати користувачам готові сторінки каталогу автомобілів та знижує навантаження на обчислювальні потужності. Для прискорення передачі даних вебсервер стискає текстові ресурси алгоритмами компресії, а також кешує статичні медіафайли на тривалий термін. На серверному рівні додаток NestJs переводиться у продуктивний режим, що вимикає ресурсомісткі процеси збору налагоджувальної інформації та покращує використання оперативної пам'яті. Паралельно оптимізується взаємодія з базою даних шляхом створення індексів для полів, за якими

відбувається найчастіша фільтрація транспортних засобів, та налаштування пулу з'єднань, що мінімізує затримки при обробці великої кількості одночасних транзакцій. Такий комплексний підхід до розгортання та конфігурації забезпечує створення стабільного, захищеного та високопродуктивного середовища для функціонування платформи оренди авто.

3.2 Організація інтерфейсу та порядок роботи з платформою

Головна сторінка розробленої платформи є ключовою точкою входу та виконує роль презентаційного й навігаційного хабу для двох основних категорій користувачів: кінцевих клієнтів (орендарів) та бізнес-партнерів (власників автопарків). Інтерфейс сторінки містить верхню панель навігації, яка дозволяє неавтентифікованому користувачу миттєво перейти до модулів реєстрації або входу в систему. Окрім цього, на головній сторінці розміщено інтерактивний блок із закликом до співпраці («Стати партнером»), інформаційний розділ «Про нас», де висвітлено місію та переваги компанії, а також прев'ю-секцію з актуальними та доступними для оренди автомобілями (див. рис. 3.4).

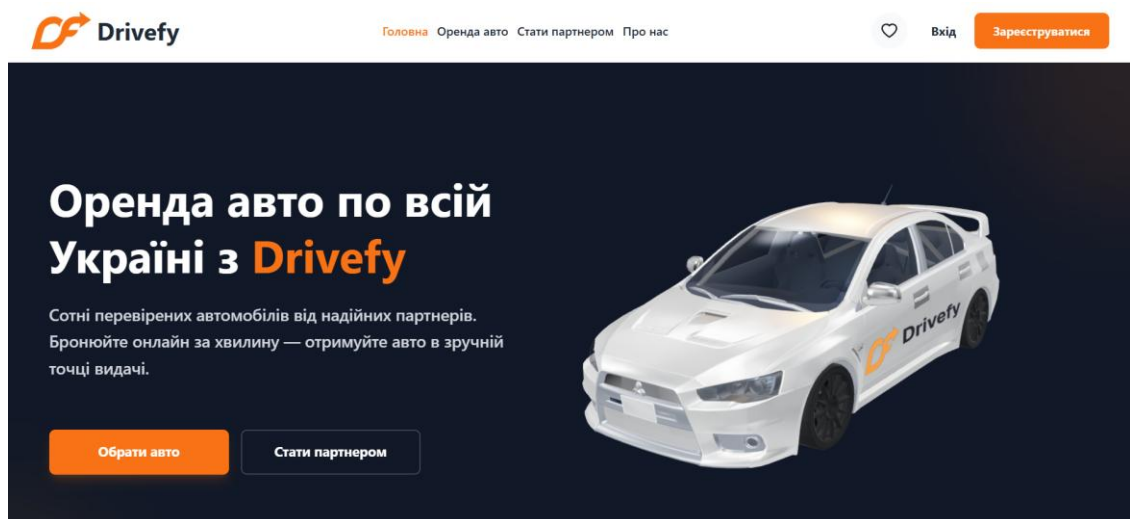


Рис. 3.4 - Головна сторінка платформи оренди авто

Для детального ознайомлення з наявним модельним рядом у системі передбачено спеціалізовану сторінку каталогу автомобілів. Інтерфейс цієї сторінки спроектовано за принципом категорійного розподілу, що дозволяє

Головна
Оренда авто
Стати партнером
Про нас

Фільтри

Сортуння

Найпопулярніші (перегляди)

Дати оренди

ДАТА БРОНЮВАННЯ

від до

Оберіть дату

Оберіть дату

Марка

- ☐ Audi
- ☒ Mercedes-Benz
- ☐ Ford
- ☐ Honda
- ☐ Kia
- ☐ Nissan
- ☐ Opel

Модель

Знайдено авто: 2

Toyota Camry 2023 ☆ 1.0

Житловий... Безкош. Парк. З.Б.А.

\$100 / в день Орендувати

BMW 2 Series 2023

Житловий... Безкош. Телпункт. З.Б.А.

\$80 / в день Орендувати

Для масштабування бізнес-моделі платформи передбачено модуль залучення нових контрагентів. Фізичні або юридичні особи, які бажають здавати в оренду власний автопарк, мають можливість перейти на сторінку реєстрації партнера. Процес передбачає заповнення розширеної форми з внесенням верифікаційних даних, інформації про компанію та контактних реквізитів, після чого створюється обліковий запис із підвищеними правами доступу (див. рис. 3.6).

Drivefy

ГоловнаОренда автоСтати партнеромПро нас

ВхідЗареєструватися

Регістрація партнера

Створіть акаунт власника та зареєструйте свою компанію

Особисті дані власника

Ім'я

Ivan

Прізвище

Sereda

Email

ivansereda@gmail.com

Номер телефону

+380555445444

Дані компанії

Назва компанії

LuxAUTOKyiv

Email компанії

ivansereda@gmail.com

Телефон компанії

+380555445444

Юридична адреса

Kyivska St, 23

					ИЗ.КР.Б – 121 – 26 – ИЗ	Арк.
						68

Після автентифікації користувачу стає доступним особистий кабінет, що дозволяє виконувати CRUD-операції над особистими даними: редагувати контактну інформацію, змінювати пароль чи оновлювати документи. З профілю також реалізовано переходи до журналу бронювань та модуля керування фінансами (див. рис. 3.7).

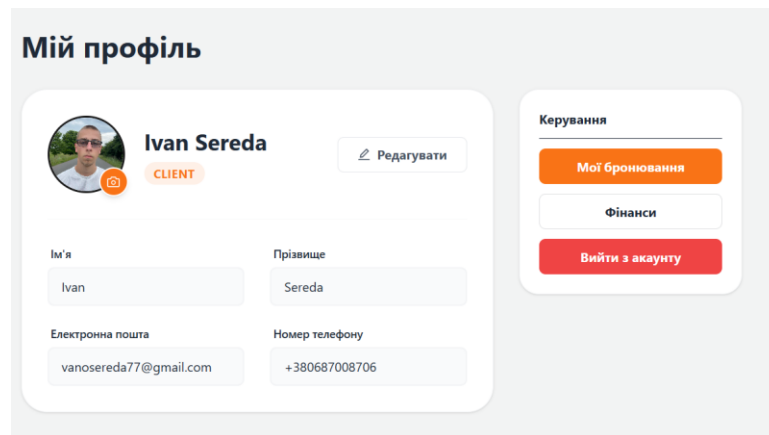


Рис. 3.7 - Сторінка особистого профілю користувача

Для користувачів зі статусом «Партнер» розроблено масштабну підсистему - робочий простір (Workspace), що є централізованою адміністративною панеллю для управління бізнес-процесами. Вона містить вкладки огляду, динаміки, локацій, автомобілів, оренд, видачі/повернення, штрафів, аналітики, налаштувань, бронювань і фінансів. На стартовій сторінці «Огляд» партнер отримує зведену інформацію про стан бізнесу в реальному часі: графіки фінансової динаміки, баланс доходів і витрат, стрічку останніх відгуків та показник задоволеності клієнтів у відсотках (див. рис. 3.8).

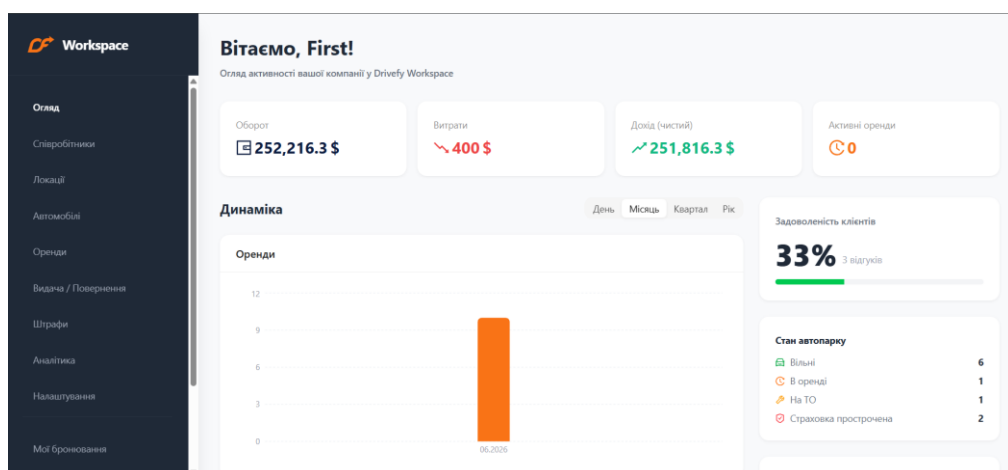


Рис. 3.8 - Сторінка огляду на панелі інструментів Workspace

Невід'ємною частиною Workspace є HR-модуль, призначений для керування співробітниками партнерської мережі (наприклад, менеджерами на локаціях або операційними працівниками). Інтерфейс дозволяє власнику автопарку переглядати повний список персоналу, додавати нових працівників, а також здійснювати швидкий пошук та фільтрацію колег за їхньою системною роллю, іменем або адресою електронної пошти (див. рис. 3.9).

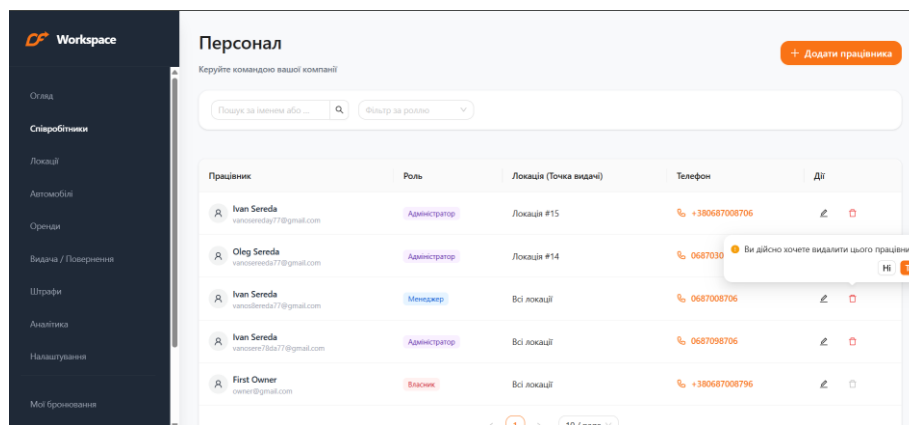


Рис. 3.9 - Сторінка для керування співробітниками

Для оперативного коригування кадрових даних розроблено модальне діалогове вікно (попап) редагування інформації. За його допомогою адміністратор може в декілька кліків змінити рівень доступу та роль працівника в системі, перепризначити його на іншу фізичну локацію (автосалон), оновити контактний номер телефону або стандартні анкетні дані (ім'я та прізвище) (див. рис. 3.10).

Додати нового працівника

✕

Ім'я

Іван

Прізвище

Іванов

Email

example@mail.com

Телефон

+380...

Роль у системі

Оберіть роль

Локація (Точка видачі)

Оберіть точку видачі

Пароль

.....

Скасувати

Додати працівника

Рис. 3.10 - Попап для додавання нового співробітника

Управління географічним покриттям сервісу здійснюється на сторінці локацій, де партнер може створювати нові пункти видачі автомобілів своєї компанії, вказуючи точну адресу (див. рис. 3.11).

Додати нову точку

Швидкий пошук адреси

Пошук адреси...

Локація на карті (клікніть, щоб обрати)

Назва локації

Боярка точка видачі

Місто

Боярка

Телефон

+380...

Адреса

вулиця Лесі Українки 65

Це автосалон?

Скасувати

Створити

Рис. 3.11 - Попап для створення нової локації автосалону

Після збереження даних уся створена мережа філій та представництв відображається у вигляді структурованого списку на відповідній сторінці, що дає партнеру чітке розуміння розподілу його точок обслуговування (див. рис. 3.12).

Точки видачі

Керуйте локаціями, де клієнти можуть забрати авто

Додати точку

Пошук за назвою або ...

Фільтр за містом

Тип локації

Назва	Місто	Адреса	Телефон	Дії
Головний офіс Житомир	Житомир	Хлібна 3	+380000000000	🔗 🗑️
New pickup Салон	Львів	Пильницька вулиця 2	fowefwet	🔗 🗑️
асвімівмі	Костопіль	вулиця Симона Петлюри 10	ушцццц32	🔗 🗑️
орнігми Салон	Київ	вулиця Богдана Хмельницького 4	+380000000000	🔗 🗑️

Рис. 3.12 - Сторінка відображення створених локацій автосалонів

Наявність зареєстрованих локацій є обов'язковою технічною умовою для наповнення автопарку. Переходячи до сторінки «Автомобілі», партнер отримує можливість додавати нові транспортні засоби до системи, обов'язково закріплюючи кожен автомобіль за конкретним створеним салоном (локацією), де машина фактично перебуватиме та очікуватиме на клієнта (див. рис. 3.13).

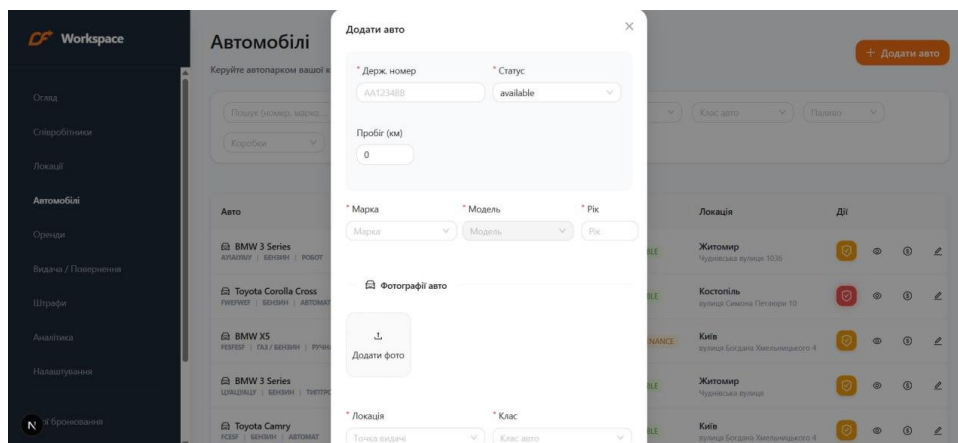


Рис. 3.13- Інтерфейс додавання автомобіля та прив'язки до локації

Контроль за фізичним процесом передачі ключів та повернення майна покладено на сторінку «Видачі авто». Цей інтерфейс автоматизує роботу лінійного персоналу на локаціях, фіксуючи точний час передачі автомобіля клієнту, стан паливного бака, наявність пошкоджень під час повернення та закриття договору оренди (див. рис. 3.14).

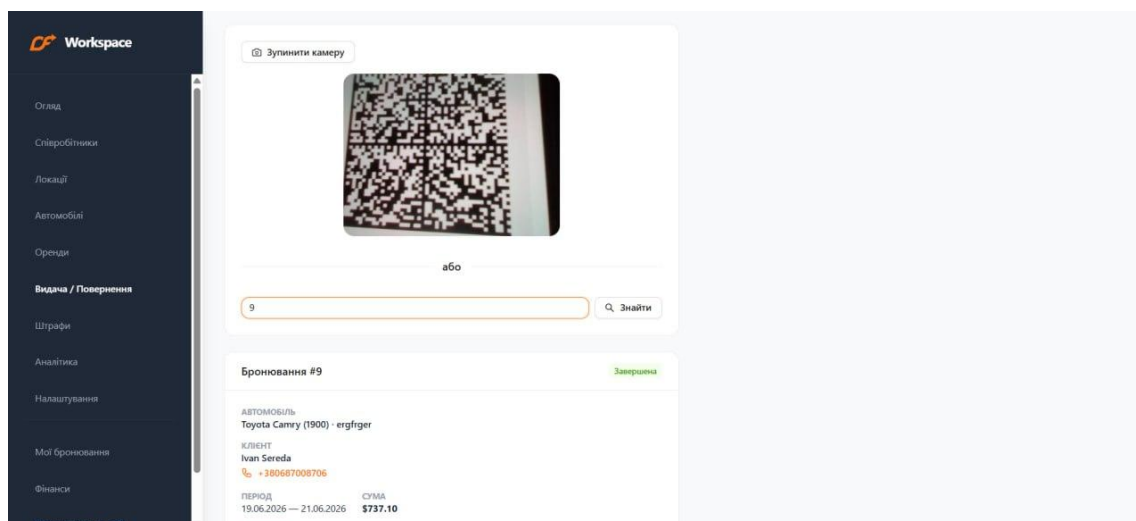


Рис. 3.14 - Сторінка обліку видачі та повернення автомобілів

Розділ «Налаштування» в робочому просторі дозволяє партнеру конфігурувати глобальні параметри своєї компанії: змінювати платіжні

реквізити, налаштовувати внутрішні правила розрахунку вартості, вказувати розміри штрафів за запізнення або пошкодження, а також керувати системними сповіщеннями (див. рис. 3.15).

Рис. 3.15 - Сторінка системних налаштувань Workspace

У випадку, якщо відвідувач платформи не є бізнес-партнером, для нього система працює в стандартному B2C-режимі (Business-to-Consumer). Такий користувач проходить спрощену реєстрацію як звичайний клієнт, після чого отримує повноцінний доступ до каталогу, може вільно переглядати картки машин, користуватися панеллю фільтрації та безпосередньо бронювати обраний автомобіль для власних потреб (див. рис. 3.16).

Рис. 3.16 - Сторінка відображення доступного автопарку для клієнтів

Безпосередній процес оформлення замовлення ініціюється на сторінці детальної картки автомобіля через спеціальний інтерактивний модуль

бронювання. Інтерфейс, який в адаптивній мобільній версії реалізовано у вигляді зручної висувної панелі, дозволяє клієнту чітко вказати бажаний діапазон дат оренди (див. рис. 3.17).

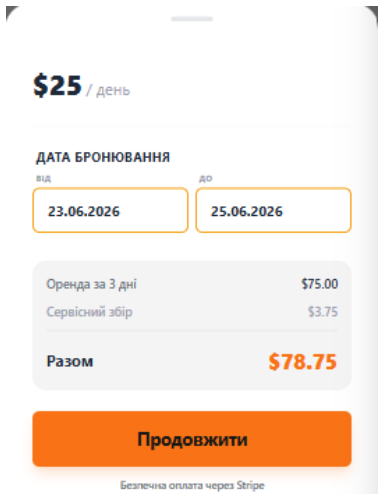


Рис. 3.17 - Інтерфейс модуля бронювання автомобіля та розрахунку вартості (мобільна версія)

Для безпомилкового введення дат оренди реалізовано інтерактивний компонент календаря: під час натискання на поля вибору дат відкривається модальне вікно, яке дозволяє візуально виділити бажаний безперервний період, що превентивно унеможливорює овербукінг. Вибраний проміжок підсвічується контрастним кольором для миттєвого візуального зворотного зв'язку (див. рис. 3.18).

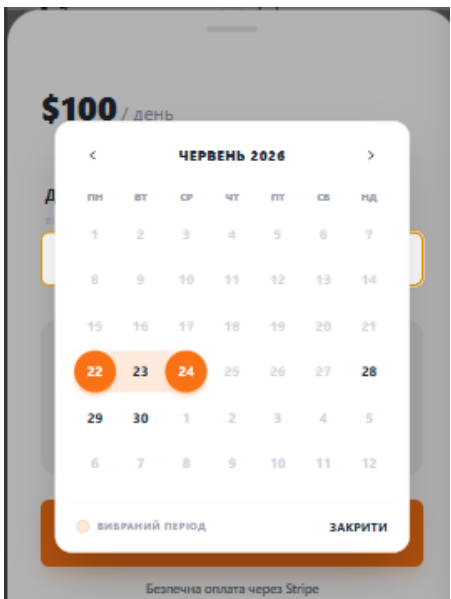


Рис. 3.18 - Інтерактивний календар для вибору періоду бронювання з динамічним відображенням доступних дат

Після підтвердження вартості оренди користувач переходить до безпечного розрахунку через інтегрований платіжний віджет Stripe. У модальному вікні «Оплата оренди» клієнт може скористатися швидким оформленням (Stripe Link зі збереженою карткою) або ввести нові реквізити, завершивши процес натисканням кнопки «Сплатити» (див. рис. 3.19).

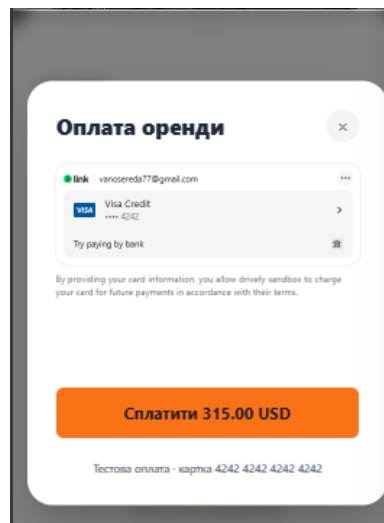


Рис. 3.19 - Модальне вікно безпечної оплати оренди через інтегрований шлюз Stripe

Після успішної оплати користувач перенаправляється на сторінку підтвердження з цифровою квитанцією, що містить номер бронювання, обране авто, період оренди, пункт видачі та сплачену суму. Центральним елементом є унікальний QR-код, який клієнт пред'являє менеджеру локації для отримання автомобіля (див. рис. 3.20).

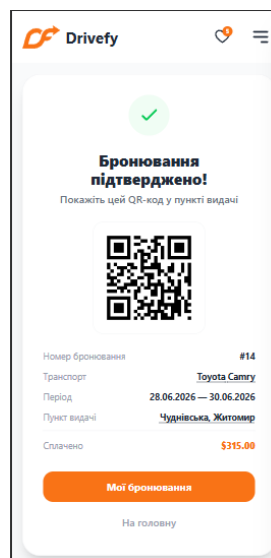


Рис. 3.20 - Сторінка підтвердження бронювання з QR-кодом для видачі авто

3.3 Тестування платформи

Процес тестування розробленої платформи оренди автомобілів є невід'ємною частиною життєвого циклу розробки (SDLC) і виступає головним інструментом гарантування надійності, стабільності та безпеки системи. Враховуючи модульну архітектуру, де клієнтська частина на Next.js та серверна на NestJS функціонують незалежно, а розробка здійснюється одноосібно одним full-stack фахівцем, стратегія забезпечення якості потребувала комплексного підходу. Впровадження автоматизованого багаторівневого тестування дозволило компенсувати відсутність окремої QA-команди, своєчасно виявляти дефекти на ранніх етапах та гарантувати продуктивність системи під час одночасного пошуку й бронювання авто багатьма користувачами.

Першочерговим етапом верифікації стало тестування серверного API за допомогою інструментарію Postman [30]. Оскільки бекенд відповідає за критичну бізнес-логіку, автентифікацію, роботу з базою даних та перевірку доступності автомобілів, кожен кінцеву точку (endpoint) перевіряли до інтеграції з інтерфейсом, що дозволило усунути помилки ще на етапі проєктування. Для цього створено структуровані колекції запитів (Collections), які повторювали логічну структуру REST API та охоплювали модулі користувачів, автомобілів, автопарку, фільтрації, штрафів і бронювань, а в межах кожної за допомогою Postman Sandbox розроблялися автоматизовані тест-сценарії з перевіркою HTTP-статусів і глибокою асерцією даних (валідація JSON-схем, обов'язкових полів, типів та швидкості відгуку). Завдяки динамічним середовищам (Environments) автоматизовано наскрізні сценарії: скрипт після авторизації вилучав JWT Access Token і підставляв його в заголовки Authorization: Bearer для всіх захищених запитів, що дозволило проводити ланцюжкові тести реальної поведінки користувачів - пошук авто за датами, розрахунок вартості, створення бронювання та зміну статусів оренди (див. рис. 3.21).

Схема процесу тестування API та автоматизації автентифікації в Postman



Рис. 3.21 - Схема процесу тестування API та автоматизації автентифікації в Postman

Зі збільшенням кількості кінцевих точок API ручне оновлення колекцій у Postman стало неефективним, що спонукало до впровадження Swagger (OpenAPI Specification). Його інтеграція в кодову базу NestJS дозволила автоматично генерувати інтерактивну документацію на основі декораторів та DTO, описуючи кожен маршрут, параметри й варіанти відповідей, причому Swagger розгортається разом із сервером і завжди відображає актуальний стан API (див. Додаток F). Цей веб-інтерфейс став єдиним джерелом істини щодо архітектури даних, що прискорило розробку клієнтської частини на Next.js та усунуло неузгодженість форматів між фронтендом і бекендом, а функція інтерактивного виконання запитів («Try it out») дозволила миттєво тестувати ендпоінти безпосередньо через браузер.

Наступним етапом забезпечення якості під час розробки інтерфейсу на Next.js стало впровадження UI-тестів для найважливіших інтерактивних компонентів, оскільки складна логіка взаємодії та динамічна зміна станів робили ручне тестування недостатнім. Для автоматизації обрано стек Vitest та React Testing Library, що рендерять компоненти в ізольованому віртуальному DOM та імітують поведінку користувача, покриваючи критичні вузли: форму пошуку й фільтрації, сторінку деталей, калькулятор вартості та форму бронювання. Ключовою особливістю стало ізолювання компонентів від

сервера через макетування (mocking) API-запитів, що дозволило створювати стабільні й швидкі тести, незалежні від бази даних, але здатні перевіряти реакцію інтерфейсу на будь-які відповіді (див. рис. 3.22).

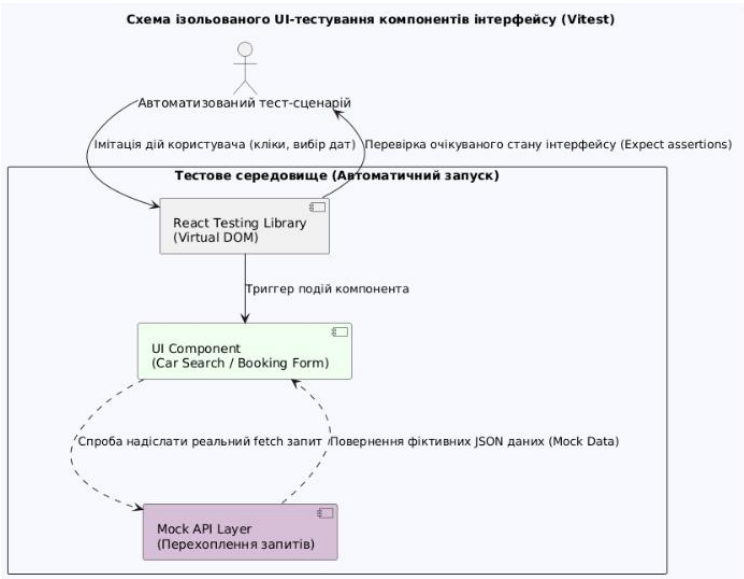


Рис. 3.22 - Схема ізолюваного UI-тестування компонентів інтерфейсу (Vitest)

Усі розроблені тестові модулі - серверні інтеграційні та клієнтські UI-тести - інтегровано в єдиний автоматизований CI/CD-конвеєр на базі GitLab. При кожному оновленні репозиторію система розгортає тестове середовище, запускає лінери, виконує Postman-колекції через Newman та проводить UI-тестування компонентів Next.js, а в разі невдачі хоча б одного тесту розгортання блокується, що виключає потрапляння нестабільного коду на продуктивний сервер. Така синергія Postman, Swagger та UI-тестів дозволила створити надійну систему забезпечення якості й полегшити супровід платформи одним спеціалістом.

Для верифікації найважливіших модулів розроблено серію тест-кейсів, що описують передумови, покрокові дії та еталонні реакції платформи для фільтрації автопарку й фінального оформлення броні. Оскільки бізнес-логіка критично залежить від обробки часових проміжків, розрахунку вартості та валідації даних, саме ці компоненти стали пріоритетними об'єктами перевірки, а формалізовані тест-кейси водночас слугують регресійною базою при подальшому розвитку.

Тест-кейс 1: Алгоритм пошуку та фільтрації

Параметр	Опис
ID тест-кейсу	ТК-ЦАР-01
Назва	Пошук та фільтрація автомобілів за тимчасовим проміжком та критеріями доступності
Пріоритет	Високий
Передумови	1. У базі даних є автомобілі різних класів (Економ, Бізнес, SUV). 2. Автомобіль А (наприклад, Toyota Corolla, Економ) уже заброньований на період з 22.06.2026 по 25.06.2026. 3. Користувач знаходиться на головній сторінці пошуку.
Кроки відтворення	1. У формі пошуку обрати дату початку оренди: 23.06.2026 та дату завершення: 26.06.2026 (період частково перетинається з чинним бронюванням Автомобіля А). 2. У фільтрі категорій обрати клас «Економ». 3. Натиснути кнопку «Знайти автомобіль».
Очікуваний результат	Система успішно обробляє запит і виводить список доступних машин. Автомобіль А відсутній у результатах видачі, оскільки дати перетинаються. Натомість відображаються інші вільні авто класу «Економ». Для кожного доступного авто коректно розраховано базову вартість оренди за вказані 3 доби.

Тест-кейс ТК-ЦАР-01 спрямований на верифікацію однієї з найбільш критичних функціональних складових платформи - алгоритму пошуку та фільтрації транспортних засобів за часовими проміжками. Головною метою цього сценарію є перевірка надійності механізму перевірки доступності автомобілів, що унеможливорює виникнення овербукінгу (double-booking - повторного бронювання вже зайнятого авто). Система повинна в реальному часі аналізувати масив наявних у базі даних бронювань, порівнювати їхні часові межі із запитом користувача та динамічно виключати з результатів видачі ті транспортні засоби, які вже зафрахтовані на обрані дати (навіть якщо перетинається лише один день). Окрім цього, сценарій валідує роботу супутніх фільтрів за класом авто (економ, бізнес тощо) та правильність базового математичного розрахунку вартості оренди на бекенді NestJS.

Успішне проходження цього тест-кейсу підтверджує стійкість ізоляції транзакцій у базі даних та гарантує коректність відображення цінових діапазонів у користувацькому інтерфейсі під час пікових навантажень на сервер.

Таблиця 3.2

Тест-кейс 2: Розрахунок фінальної вартості та оформлення бронювання

Параметр	Опис
ID тест-кейсу	ТК-ЦАР-02
Назва	Калькуляція повної вартості замовлення
Пріоритет	Критичний
Передумови	Користувач авторизований у системі, обрав доступний автомобіль (базова ціна - \$50/доба, термін - 2 доби) та перейшов на сторінку оформлення замовлення.
Кроки відтворення	1. Перевірити початкову суму в блоці «Разом» (має бути \$100 за дві доби). 3. Натиснути кнопку «Підтвердити бронювання».
Очікуваний результат	Після кроку 2: Сума в блоці «Разом» динамічно оновлюється на клієнті в реальному часі й становить \$120 (\$100 авто + \$20 сервісний збір). Після кроку 3: Система блокує відправку форми, підсвічує поле червоним і виводить помилку Після кроку 4: Бронювання успішно створюється в БД зі статусом «Очікує оплати», користувача перенаправляє на платіжний шлюз, а цей автомобіль на обрані дати блокується для пошуку іншими клієнтами.

Тест-кейс ТК-ЦАР-02 призначений для комплексної перевірки калькуляції повної вартості замовлення та валідації клієнтської форми бронювання. Сценарій тестує здатність системи динамічно обчислювати базову суму (наприклад, \$100 за дві доби) та оновлювати її в реальному часі при нарахуванні сервісного збору (до \$120). Водночас перевіряється захисна логіка інтерфейсу: система повинна блокувати відправку форми та підсвічувати помилку, якщо користувач намагається підтвердити бронювання з порушенням умов. За умови успішної валідації підтверджується коректне створення запису в базі даних зі статусом «Очікує оплати», перенаправлення

клієнта на платіжний шлюз та негайне блокування автомобіля для пошуку іншими користувачами.

Додатково, за допомогою автоматизованих UI-тестів (Vitest та React Testing Library), перевіряється ізольована поведінка цих фронтенд-компонентів: коректність рендерингу калькулятора, імітація кліків користувача та миттєве відображення станів помилок або завантаження (див рис. 3.23).

HeaderWrapper.tsx	0	0	0	0	1-18
MobileDrawer.tsx	0	0	0	0	1-246
RoleGuard.tsx	100	90.9	100	100	26,37
StaffActions.tsx	0	0	0	0	1-39
StoreProvider.tsx	0	0	0	0	1-38
UserActions.tsx	0	0	0	0	1-97
src/components/error	0	0	0	0	
NotFoundIllustration.tsx	0	0	0	0	1-31
src/components/features/reviews	100	100	100	100	
ReviewCard.tsx	100	100	100	100	
ReviewList.tsx	100	100	100	100	
StarDisplay.tsx	100	100	100	100	
src/components/ui	31.58	90.74	65.51	31.58	
BaseTable.tsx	0	0	0	0	1-56
Button.tsx	100	100	100	100	
ContactButton.tsx	100	100	100	100	
DatePicker.tsx	0	0	0	0	1-225
DateRangePicker.tsx	0	0	0	0	1-312
Dropdown.tsx	100	100	100	100	
Hi.tsx	100	100	100	100	
ImageUpload.tsx	0	0	0	0	1-82
Input.tsx	100	100	100	100	
LikeButton.tsx	100	100	100	100	
MapPicker.tsx	0	0	0	0	1-142
NavLink.tsx	100	100	100	100	
PickupPointsMap.tsx	0	0	0	0	1-123
PriceRangeInput.tsx	100	95.83	66.66	100	40
SearchableSelect.tsx	0	0	0	0	1-98
Select.tsx	100	88.23	87.5	100	48,72,124,131,166,207
WorkspaceDateRangePicker.tsx	0	0	0	0	1-52
src/components/workspace	0	0	0	0	
WorkspaceSidebar.tsx	0	0	0	0	1-176
src/features/auth/api	0	0	0	0	

Рис. 3.23 - Результати проходження UI-тестів

Висновки до третього розділу

У третьому розділі кваліфікаційної роботи успішно реалізовано інфраструктуру розгортання платформи на базі Docker, Docker Compose та Nginx із шифруванням HTTPS, а також впроваджено автоматизований CI/CD-конвеєр. Розроблено адаптивний інтерфейс для клієнтів та комплексний робочий простір для партнерів, що автоматизує управління автопарком, локаціями, персоналом, орендами та аналітикою. Завдяки багаторівневому тестуванню (Postman, Swagger UI, Vitest) та виконанню тест-кейсів повністю верифіковано критичну бізнес-логіку системи, підтверджено безпомилковість динамічного розрахунку вартості й надійність захисту від овербукінгу, що гарантує стабільне та безпечне функціонування платформи.

ВИСНОВКИ

У межах кваліфікаційної роботи спроектовано та реалізовано цифрову платформу оренди автотранспорту Drivefy - гібридний B2B2C-маркетплейс, де на базі спільного програмного ядра інтегровано публічний вебсайт для клієнтів та хмарне SaaS-рішення для автоматизації бізнес-процесів орендодавців. На основі порівняльного аналізу ринку каршерингу та P2P-платформ сформовано ключові вимоги до системи: підтримку рольової моделі доступу, цифровізацію логістики через QR-перепустки, автоматизацію контролю доступності транспорту та безпеку онлайн-платежів, для чого обґрунтовано клієнт-серверну архітектуру на базі REST API.

Для реалізації обрано сучасний технологічний стек: клієнтську частину побудовано на Next.js та Ant Design, що забезпечило баланс між серверним рендерингом публічних сторінок і реактивністю панелей керування, серверну логіку - на NestJS, а транзакційну цілісність даних покладено на PostgreSQL з ORM Sequelize. Фінансовий цикл інтегровано через платіжний шлюз Stripe з токенизацією даних.

Проектний етап охопив побудову поведінкової та структурної моделей: за допомогою діаграм Use Case виділено п'ять типів акторів із ієрархічним успадкуванням повноважень, спроектовано логічну схему бази даних із 20 таблиць із мультитенантністю через систему зовнішніх ключів, а діаграмами активностей формалізовано п'ять фундаментальних алгоритмів - stateless-автентифікацію на JWT, рольовий доступ (RBAC), багатовимірну фільтрацію авто із захистом від овербукінгу, синхронізацію фільтрів з URL та керування списком обраного. На завершальному етапі створено адаптивний інтерфейс каталогу та робочий простір партнерів, організовано інфраструктуру розгортання на базі Docker, Nginx із HTTPS від Let's Encrypt та CI/CD-конвеєром, а комплексне тестування підтвердило коректність бізнес-логіки й розрахунку вартості. Розроблена платформа є масштабованою та безпечною, а її модульна архітектура закладає надійне підґрунтя для подальшого комерційного розвитку.

					ІІЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						82

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Next.js documentation [Електронний ресурс]. - URL: <https://nextjs.org/docs> (дата звернення: 24.03.2026).
2. SSR - SSG - ISR - CSR in Next.js - The Ultimate Guide [Електронний ресурс]. - URL: <https://dev.to/idrazhar/ssr-ssg-isr-csr-in-nextjs-the-ultimate-guide-256m> (дата звернення: 20.03.2026).
3. The Complete Guide to SEO Optimization in Next.js 16 [Електронний ресурс]. - URL: <https://medium.com/@thomasaugot/the-complete-guide-to-seo-optimization-in-next-js-15-1bdb118cffd7> (дата звернення: 02.03.2026).
4. Stripe API Reference [Електронний ресурс]. - URL: <https://stripe.com/docs/api> (дата звернення: 16.03.2026).
5. Introduction to Manual Testing - Software Testing - GeeksforGeeks. [Електронний ресурс] - URL: <https://www.geeksforgeeks.org/software-testing/software-testing-manual-testing/> (дата звернення: 15.03.2026).
6. Getmancar [Електронний ресурс]. - URL: <https://getmancar.com.ua/> (дата звернення: 01.03.2026).
7. NarsCars [Електронний ресурс]. - URL: <https://narscars.com.ua/> (дата звернення: 01.03.2026).
8. 7Cars [Електронний ресурс]. - URL: <https://7cars.com.ua/> (дата звернення: 01.03.2026).
9. Їду сам [Електронний ресурс]. - URL: <https://idycam.com> (дата звернення: 02.03.2026).
10. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures (REST) [Електронний ресурс]. - URL: https://ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (дата звернення: 04.05.2026).
11. Using HTTP cookies - MDN Web Docs [Електронний ресурс]. - URL: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies> (дата звернення: 11.05.2026).

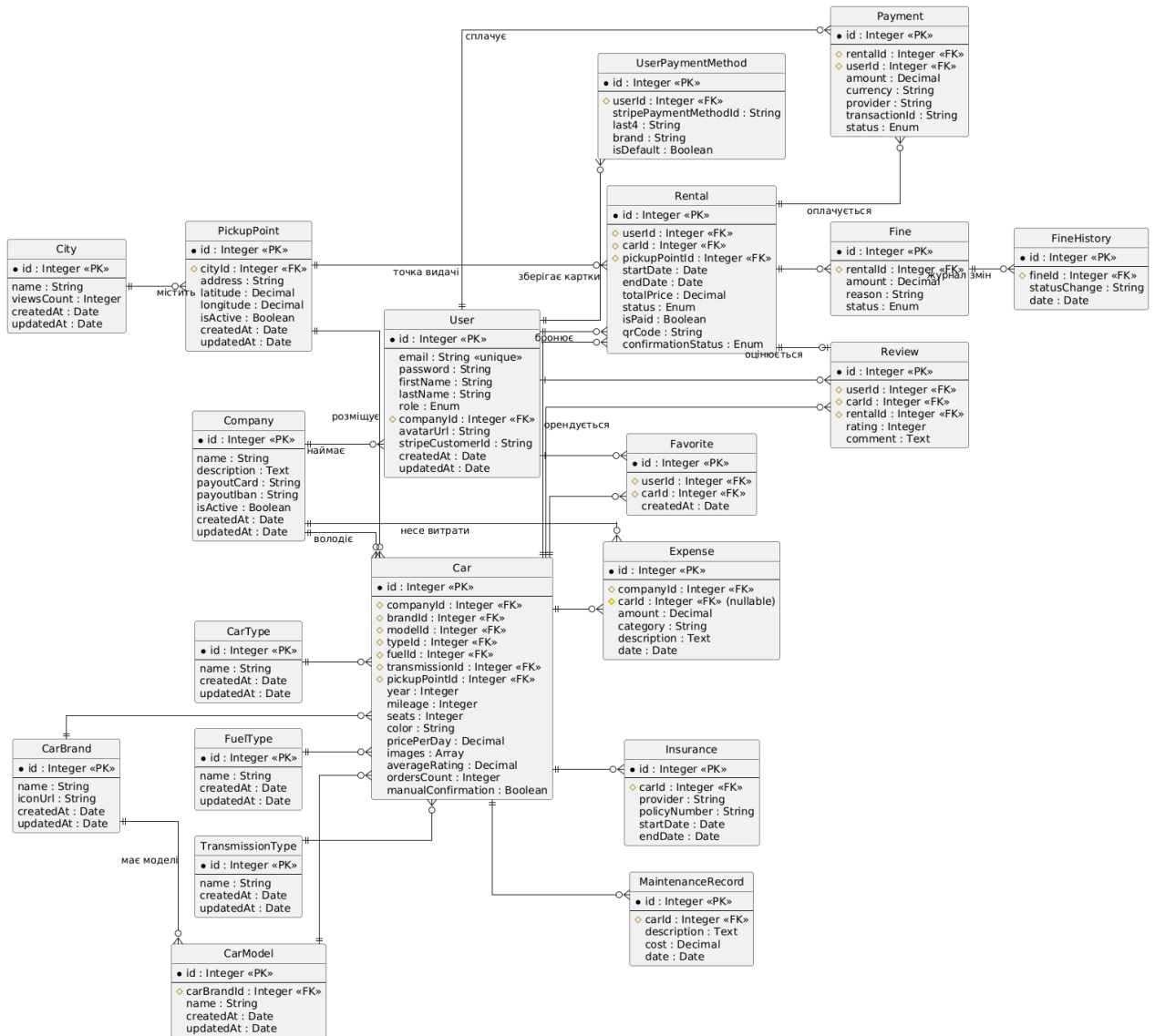
12. RTK Query Overview. [Электронный ресурс] - URL: <https://redux-toolkit.js.org/rtk-query/overview> (дата звернения: 26.03.2026).
13. Redux Toolkit documentation [Электронный ресурс]. - URL: <https://redux-toolkit.js.org/> (дата звернения: 21.03.2026).
14. Node.js Official documentation [Электронный ресурс]. - URL: <https://nodejs.org/en/docs/> (дата звернения: 15.03.2026).
15. NestJS - A progressive Node.js framework [Электронный ресурс]. - URL: <https://docs.nestjs.com/> (дата звернения: 10.03.2026).
16. Develop clean MVC web applications with Nest.js [Электронный ресурс]. - URL: <https://medium.com/@danielgara/develop-clean-mvc-web-applications-with-nest-js-797bdcafd7f5> (дата звернения: 15.04.2026).
17. PostgreSQL Official documentation [Электронный ресурс]. - URL: <https://www.postgresql.org/docs/> (дата звернения: 02.03.2026).
18. JSON Web Token (JWT) Introduction [Электронный ресурс]. - URL: <https://jwt.io/introduction> (дата звернения: 14.04.2026).
19. Role-Based Access Control (RBAC) - NIST [Электронный ресурс]. - URL: <https://csrc.nist.gov/projects/role-based-access-control> (дата звернения: 25.05.2026).
20. Cross-Site Scripting (XSS) - MDN Web Docs [Электронный ресурс]. - URL: https://developer.mozilla.org/en-US/docs/Glossary/Cross-site_scripting (дата звернения: 18.05.2026).
21. Docker documentation [Электронный ресурс]. - URL: <https://docs.docker.com/> (дата звернения: 07.03.2026).
22. GitLab CI/CD documentation [Электронный ресурс]. - URL: <https://docs.gitlab.com/ee/ci/> (дата звернения: 27.04.2026).
23. React documentation: The library for web and native user interfaces. [Электронный ресурс] - URL: <https://react.dev/learn> (дата звернения: 04.04.2026).
24. Sequelize v6 Official documentation [Электронный ресурс]. - URL: <https://sequelize.org/docs/v6/> (дата звернения: 02.03.2026).

25. Vitest: A blazing fast unit test framework powered by Vite [Електронний ресурс]. - URL: <https://vitest.dev/guide/> (дата звернення: 08.04.2026).
26. What is Multitenancy? Meaning, architecture, benefits & risks | Blog - Northflank [Електронний ресурс]. - URL: <https://northflank.com/blog/what-is-multitenancy> (дата звернення: 03.03.2026).
27. A Future-Adaptable Password Scheme (Bcrypt) [Електронний ресурс]. - URL: https://www.usenix.org/legacy/events/usenix99/provos/provos_html/node1.html (дата звернення: 29.05.2026).
28. Swagger: API documentation & Design Tools for Teams [Електронний ресурс]. - URL: <https://swagger.io/> (дата звернення: 23.05.2026).
29. UI тестування: що це і навіщо воно потрібно [Електронний ресурс]. - URL: <https://megasite.ua/ua/ui-testirovanie-chto-eto-takoe-i-zachem-ononugno> (дата звернення: 25.05.2026).
30. Postman API Platform: Build, test, and document APIs [Електронний ресурс]. - URL: <https://www.postman.com/> (дата звернення: 27.05.2026).

ДОДАТКИ

					ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
						86

Додаток А. «Діаграма бази даних»

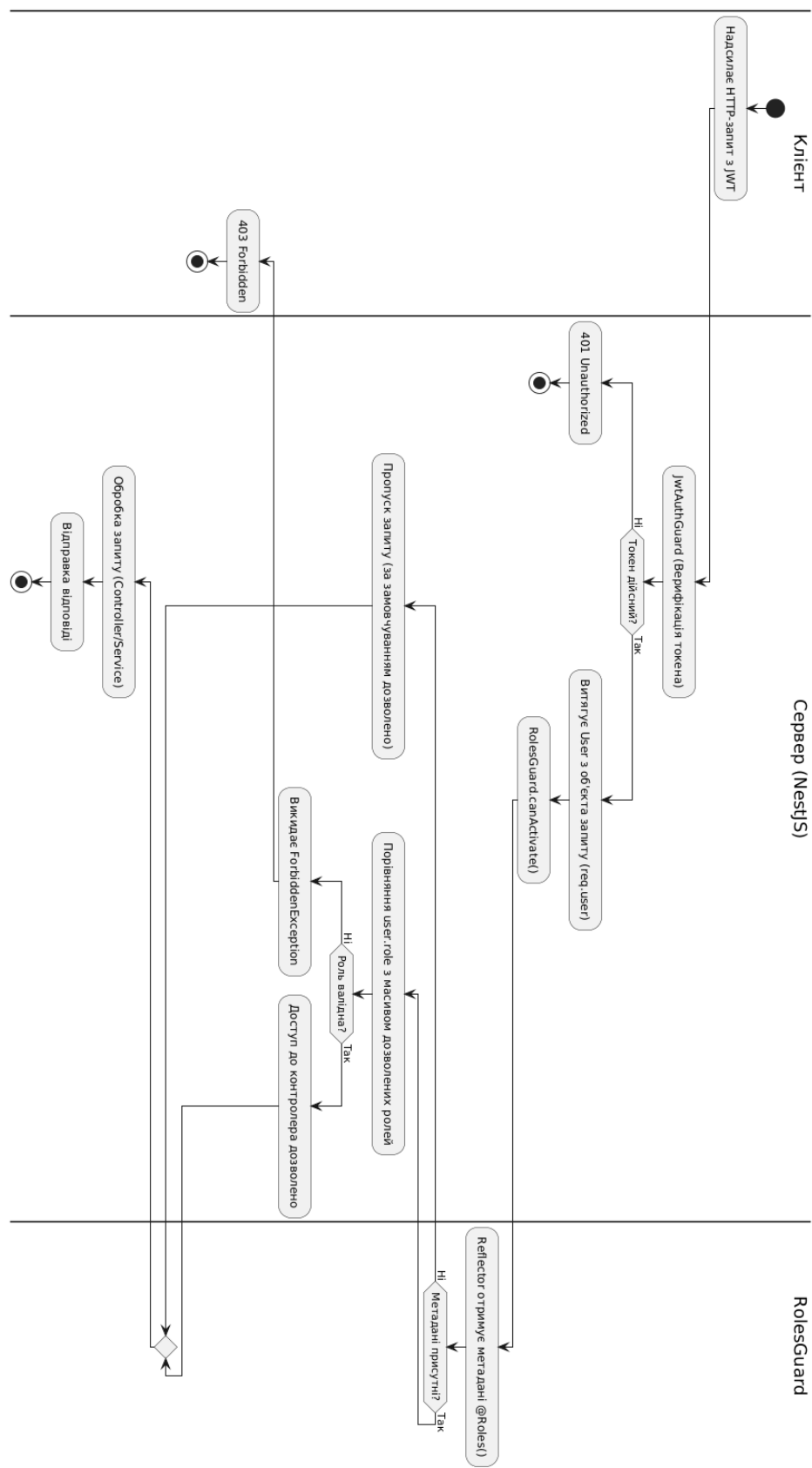


Додаток В. Лістинг сервісу автентифікації

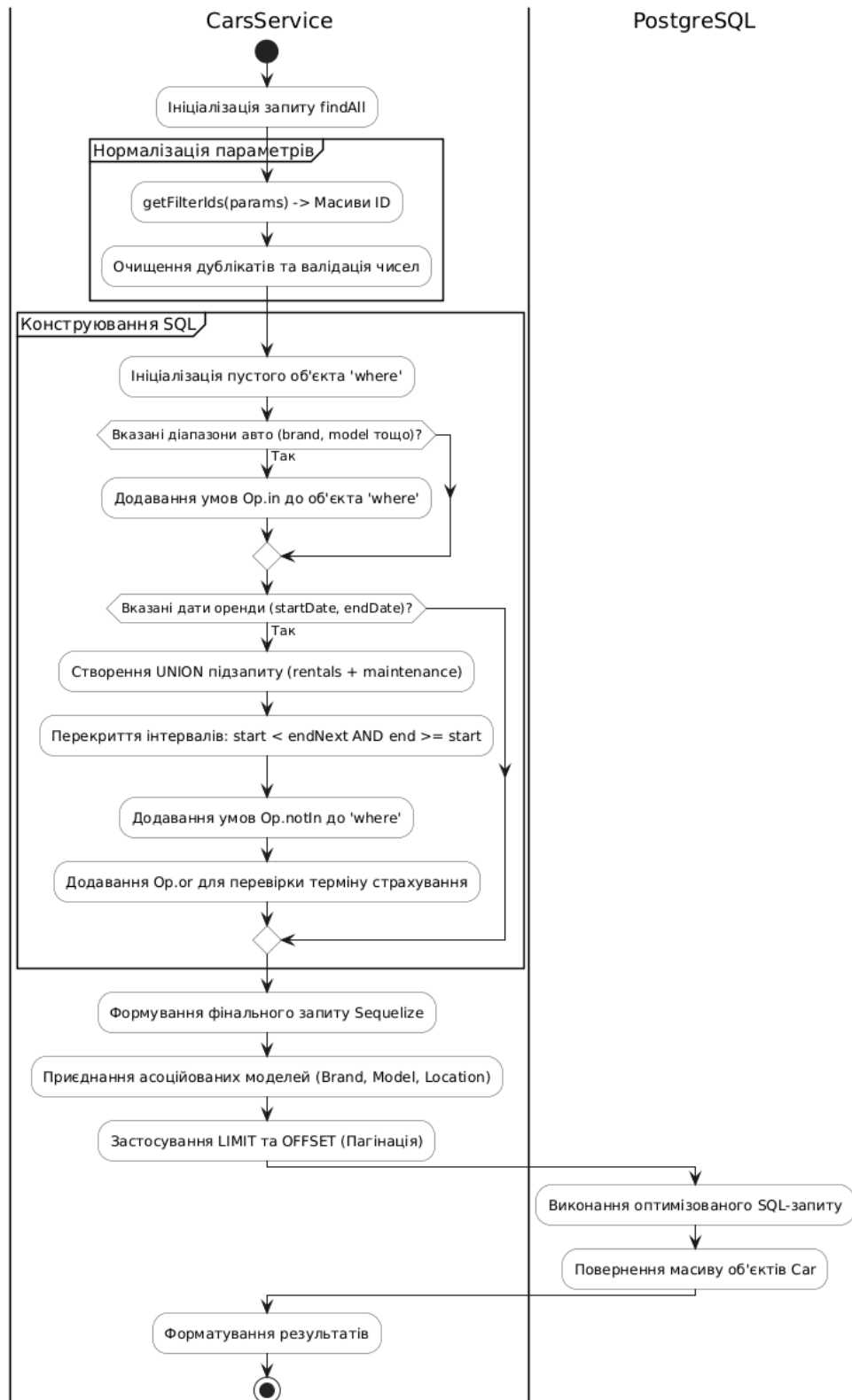
```
export class AuthService {
  async validateUser(email: string, pass: string): Promise<any> {
    const user = await this.usersService.findByEmail(email);
    if (user && (await bcrypt.compare(pass, user.password))) {
      const { password, ...result } = user.get({ plain: true });
      return result; // хеш пароля ніколи не повертається назовні
    }
    return null;
  }
  async login(user: any) {
    const payload = {
      email: user.email, sub: user.id,
      role: user.role, companyId: user.companyId,
    };
    return { access_token: this.jwtService.sign(payload) };
  }
}

export class JwtStrategy extends PassportStrategy(Strategy) {
  constructor(private configService: ConfigService) {
    super({
      jwtFromRequest: ExtractJwt.fromExtractors([
        (req: Request) => req?.cookies?.token, // 1. HTTP-only cookie
        ExtractJwt.fromAuthHeaderAsBearerToken(), // 2. Bearer header
      ]),
      ignoreExpiration: false,
      secretOrKey: configService.get('JWT_SECRET'),
    });
  }
  async validate(payload: any) {
    return { id: payload.sub, email: payload.email,
      role: payload.role, companyId: payload.companyId };
  }
}
```

Додаток С. Діаграма активностей «Алгоритм рольового розмежування доступу»



Додаток D. Діаграма активностей «Алгоритм багатовимірної фільтрації автомобілів»



Додаток Е. Лістинг алгоритму фільтрації з перевіркою доступності

```
const getFilterIds = (plural?: any[], singular?: any): number[] => {
  const ids: number[] = [];
  if (Array.isArray(plural)) ids.push(...plural);
  else if (plural) ids.push(Number(plural));
  if (Array.isArray(singular)) ids.push(...singular);
  else if (singular) ids.push(Number(singular));
  return [...new Set(ids)].filter(id => !isNaN(id));
};

if (startDate && endDate) {
  const endNext = new Date(endDate);
  endNext.setUTCDate(endNext.getUTCDate() + 1); // +1 день для граничних умов
  const busyCarIds = literal(`
    SELECT «carId» FROM «rentals»
      WHERE «status» != :cancelledStatus
        AND «start_date» < :availEndNext
        AND «end_date»   >= :availStart
    UNION
    SELECT «carId» FROM «maintenance_records»
      WHERE «dateStart» < :availEndNext
        AND «dateEnd»   >= :availStart
  `);
  where.id = { [Op.notIn]: busyCarIds };
  where[Op.and] = [...(where[Op.and] ?? [])], {
    [Op.or]: [
      { insuranceExpiryDate: null },
      { insuranceExpiryDate: { [Op.gte]: endDate } },
    ],
  },
}
```

Додаток F. Діаграма компонентів «Архітектура інтеграції Swagger UI у процес solo-розробки»

Архітектура інтеграції Swagger UI у процес solo-розробки

