

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка веб-орієнтованої інформаційної системи для
подовгової оренди автомобілів»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

*Кваліфікаційна робота містить результати власних досліджень. Використання
ідей, результатів і текстів інших авторів мають посилання
на відповідне джерело*

_____ Владислав ГРИЦЮК
(підпис)

Виконав: здобувач вищої освіти групи ПД-41

_____ Владислав ГРИЦЮК

Керівник: _____ Владислав ЯСКЕВИЧ
канд.техн.наук, доц.

Рецензент: _____

Київ 2026

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення

_____ Ірина ЗАМРІЙ

« ____ » _____ 2026 р.

**ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Грицюку Владиславу Віталійовичу

1. Тема кваліфікаційної роботи: «Розробка веб-орієнтованої інформаційної системи для подового оренди автомобілів»

керівник кваліфікаційної роботи канд.техн.наук, доцент Владислав ЯСКЕВИЧ,
затверджені наказом Державного університету інформаційно-комунікаційних
технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «29» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про бізнес-процеси в сфері оренди автомобілів, технічна документація до використаних технологій веб-розробки.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної галузі оренди автомобілів, ознайомлення з існуючими рішеннями, визначення вимог до програмного забезпечення.

2. Огляд та аналіз засобів реалізації застосунку для оренди автомобілів.

3. Проектування архітектури інформаційної системи.

4. Програмна реалізація та тестування веб-застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Діаграма класів сутностей.
6. ER-діаграма бази даних.
7. Діаграма класів логіки.
8. Діаграма послідовності пошуку.
9. Діаграма послідовності процесу бронювання.
10. Екранні форми.
11. Апробація результатів дослідження

6. Дата видачі завдання «23» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	23.02-28.02.2026	
2	Аналіз та дослідження існуючих аналогів	01.03-07.03.2026	
3	Аналіз предметної галузі оренди автомобілів та визначення вимог до програмного забезпечення	08.03-12.03.2026	
4	Огляд та аналіз засобів реалізації застосунку для оренди автомобілів	13.03-22.03.2026	
5	Проектування архітектури застосунку для оренди автомобілів	23.03-06.04.2026	
6	Програмна реалізація та тестування застосунку	07.04-21.04.2026	
7	Оформлення роботи: вступ, висновки, реферат	22.04-02.05.2026	
8	Розробка демонстраційних матеріалів	03.05-10.05.2026	
9	Попередній захист роботи	11.05-22.05.2026	
10	Подання кваліфікаційної роботи	22.05-29.05.2026	

Здобувач вищої освіти

(підпис)

Владислав ГРИЦЮК

Керівник
кваліфікаційної роботи

(підпис)

Владислав ЯСКЕВИЧ

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 75 стор., 2 табл., 41 рис., 21 джерело.

Мета роботи – покращення якості процесів пошуку, бронювання, обліку та управління послугами прокату авто шляхом проєктування та розробки веб-орієнтованої інформаційної системи.

Об'єкт дослідження – процеси пошуку, бронювання, обліку та управління послугами подового прокату авто.

Предмет дослідження – методи, засоби і технології веб-програмування, що застосовуються для забезпечення процесів пошуку, бронювання, обліку та управління подовим прокатом авто.

Короткий зміст роботи: В роботі описано створення веб-орієнтованої інформаційної системи для подової оренди автомобілів, а також проаналізовано технології і методи веб-розробки, що були для цього використані. Проаналізовано існуючі програмні рішення на ринку прокату автомобілів. Розроблено архітектуру системи “Wheelix” та програмно реалізовано ключові функціональні можливості, зокрема: управління каталогом автомобілів, можливість пошуку і бронювання авто, управління бронюваннями, безпечна автентифікація на базі JWT із розподілом доступу за ролями. Розроблено та впроваджено підсистему інтелектуального пошуку автомобілів вільною мовою, яка використовує механізм семантичного парсингу запитів для генерації динамічних SQL-запитів. Проведено функціональне та модульне тестування веб-застосунку. В роботі використано бібліотеку React та фреймворк Next.js для створення клієнтської частини, Spring Boot для серверної частини, СУБД PostgreSQL для роботи з базою даних.

Сферою використання застосунку є автоматизація та оптимізація бізнес-процесів компаній, що надають послуги подової оренди автомобілів, а також забезпечення цифровізованого сервісу підбору і бронювання транспортних засобів для кінцевих клієнтів.

КЛЮЧОВІ СЛОВА: ОRENDA АВТОМОБІЛІВ, ВЕБ-ЗАСТОСУНОК, SPRING BOOT, REACT, NEXT.JS, POSTGRESQL, LLM, ОБРОБКА ПРИРОДНОЇ МОВИ, JWT

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	9
ВСТУП.....	10
1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ	13
1.1 Аналіз бізнес-процесів подовгової оренди транспортних засобів	13
1.2 Аналіз цільової аудиторії сервісу	15
1.3 Дослідження існуючих аналогів на ринку України.....	16
1.3.1 Укр-Прокат	16
1.3.2 BLS	19
1.3.3 Megarent	22
1.3.4 7cars	24
1.3.5 Зведена таблиця результатів аналізу.....	26
1.4 Формування вимог до розроблюваної інформаційної системи	27
1.4.1 Функціональні вимоги.....	27
1.4.2 Нефункціональні вимоги.....	28
1.5 Обґрунтування вибору технічних засобів та технологій розробки	29
1.5.1 Середовища розробки.....	29
1.5.2 Мови програмування	30
1.5.3 Фреймворки та бібліотеки.....	31
1.5.4 Зберігання даних	31
1.5.5 Засоби контролю версій та сторонні сервіси	32
2 ПРОЄКТУВАННЯ СИСТЕМИ.....	33
2.1 Діаграма варіантів використання	33
2.2 Архітектура системи.....	34
2.3 Архітектура серверної частини системи	36
2.4 Архітектура клієнтської частини системи.....	38
2.5 Архітектура бази даних	40
2.6 Алгоритм роботи модуля семантичного пошуку	42
3 ОПИС РЕАЛІЗАЦІЇ СИСТЕМИ.....	45
3.1 Реалізація серверної частини	45

3.2 Реалізація клієнтської частини	52
3.3 Результати та демонстрація	56
3.3.1 Головна сторінка	56
3.3.2 Каталог автомобілів	56
3.3.3 Сторінка автомобіля	58
3.3.4 Особистий кабінет користувача	59
3.3.5 Мої бронювання	61
3.3.6 Контакти.....	61
3.3.7 Управління автопарком.....	61
3.3.8 Управління категоріями	63
3.3.9 Управління бронюваннями	63
3.3.10 Управління користувачами.....	64
3.3.11 Реєстрація та вхід.....	64
4 ТЕСТУВАННЯ СИСТЕМИ	66
4.1 Обґрунтування видів тестування.....	66
4.2 Модульне тестування.....	67
4.3 Ручне тестування.....	68
4.4 Тестування під навантаженням	71
ВИСНОВКИ.....	73
ПЕРЕЛІК ПОСИЛАНЬ	75
ДОДАТОК А. ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ.....	77
ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ.....	89

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – База даних

ПЗ – Програмне забезпечення

API – Application Programming Interface

CMS – Content Management System

IDE – Integrated Development Environment

JWT – JSON Web Token

LLM – Large Language Model

RBAC – Role-Based Access Control

REST – Representational State Transfer

SEO – Search Engine Optimization

SQL – Structured Query Language

SSR – Server-Side Rendering

UI – User Interface

UX – User Experience

ВСТУП

Актуальність: у сучасному світі концепція “мобільність як послуга” (mobility as a service) набуває все більшого поширення, розширюючи звичні підходи до використання транспорту. Гнучкі та доступні сервіси тимчасового користування автомобілем стають дедалі зручнішою опцією для комфортного вирішення повсякденних та специфічних завдань.

Ця послуга є особливо актуальною для туристів, які потребують свободи пересування та незалежності від розкладів громадського транспорту під час подорожей. Не менший попит формують спеціалісти у бізнес-поїздках, для яких критично важливими є мобільність, пунктуальність та статусність транспортного засобу. Крім того, цільовою аудиторією виступають автолюбители, які бажають протестувати нові моделі на практиці перед можливою покупкою або просто орендувати авто певного класу для особливих подій чи подорожей.

Наявні на ринку традиційні сервіси оренди часто вимагають зайвих витрат часу на бюрократичні процедури та пропонують не надто зручні цифрові рішення. Актуальність розробки інформаційної системи “Wheelix” зумовлена необхідністю створення повністю цифровізованого процесу прокату автомобілів. Завдяки впровадженню безпечної віддаленої верифікації, прозорому управлінню бронюваннями та зручному особистому кабінету, система суттєво спрощує взаємодію клієнта з сервісом.

Додатковою перевагою, що якісно покращує користувацький досвід, є використання інструментів інтелектуального пошуку, які дозволяють підбирати авто за допомогою інтуїтивних запитів вільною мовою. Такий комплексний підхід не лише оптимізує операційну діяльність компаній з прокату, але й забезпечує швидке, сучасне та комфортне обслуговування кінцевих споживачів.

Об’єктом дослідження є процеси пошуку, бронювання, обліку та управління послугами подового прокату авто.

Предметом дослідження є методи, засоби і технології веб-програмування, що застосовуються для забезпечення процесів пошуку, бронювання, обліку та управління подовим прокатом авто.

Метою роботи є покращення якості процесів пошуку, бронювання, обліку та управління послугами прокату авто шляхом проєктування та розробки веб-орієнтованої інформаційної системи.

Методи дослідження: першим етапом дослідження стало вивчення бізнес-процесів сфери подової оренди автомобілів, а також аналіз конкурентного середовища та наявних на ринку сервісів прокату. Це дозволило чітко сформулювати функціональні та нефункціональні вимоги до майбутньої інформаційної системи. Далі було проведено огляд сучасного стеку технологій, здатного забезпечити виконання поставлених завдань. Для реалізації надійної серверної частини було обрано Java-фреймворк Spring Boot.

Для розробки клієнтської частини використано бібліотеку React у поєднанні з фреймворком Next.js, що забезпечує високу продуктивність та адаптивність інтерфейсу. PostgreSQL обрано як реляційну систему управління базами даних для збереження інформації про автопарк, користувачів та бронювання.

Для зберігання фотографій авто і конфіденційних документів обрано об'єктне сховище AWS S3, що забезпечує надійний рівень захисту інформації. Для реалізації підсистеми інтелектуального пошуку автомобілів застосовано інтеграцію з великими мовними моделями через API.

Обмін даними між клієнтським та серверним компонентами системи об'єднано за допомогою архітектурного стилю REST. Архітектуру програмного забезпечення спроектовано та візуалізовано з використанням UML-діаграм. Реалізацію функціоналу здійснено поетапно із подальшим тестуванням.

Наукова новизна роботи полягає у розробці комплексної веб-орієнтованої інформаційної системи, що надає вичерпний набір інструментів для цифровізації клієнтського досвіду. Головним елементом науково-практичної новизни є проєктування та впровадження підсистеми інтелектуального семантичного пошуку на базі великих мовних моделей (LLM).

Практична значущість результатів полягає у можливості використання розробленого веб-застосунку “Wheelix” для автоматизації роботи компаній з подової оренди автомобілів. Впровадження системи дозволяє підприємствам централізовано управляти автопарком та статусами замовлень, а клієнтам – проходити весь цикл підбору та оформлення оренди транспортного засобу в онлайн-форматі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз бізнес-процесів подовгової оренди транспортних засобів

У сучасних економічних умовах на ринку транспортних послуг спостерігається перехід від традиційної моделі обов'язкового володіння транспортним засобом до концепції “мобільність як послуга” (mobility as a service). Цей підхід передбачає надання доступу до засобів пересування за запитом споживача, що є особливо актуальним у сегменті короткострокової та подовгової оренди автомобілів [1].

Предметна область оренди автомобілів загалом охоплює комплекс взаємопов'язаних процесів, спрямованих на тимчасове передання транспортного засобу у користування клієнту. Ця задача має яскраво виражений двосторонній характер: з одного боку знаходиться підприємство (орендодавець), яке прагне максимізувати прибуток та репутацію, а з іншого – клієнт (орендар), який очікує отримати якісну послугу з мінімальними витратами часу та зусиль.

Аналізуючи загальні тенденції, варто відзначити, що цифровізація сфери надання послуг загалом і оренди автомобілів зокрема є ключовим вектором розвитку ринку [2]. Традиційні підходи з паперовим документообігом та ручним бронюванням через телефонні дзвінки стрімко відходять у минуле, поступаючись місцем комплексним інформаційним системам та застосункам.

Для розуміння проблематики та потреб користувачів доцільно розглянути життєвий цикл послуги оренди, порівнявши традиційний (офлайн) та цифровізований (онлайн) підходи до основних бізнес-процесів.

1. Пошук та підбір авто

- **Офлайн:** клієнт змушений телефонувати в офіс або особисто відвідувати стоянку. Підбір здійснюється зі слів менеджера, що обмежує можливість

порівняння характеристик та візуального огляду доступних варіантів.

- Онлайн: клієнт самостійно переглядає каталог через вебсайт або застосунок. Використання багатовимірних фільтрів та пошуку дозволяє миттєво знайти авто під специфічні потреби.

2. Бронювання

- Офлайн: фіксація дат оренди відбувається у паперових журналах або простих електронних таблицях. Існує високий ризик людської помилки – подвійного бронювання одного авто на ті самі дати.
- Онлайн: автоматична перевірка доступності транспортного засобу у режимі реального часу. Система самостійно блокує дати після підтвердження, усуваючи колізії.

3. Взаємодія з документами

- Офлайн: вимагає фізичної присутності клієнта для зняття ксерокопій паспорта та водійського посвідчення та витратного за часом ручного заповнення паперових договорів.
- Онлайн: можливість завантажити фото документів у систему заздалегідь, що значно прискорює процес.

4. Облік та адміністрування

- Офлайн: облік ведеться вручну, а процеси адміністрування, такі як підтвердження бронювання, вимагають кратно більше часу в порівнянні з цифровою альтернативою.
- Онлайн: персонал отримує уніфіковану платформу у вигляді застосунку з усіма необхідними інструментами адміністрування, часові затрати на операційні процеси зменшуються.

Аналіз описаних процесів наочно демонструє, що цифровізація є необхідною умовою виживання бізнесу на ринку автопрокату і що саме цифрові рішення здобудуть перевагу в умовах високого попиту. Впровадження спеціалізованих програмних рішень дозволяє значно змістити фокус з операційної рутини на покращення клієнтського досвіду. Для клієнта єдина цифрова екосистема означає

прозорість, передбачуваність та економію часу – цінності, що лежать в основі концепції “мобільність як послуга”. Для бізнесу ж це інструмент масштабування, оптимізації витрат та усунення людського фактора.

1.2 Аналіз цільової аудиторії сервісу

Успішне проєктування інформаційної системи вимагає глибокого розуміння потреб цільової аудиторії та застосування клієнтоорієнтованого підходу. Цільову аудиторію сервісу подовгової оренди автомобілів формує широкий спектр користувачів, яких об’єднує потреба в мобільності, проте відрізняють специфічні мотиви та очікування. Значну частку клієнтської бази становлять бізнес-користувачі та корпоративні клієнти, які понад усе цінують власний час, пунктуальність та відповідний статус транспортного засобу. Для цієї групи критично важливим є максимально швидке бронювання з повною відсутністю бюрократичних перепон. Іншим потужним сегментом є туристи та мандрівники, що прагнуть незалежності та свободи пересування під час поїздок. Оскільки вони часто не орієнтуються у складних технічних характеристиках автомобілів, для них ключовим фактором стає інтуїтивність вибору, наявність вичерпних описів та якісних візуальних матеріалів.

Окрім приїжджих, сервісом можуть активно користуватися і місцеві мешканці. Часто оренда потрібна їм для вирішення специфічних логістичних завдань (наприклад, поїздки за місто на вихідні) або ж для проведення своєрідних тест-драйвів перед прийняттям рішення про купівлю власного авто аналогічної моделі. Окрему, але надзвичайно стабільну групу клієнтів формують автовласники, які потребують тимчасової заміни машини. Їхні транспортні засоби можуть раптово вийти з ладу, потрапити в ДТП або перебувати на тривалому технічному обслуговуванні. Для таких користувачів оренда є вимушеним кроком задля збереження звичного ритму життя, тому головним критерієм стає оперативність надання підмінного авто. Крім того, варто виділити організаторів

заходів та приватних осіб, що шукають авто для особливих подій: весіль, урочистостей чи фотосесій, де на першому місці стоїть ідеальний естетичний стан та преміальність екстер'єру.

Незважаючи на настільки різні цілі та мотиви оренди, усі перелічені категорії користувачів мають абсолютно спільні технічні очікування щодо цифрового досвіду взаємодії із системою. Сучасний клієнт безапеляційно вимагає високої швидкості роботи та адаптивності, що гарантує миттєве завантаження та коректне відображення платформи як на десктопних, так і на мобільних пристроях. Важливою умовою довіри є абсолютна прозорість інформації: користувачі очікують бачити винятково актуальні дані про ціноутворення без прихованих комісій, реальні фотографії конкретних автомобілів, а також синхронізацію їхнього статусу у режимі реального часу.

Враховуючи необхідність завантаження конфіденційних документів, таких як паспорт чи водійське посвідчення, фундаментальною вимогою до системи стає гарантована безпека персональних даних. Зрештою, успішна платформа повинна забезпечити максимальну мінімізацію рутини, пропонуючи ергономічні інструменти пошуку і фільтрації, швидкий процес оформлення угоди та зручний особистий кабінет, де збереження необхідної контактної інформації та історії замовлень суттєво прискорить майбутні бронювання.

1.3 Дослідження існуючих аналогів на ринку України

В ході роботи проведено аналіз вебзастосунків для оренди авто, представлених на ринку України, а саме: Укр-Прокат, BLS, Megarent, 7cars. Далі представлений аналіз функцій таких застосунків, їх переваги і недоліки, а також екранні форми.

1.3.1 Укр-Прокат

Укр-Прокат – це одна з найбільших і найвідоміших національних компаній

на ринку України, що спеціалізується на наданні послуг прокату автомобілів без водія. Вебсайт компанії функціонує як основний канал залучення клієнтів та оформлення замовлень, орієнтуючись як на внутрішніх іноземних туристів, так і на корпоративний сегмент. Платформа забезпечує доступ до широкого автопарку з можливістю оренди у більшості великих міст та міжнародних аеропортів країни [3].

Основний функціонал веб-платформи включає можливість перегляду каталогу транспортних засобів, які розділені за фіксованими класами (від економ до преміум), та оформлення онлайн-бронювання із зазначенням міста, локації подачі та дат оренди. Користувачам також доступна опція додавання супутніх послуг (наприклад, дитяче автокрісло або GPS-навігатор).

Переваги:

- Структурований каталог та класифікація: автопарк компанії чітко розподілений за класами (економ, середній, бізнес, мікроавтобуси тощо). Це створює базову зручність для користувача, дозволяючи одразу перейти до потрібного сегмента автомобілів та ознайомитися з відповідними ціновими пропозиціями.
- Візуальне оформлення: UI-дизайн є лаконічним, не перевантаженим зайвими візуальними елементами, що викликає довіру до сервісу та сприяє комфортному сприйняттю інформації.
- Зрозуміла форма бронювання: процес ініціації замовлення реалізований через просту та інтуїтивно зрозумілу форму. Це дозволяє клієнту швидко та без ускладнень надіслати базові дані для оформлення заявки на оренду.

Недоліки:

- Відсутність особистого кабінету: платформа не передбачає реєстрації користувачів. Це унеможливорює збереження історії попередніх бронювань, персональних даних або цифрових копій документів. Як наслідок, клієнт змушений щоразу вводити всю інформацію заново, а компанія втрачає

інструмент для реалізації персоналізованих програм лояльності.

- Відсутність інтерактивного календаря доступності: у системі не реалізовано механізм відображення зайнятості конкретного транспортного засобу в режимі реального часу. Клієнт не може самостійно переглянути недоступні дати, тому фактична перевірка доступності авто відбувається лише на етапі обробки заявки.
- Відсутність системи фільтрації: користувач позбавлений можливості звузити вибірку автомобілів за важливими технічними параметрами (тип коробки передач, вид пального, кількість посадкових місць). Це суттєво уповільнює процес підбору оптимального варіанту.
- Повна відсутність пошуку: на платформі не реалізовано інструментів текстового пошуку. Це змушує клієнта здійснювати виключно ручний перебір каталогу. Це є критичним недоліком для сучасних інформаційних систем, оскільки значно знижує ефективність взаємодії користувача з продуктом.

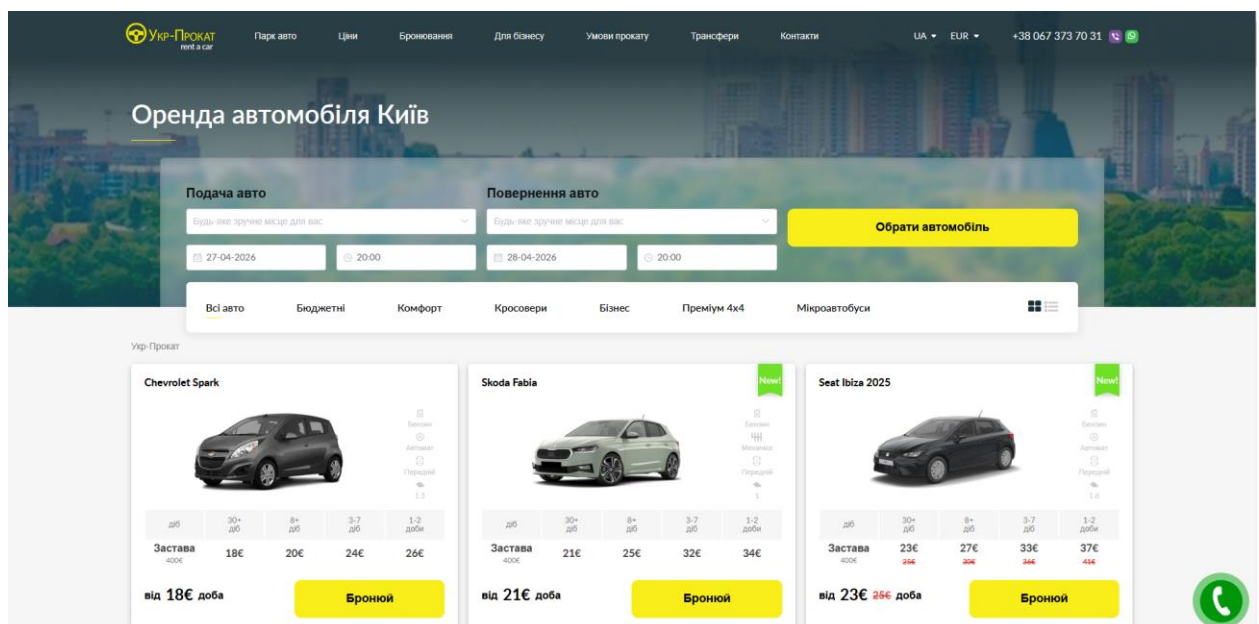


Рис. 1.1 Сторінка каталогу сервісу Укр-Прокат

Бронювання автомобіля

« Назад

Місце подачі авто

Будь-яке зручне місце для вас

27-04-2026 20:00

Міс

Будь-яке зручне місце для вас

2028 Квітень

Пн Вт Ср Чт Пт Сб Нд

27 28 29 30 31 1 2

3 4 5 6 7 8 9

10 11 12 13 14 15 16

17 18 19 20 21 22 23

24 25 26 27 28 29 30

1 2 3 4 5 6 7

GPS 5€ /доба

Дитяче крісло 5€ /доба

Wi-Fi 4€ /доба

Стандартне страхування (Застава 400€) Без доплати 0€

Повне страхування (Застава 100€) +13€ / доба 13€

Особисті дані

Chevrolet Spark

Всього 56 €

Застава: 400€

У вартість оренди включено

- Автомобіль: 26€ (к-сть дб: 1)
- Подача в неробочий час: 15 €
- Повернення в неробочий час: 15 €
- Графік роботи: Пн - Пт 09:00 - 18:00

Промокод

Активувати

Рис. 1.2 Сторінка бронювання сервісу Укр-Прокат

1.3.2 BLS

BLS – ще один вагомий гравець на українському ринку автопрокату, який надає послуги оренди транспортних засобів для приватних та корпоративних клієнтів. Цифрова платформа цієї компанії відрізняється глибшим рівнем інтеграції клієнтських сервісів та орієнтована на оптимізацію процесу підбору автомобіля через функціональний веб-інтерфейс [4].

Основний функціонал веб-платформи зосереджений навколо розширеного каталогу транспортних засобів, який пропонує деталізований поділ на класи, включаючи такі специфічні категорії, як електромобілі та автомобілі VIP-сегменту. Важливою складовою системи є інтегрований динамічний калькулятор, що автоматично розраховує підсумкову вартість оренди залежно від обраних дат, загальної тривалості прокату та бажаного рівня страхового покриття. Платформа також надає користувачам гнучкі інструменти управління локаціями, дозволяючи оформлювати адресну доставку автомобіля або його подачу до ключових інфраструктурних вузлів (аеропорти, вокзали).

Переваги:

- Пошук за марками та моделями авто: на відміну від попередньо розглянутого рішення, платформа BLS має інтегровану систему текстового пошуку. Це

дозволяє клієнту не витрачати час на перегляд усього каталогу, а одразу використовувати цільові запити для знаходження конкретної моделі або марки транспортного засобу, що суттєво підвищує ергономічність системи.

- Наявність системи фільтрації: інтерфейс забезпечує користувача розширеним набором фільтрів. Клієнт має змогу гнучко налаштовувати параметри вибірки (наприклад, тип трансмісії, вид пального, клас авто, ціновий діапазон), відсіюючи нерелевантні пропозиції та залишаючи лише ті автомобілі, які відповідають його вимогам.
- Функціональний особистий кабінет: платформа передбачає можливість повноцінної реєстрації та створення профілю користувача. Це забезпечує збереження історії бронювань, управління персональними даними та зручний моніторинг статусів поточних замовлень.
- Завчасне завантаження документів: сервіс пропонує часткову цифровізацію процесу верифікації клієнта. Через особистий кабінет користувач може заздалегідь завантажити цифрові копії необхідних документів (паспорт, водійське посвідчення). Це значно прискорює процес оформлення оренди в офлайн, оскільки менеджерам залишається лише звірити оригінали при видачі авто.

Недоліки:

- Відсутність інтерактивного календаря: як і в попередньому сервісі, залишається проблема неможливості перегляду зайнятих дат конкретного автомобіля в режимі реального часу, що змушує клієнта чекати на підтвердження доступності від менеджера.

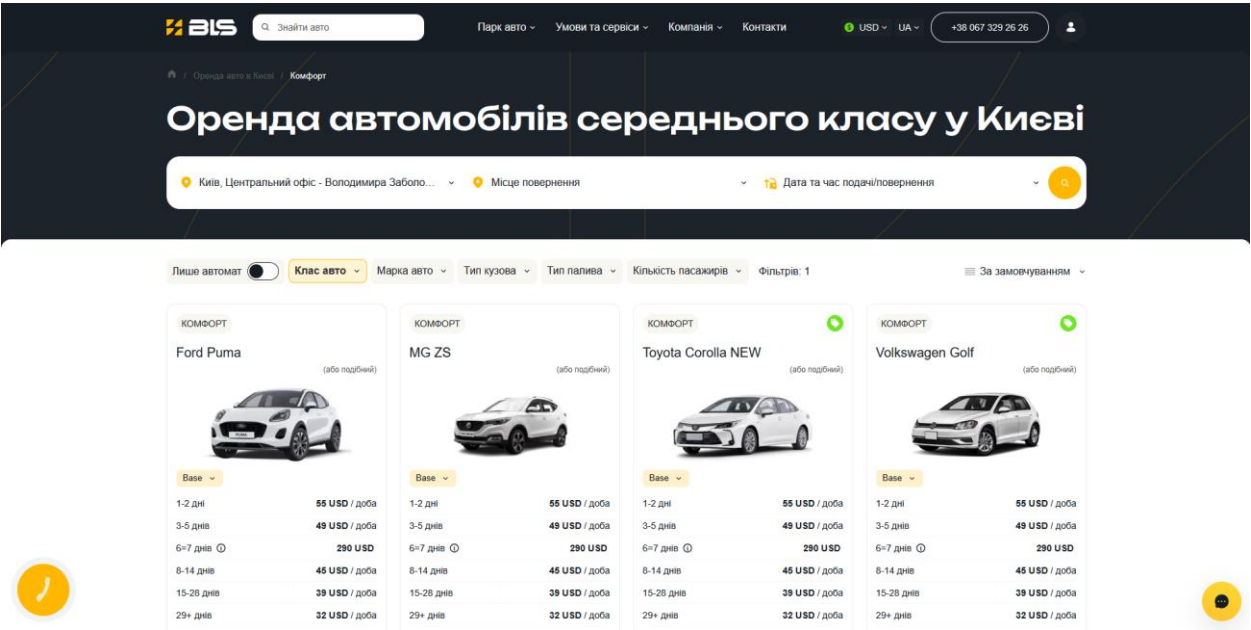


Рис 1.3 Сторінка каталогу сервісу BLS

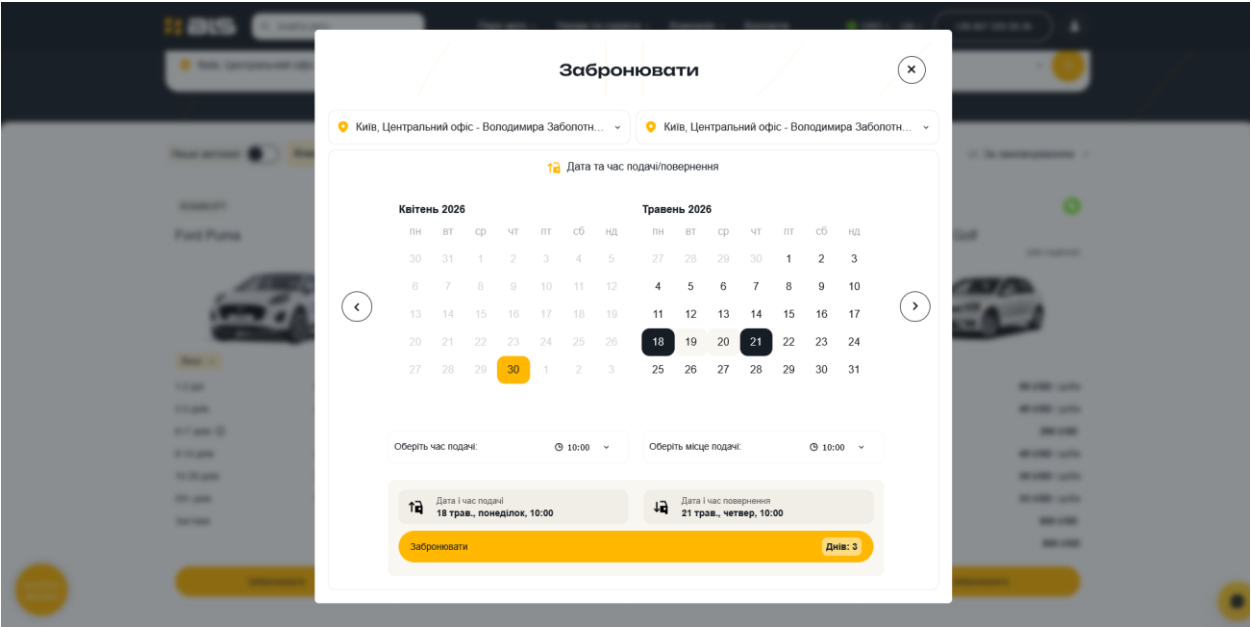


Рис. 1.4 Процес вибору дат для бронювання в сервісі BLS

Доповнення

Додатковий водій
Керуйте автомобілем спільно і будете вписані в тому, що всі водії застраховані 4 USD / доба

Крісло для дитини
Безпечна поїздка для дитини. Авторісло відповідно до віку. 4 USD / доба

Відомості про водія
Будьласка, введіть ім'я, яке вказане у вашому паспорті або міжнародному водійському посвідченні

Ім'я *

Прізвище *

Телефон *

Ел. пошта *

Додати документи +

Натисніть, щоб завантажити файл
або перекладіть у ліву колонку

Відомості про авто

5 місць

Передній привід

Кондиціонер

Бензин 1.0 л.

Автомат

6 л./100км

Місце прокату та дата-час

18 трав., понеділок, 10:00
Київ, Центральний офіс - Володимира Заболотного, 1А

21 трав., четвер, 10:00
Київ, Центральний офіс - Володимира Заболотного, 1А

Вартість оренди авто 147 USD

Загальна вартість 147 USD

* Під час отримання авто з вас також буде стягнуто заставу в розмірі 500 USD. Після повернення авто сума буде повернута.

Забронювати

Рис. 1.5 Сторінка бронювання сервісу BLS

1.3.3 Megarent

Megarent – ще один представник українського ринку оренди транспортних засобів. Веб-ресурс компанії функціонує переважно як цифрова вітрина автопарку та інструмент первинного збору заявок від клієнтів, орієнтуючись на базове забезпечення процесу бронювання без глибокої інтеграції додаткових цифрових сервісів [5].

Переваги:

- Форма швидкого підбору: на головній сторінці реалізована зручна форма стартового підбору автомобіля, яка дозволяє користувачу оперативно задати базові параметри для швидкого переходу до оформлення заявки.
- Класифікація автопарку: як і в системі Укр-Прокат, наявне чітке розділення транспортних засобів за класами, що полегшує базову навігацію каталогом.

Недоліки:

- Застарілий користувацький інтерфейс: на відміну від сучасніших аналогів, візуальне оформлення та ергономіка платформи Megarent не відповідають актуальним стандартам веб-розробки. Дизайн виглядає застарілим, що може

ускладнювати інтуїтивну взаємодію з сайтом, погіршувати загальний клієнтський досвід та негативно впливати на рівень довіри до сервісу.

- Проблеми з актуальністю даних та адмініструванням: на платформі періодично зустрічається неактуальна інформація (наприклад, щодо цін, характеристик або наявності певних моделей). Це може бути індикатором недосконалого функціоналу панелі адміністратора. Очевидно, система не має автоматизованої синхронізації між фронтом та базою даних підприємства, що змушує менеджерів оновлювати інформацію вручну. Як наслідок – високий ризик людської помилки та дезінформація кінцевих споживачів.
- Відсутність ключового функціоналу: як і в попередніх випадках, у системі відсутні особистий кабінет клієнта, інструменти текстового пошуку та система технічних фільтрів.

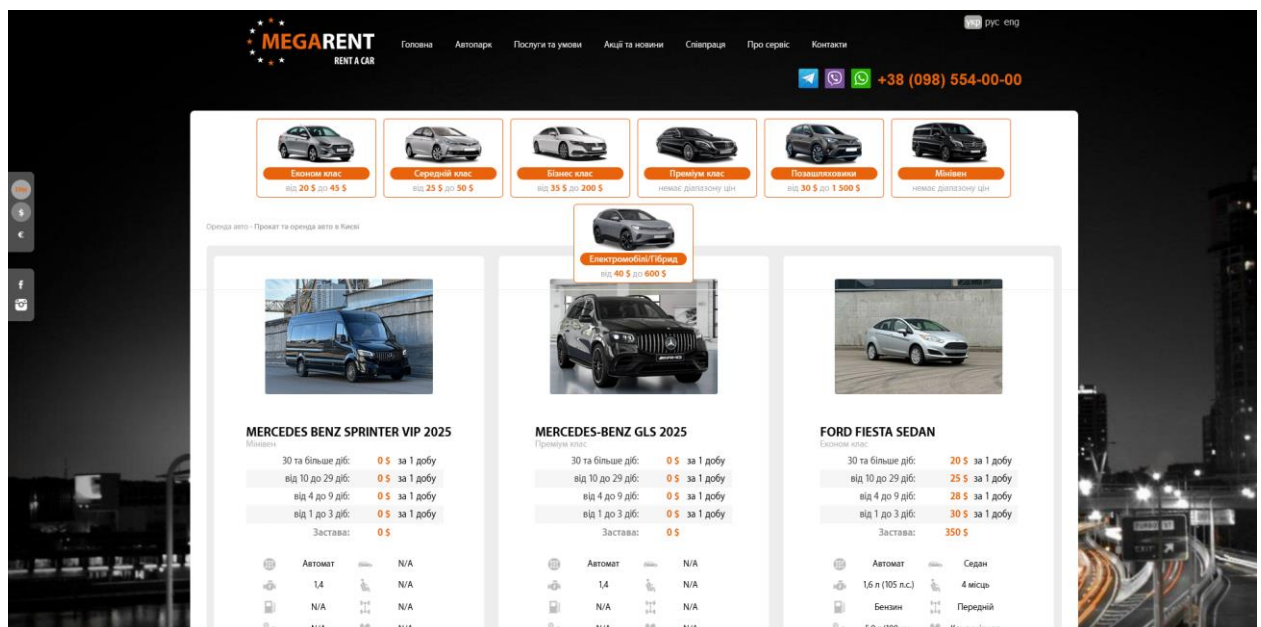


Рис 1.6 Сторінка каталогу сервісу Megarent

Online бронювання автомобіля Ford Fiesta sedan

30 та більше днів	20 \$	за 1 добу	Мінімальний вік - 21 рік
від 10 до 29 днів	25 \$	за 1 добу	Мінімальний стаж - 2 роки
від 4 до 9 днів	28 \$	за 1 добу	Мінімальний термін оренди - 1 доба
від 1 до 3 днів	30 \$	за 1 добу	
Застава:	350 \$		

- Тариф включає**
 - Страхування КАСКО;
 - Страхування цивільної відповідальності;
 - Прокід 350 км/добу;
 - Цілодобова парковка.
- Період оренди**
 - Коли забираєте? 30.04.2026
 - Коли віддаєте? 01.05.2026
 - Де забираєте? Офіс
 - Де віддаєте? Офіс
- Додаткові опції**
 - ☐ Дитяче автокрісло (0+)
 - ☐ Другий водій (БЕЗКОШТОВНО)
- Вартість за 1 днів**
 - Попередня вартість оренди: 30 \$
 - Заставна сума: 350 \$
- Персональні дані**
 - Ваше ім'я та Прізвище
 - + Телефон
 - Ваш e-mail
 - Ваш коментар

Рис 1.7 Сторінка бронювання сервісу Megarent

1.3.4 7cars

7cars – ще одна відома українська компанія, що надає послуги прокату автомобілів. Її цифрова платформа є прикладом збалансованого підходу до надання послуг онлайн, поєднуючи необхідний базовий функціонал для підбору авто із сучасними вимогами до клієнтського сервісу та збереженням даних користувачів [6].

Платформа пропонує розширені можливості оформлення замовлень: окрім класичної оренди авто без водія, через систему можна замовляти послуги прокату з професійним водієм, ініціювати процес довгострокової оренди та бронювати пасажирські трансфери. Процес взаємодії з системою здійснюється через покрокові онлайн-форми, де клієнт обирає необхідні дати, локації та супутні опції.

Переваги:

- **Якісний UI-дизайн:** як і у випадку з Укр-Прокат, платформа має сучасне візуальне оформлення та інтуїтивно зрозумілий інтерфейс, що позитивно впливає на загальний користувацький досвід.
- **Наявність особистого кабінету:** подібно до системи BLS, сервіс надає можливість реєстрації клієнта, що дозволяє зберігати історію взаємодій та

персональні дані.

- Класифікація автопарку та фільтрація: система пропонує зручний поділ автомобілів на класи в поєднанні з набором технічних фільтрів, що значно полегшує процес звуження вибірки до релевантних варіантів.

Недоліки:

- Відсутність текстового пошуку: як і в більшості попередніх аналогів, у системі не реалізовано рядок вільного пошуку (зокрема семантичного), що обмежує навігаційні можливості клієнта.
- Відсутність інтерактивного календаря доступності: система не відображає зайняті дати обраного транспортного засобу в режимі реального часу, залишаючи етап остаточного підтвердження за менеджером.

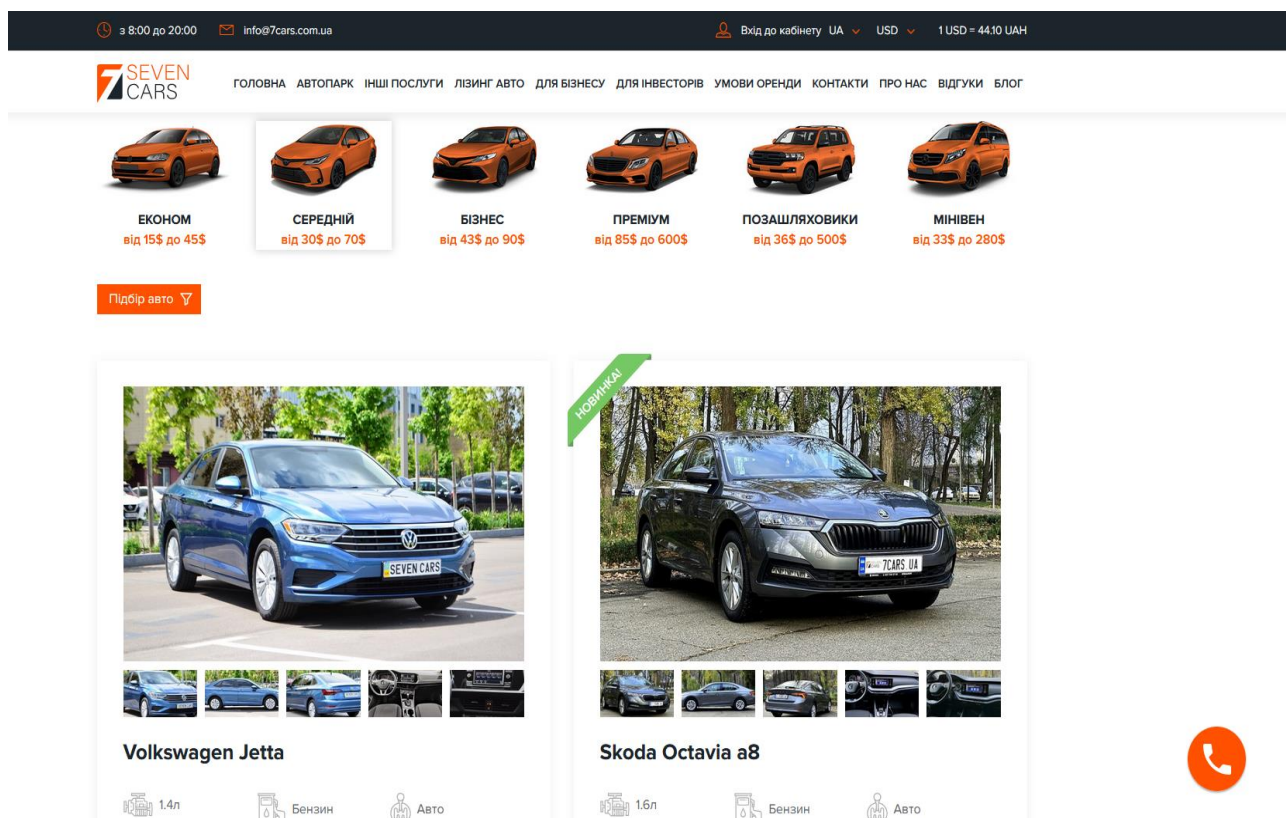


Рис 1.8 Сторінка каталогу сервісу 7cars

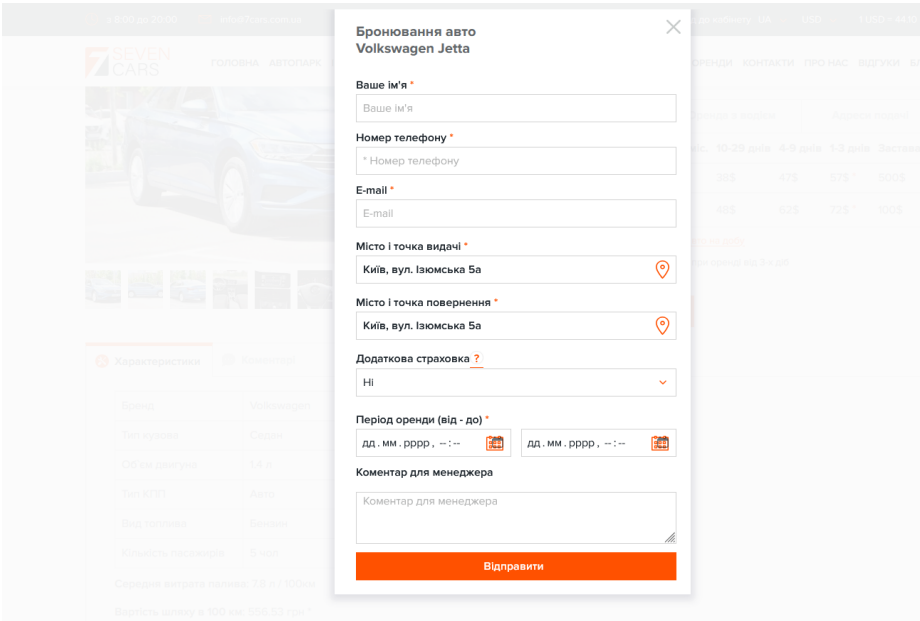


Рис 1.9 Форма бронювання сервісу 7cars

1.3.5 Зведена таблиця результатів аналізу

В результаті огляду та аналізу існуючих рішень створено зведену таблицю з основними критеріями порівняння.

Таблиця 1.1

Зведені результати аналізу характеристик застосунків для оренди автомобілів

	Укр-Прокат	BLS	Megarent	7cars
Перегляд каталогу та онлайн бронювання	+	+	+	+
Особистий кабінет з історією бронювання	—	+	—	+
Попередня віддалена верифікація документів	—	+	—	—
Інтерактивний календар доступності	—	—	—	—
Фільтри авто за тех. характеристиками	—	+	—	+
Пошук по маркам і моделям	—	+	—	—
Семантичний AI-пошук	—	—	—	—
Сучасний UI-дизайн	+	+	—	+
Технологічний стек та архітектура	Застарілі монолітні рішення (CMS, PHP)			

1.4 Формування вимог до розроблюваної інформаційної системи

На основі проведеного аналізу предметної області, специфіки бізнес-процесів подовгової оренди транспортних засобів та детального дослідження існуючих на ринку аналогів, наступним логічним етапом проєктування є визначення чітких вимог до майбутньої інформаційної системи, що носитиме назву “Wheelix”. Формування специфікації вимог є фундаментальним кроком життєвого циклу розробки програмного забезпечення, оскільки саме ці критерії визначатимуть архітектурні рішення, логіку взаємодії компонентів та успішність реалізації проєкту в цілому.

Для забезпечення системного підходу до проєктування, усі вимоги до розроблюваного веб-застосунку розділено на дві основні категорії: функціональні та нефункціональні.

Функціональні вимоги описують безпосередню поведінку системи, її бізнес-логіку та можливості, які вона надає користувачам згідно з їхніми правами доступу (клієнтам та менеджерам). Вони охоплюють процеси реєстрації та верифікації, пошуку і бронювання авто, управління каталогом автопарку тощо. Нефункціональні вимоги, в свою чергу, встановлюють стандарти якості програмного продукту, регламентуючи параметри криптографічного захисту даних, продуктивності, масштабованості системи та обмеження щодо обраного технологічного стеку [7].

1.4.1 Функціональні вимоги

Підсистема авторизації та управління профілем

- система повинна забезпечувати реєстрацію нових користувачів;
- система повинна дозволяти користувачам авторизуватися за допомогою електронної пошти та пароля;
- клієнт повинен мати можливість редагувати особисті дані (ПІБ, телефон) та завантажувати електронні копії документів (водійське посвідчення,

паспорт) для верифікації.

Підсистема пошуку та оренди авто

- користувач повинен мати можливість переглядати каталог доступних автомобілів із зазначенням їх характеристик (клас, тип кузова, трансмісія, ціна за добу);
- система повинна забезпечувати фільтрацію авто за заданими характеристиками;
- система повинна надавати користувачу можливість здійснювати пошук доступних автомобілів шляхом введення неструктурованих текстових запитів природною мовою;
- клієнт повинен мати можливість створювати заявку на бронювання авто та відстежувати її статус в особистому кабінеті;
- система повинна автоматично розраховувати загальну вартість оренди на основі обраних дат.

Підсистема адміністрування

- менеджер повинен мати можливість додавати, редагувати або видаляти інформацію про автомобілі;
- система повинна надавати менеджеру інтерфейс для перегляду всіх вхідних заявок на бронювання з можливістю їх затвердження або відхилення.

1.4.2 Нефункціональні вимоги

Вимоги до безпеки та захисту даних

- доступ до адміністративної панелі та особистого кабінету повинен контролюватися на основі рольової моделі (RBAC);
- для підтримки сесій користувачів повинен використовуватися механізм токенів доступу (JWT), що виключає необхідність зберігання стану на сервері;
- паролі користувачів повинні зберігатися в базі даних виключно у вигляді

хеш-сум, згенерованих за алгоритмом bcrypt;

-фотографії документів користувачів повинні зберігатися в хмарному об'єктному сховищі і бути захищені механізмом тимчасових підписаних посилань;

-система повинна здійснювати валідацію всіх вхідних даних на клієнтському та серверному рівнях для запобігання атакам типу SQL-ін'єкцій та XSS.

Вимоги до продуктивності та надійності

-час відгуку серверної частини (REST API) на стандартні запити (перегляд списку авто, перегляд профілю) не повинен перевищувати 2 секунди.

-система повинна коректно обробляти 50 запитів на секунду без критичної втрати продуктивності.

1.5 Технічні засоби реалізації системи

Успішна реалізація інформаційної системи безпосередньо залежить від правильно підбраного комплексу програмних засобів та технологій. Для забезпечення надійної роботи, високої швидкодії та зручності подальшої підтримки було сформовано сучасний технологічний стек, який охоплює серверну частину, клієнтський інтерфейс, інтеграцію сторонніх сервісів та засоби зберігання даних. У цьому розділі наведено обґрунтування вибору мов програмування, фреймворків та інфраструктурних рішень, що використовувалися під час створення програмного продукту.

1.5.1 Середовища розробки

IntelliJ IDEA. Для написання коду серверної частини застосунку було обрано інтегроване середовище розробки IntelliJ IDEA від компанії JetBrains. IntelliJ IDEA забезпечує інтелектуальне автодоповнення, а також підтримує засоби статичного аналізу та автоматизованого рефакторингу, що критично важливо для підтримки

чистоти кодової бази. Розширена підтримка екосистеми Spring Boot, зручні інструменти відлагодження та безпосередня інтеграція з системами керування базами даних дозволяють мінімізувати час на рутинне налаштування й зосередитися на реалізації бізнес-логіки [8].

Visual Studio Code. У якості основного інструменту для розробки клієнтської частини системи обрано Visual Studio Code від Microsoft. На відміну від важких IDE, цей редактор пропонує оптимальний баланс між продуктивністю та функціональністю під час роботи зі скриптовими мовами. Його використання обґрунтоване високою швидкістю роботи, низьким споживанням системних ресурсів та наявністю розлогої екосистеми розширень для екосистеми TypeScript і React. Це забезпечує гнучкість налаштування робочого простору, швидку навігацію по компонентах та зручну інтеграцію з терміналом і системами контролю версій [9].

1.5.2 Мови програмування

Java. Вибір Java як основної мови серверної частини зумовлений її високою надійністю, строгою типізацією та платформонезалежністю завдяки віртуальній машині JVM. Для інформаційних систем, що обробляють значні обсяги даних та критичні транзакції, Java є промисловим стандартом. Вона надає розвинені вбудовані механізми багатопоточності, зрілу архітектуру безпеки та гарантує стабільність системи при масштабуванні й високих навантаженнях, що мінімізує ризики відмови сервісу [10].

TypeScript. Для реалізації клієнтського інтерфейсу замість традиційного JavaScript було обрано мову TypeScript. Це рішення обґрунтоване потребою мінімізації помилок на етапі компіляції, що суттєво підвищує стабільність фронтенду. Статична типізація спрощує рефакторинг великих кодових баз, покращує роботу автодоповнення в IDE та забезпечує чітке дотримання контрактів даних, які передаються між клієнтом та сервером через API [11].

1.5.3 Фреймворки та бібліотеки

Spring Boot. Реалізація архітектури серверної частини спирається на фреймворк Spring Boot. Його вибір дозволяє значно прискорити процес розробки та відмовитися від громіздкої ручної конфігурації завдяки принципу автоконфігурації (convention over configuration). Використання вбудованого сервера (Tomcat) та спрощене управління залежностями за допомогою стартових пакетів забезпечують швидке розгортання застосунку. Фреймворк надає готові інфраструктурні рішення для безпеки, автентифікації (JWT) та доступу до даних, звільняючи розробника від написання шаблонного коду [12].

React. React – найпопулярніша на даний момент бібліотека для створення сучасних користувацьких інтерфейсів. Вибір цієї технології обґрунтований її компонентним підходом, який дозволяє розбивати UI на ізольовані модулі для повторного використання, спрощуючи підтримку коду. Технологія Virtual DOM підвищує швидкість рендерингу завдяки оптимізації оновлень веб-сторінок, а односпрямований потік даних призводить до передбачуваної поведінки у вашому інтерфейсі [13].

Next.js. Фреймворк Next.js дозволяє значно розширити функціонал React і пропонує ряд готових технічних рішень і оптимізацій для розробників. Його використання обумовлене необхідністю оптимізації початкового завантаження сторінок та покращення показників доступності системи за допомогою рендерингу на стороні сервера (SSR). Завдяки маршрутизації на основі файлової системи, автоматичній оптимізації зображень та розділенню коду забезпечується висока продуктивність клієнтської частини, при цьому усувається потреба у складному ручному налаштуванні інструментів збирання [14].

1.5.4 Зберігання даних

PostgreSQL. Для зберігання всіх реляційних даних та керування ними використовується PostgreSQL. Вибір цієї системи управління БД обґрунтований її повною відповідністю вимогам ACID, що гарантує надійність, несуперечливість та цілісність інформації, в тому числі даних клієнтів. PostgreSQL демонструє високу

ефективність при виконанні складних аналітичних запитів, підтримує розширені типи даних та забезпечує стабільну роботу в умовах конкурентного доступу великої кількості користувачів [15].

Amazon S3. Зберігання великих обсягів неструктурованих даних (таких як зображення об'єктів системи та завантажені користувачами документи) винесено у хмарне об'єктне сховище Amazon S3. Локальне збереження файлів на сервері застосунків є неефективним та обмежує горизонтальне масштабування. Вибір Amazon S3 обґрунтовано гарантією безперебійного доступу, високим рівнем довговічності даних, вбудованим шифруванням та гнучким керуванням правами доступу. Інтеграція через офіційний AWS SDK для Java дозволяє повністю ізолювати логіку роботи зі статичним контентом від обчислювальних потужностей сервера, оптимізуючи витрати [16].

1.5.5 Засоби контролю версій та сторонні сервіси

Git. Управління кодовою базою та відстеження змін здійснюється за допомогою розподіленої системи контролю версій Git. Її застосування є обов'язковим стандартом для забезпечення цілісності сирцевого коду, організації паралельної розробки через систему гілок (branches) та фіксації станів системи за допомогою криптографічних хешів. Наявність індексу (staging area) забезпечує гнучкість формування комітів та точний контроль над версіями програмного продукту [17].

Gemini API. Для інтеграції інтелектуальних функцій, зокрема реалізації семантичного пошуку природною мовою, було обрано програмний інтерфейс Gemini API від компанії Google. Використання хмарного API дозволяє залучити передові мультимодальні архітектури з мінімальними затримками без необхідності розгортання та підтримки власних ресурсомістких великих мовних моделей. Вибір Gemini API обґрунтований високою точністю контекстуального аналізу тексту, масштабованістю хмарної інфраструктури Google та наявністю сучасних клієнтських бібліотек, що дозволяє з легкістю інтегрувати функції штучного інтелекту в поточну архітектуру системи [18].

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Діаграма варіантів використання

Один із перших етапів проєктування будь-якої системи – це визначення ролей користувачів. В системі оренди, що розробляється, передбачено два типи користувачів – клієнт та менеджер. Відповідно, для кожної з ролей передбачено визначену кількість операцій, яку користувач може здійснювати в системі. Для наочної демонстрації цих операцій, на етапі проєктування створено діаграми використання. На рисунку 2.1 зображено діаграму використання для ролі клієнта, на рисунку 2.2 – для менеджера.

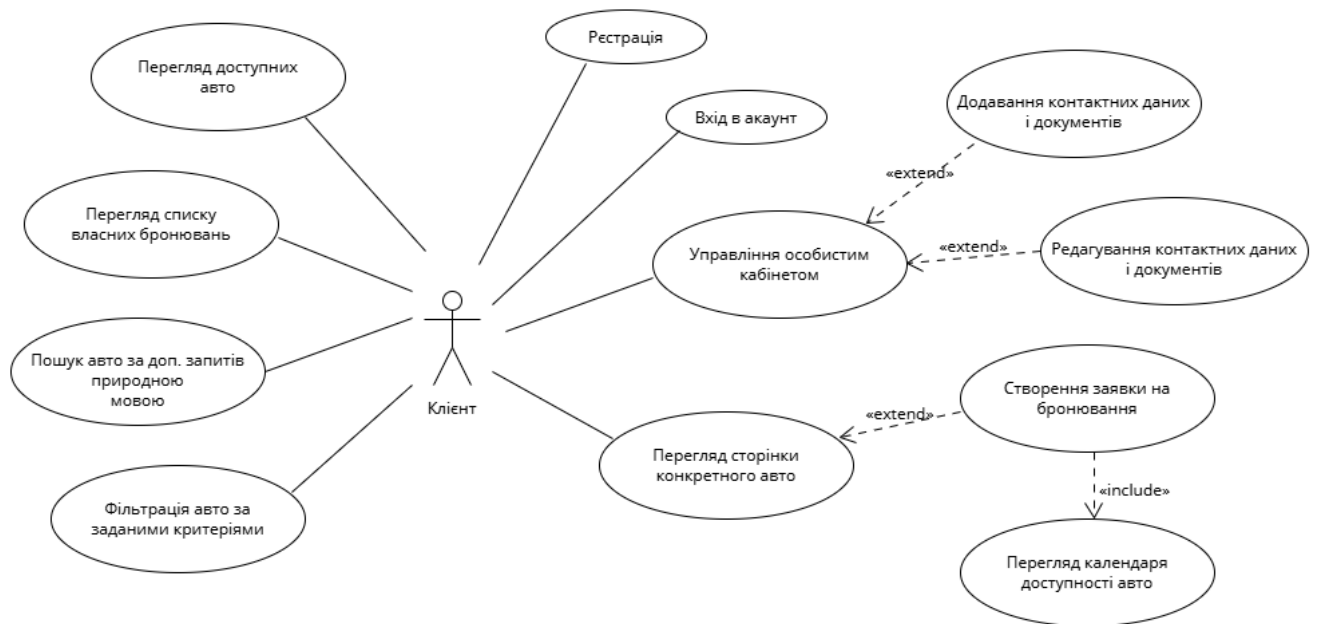


Рис 2.1 Діаграма використання для ролі “клієнт”

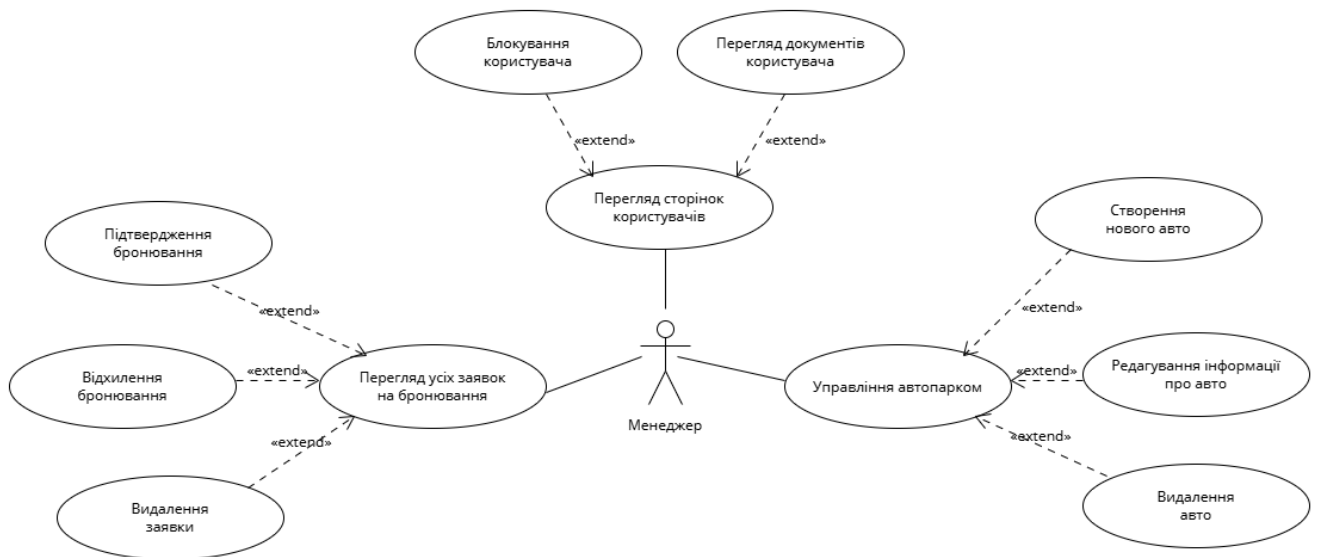


Рис 2.2 Діаграма використання для ролі “менеджер”

Основні прецеденти для ролі “клієнт”: реєстрація, вхід, управління особистим кабінетом, перегляд доступних авто, перегляд конкретного авто, створення заявки на бронювання, перегляд списку бронювань, фільтрація авто та пошук авто за допомогою запитів природною мовою.

Список функцій, доступних менеджеру, розширюють наступні прецеденти: управління автопарком, перегляд усіх заявок на бронювання, підтвердження, відхилення та видалення заявок, перегляд сторінок користувачів, блокування користувачів.

Таким чином, система поєднує зручний функціонал для користувачів і інструмент управління та обліку для менеджерів. З технічної точки зору, розділення прав доступу відбувається за допомогою авторизації через JWT токени.

2.2 Архітектура системи

Як і більшість сучасних веб-застосунків, “Wheelix” використовує модель клієнт-серверної архітектури. Клієнтську частину, тобто frontend, створено за допомогою бібліотеки React та фреймворку Next. Серверна частина, тобто backend, використовує фреймворк Spring Boot. Взаємодія двох частин відбувається за REST

архітектурою – шляхом обміну HTTP запитами та відповідями, які містять інформацію у вигляді JSON-об’єктів [19]. Такий підхід дозволяє чітко розділити інтерфейс, з яким взаємодіє кінцевий користувач, та бізнес-логіку системи. На етапі розробки це дозволяє розробнику(-кам) працювати над однією частиною системи незалежно від іншої, що додає гнучкості процесу. На етапі функціонування системи – дозволяє ізолювати помилки, а збої з меншою вірогідністю виведуть з ладу всю систему. За потреби вдосконалення застосунку, сучасний стек технологій і архітектура не створять додаткових перешкод при впровадженні нового функціоналу.

Схему взаємодії окремих компонентів системи зображено на рисунку 2.3.

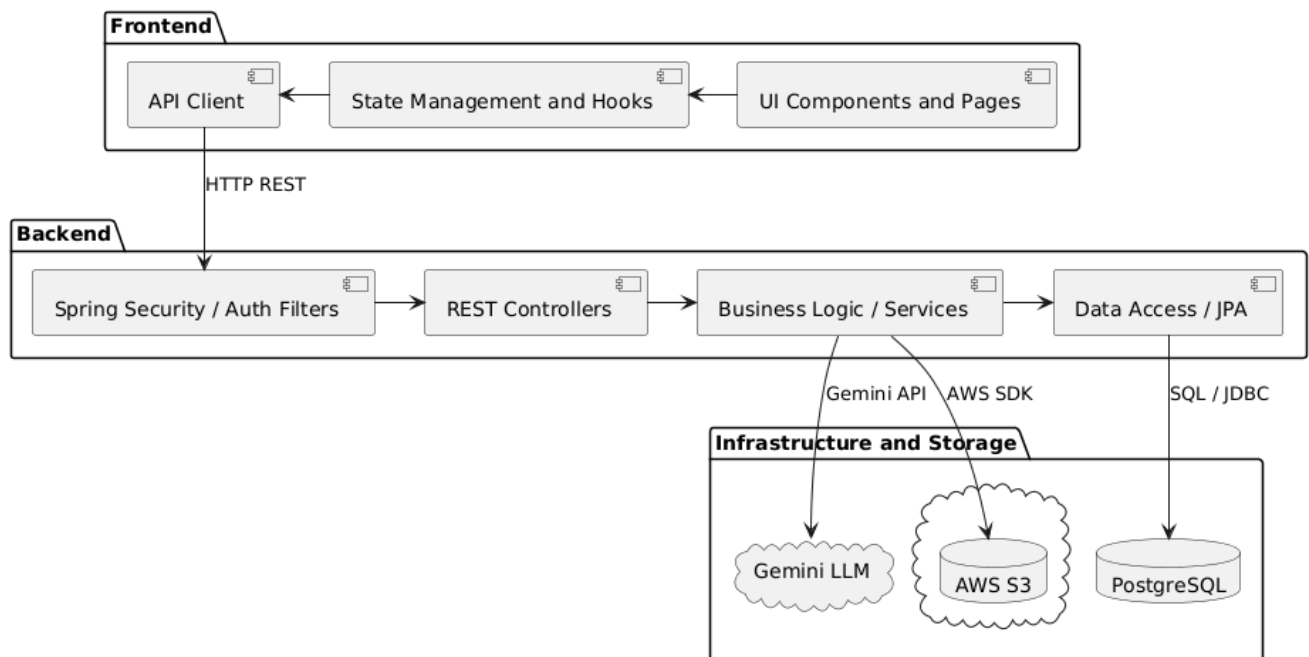


Рис 2.3 Діаграма компонентів архітектури системи

Окрім frontend та backend частин, на діаграмі зображено шар інфраструктури та зберігання даних. Основною базою даних у системі виступає реляційна PostgreSQL. Дані в ній зберігаються у вигляді таблиць (сутностей) та їх полів (атрибутів). Додатково, для зберігання фотографій автомобілів та документів користувачів, використовується хмарне об’єктне сховище AWS S3. Його використання надає ряд переваг в порівнянні з локальним зберіганням файлів на

власному сервері: додаткове шифрування даних, безперебійний доступ, можливість налаштовувати права доступу. Файли фотографій авто доступні публічно для будь-якого користувача, файли документів – захищені додатковим механізмом тимчасових підписаних посилань. Перед доступом до документів система перевіряє, чи користувач має права на доступ до них, після чого видає спеціальне посилання для доступу, що діє впродовж обмеженого часу. Така реалізація зводить ризик несанкціонованого доступу до мінімуму.

Також шар інфраструктури на діаграмі зображує інтеграцію з великою мовною моделлю Gemini. Вона використовується у системі для можливості семантичного аналізу запитів користувачів при пошуку авто. Користувач має можливість ввести у пошук абстрактний запит (наприклад, “авто для пересічної місцевості на дизелі”), а мовна модель перетворить його у чітко структурований JSON-об’єкт з параметрами для фільтрів, що надалі будуть застосовані для фільтрації списку доступних авто. Використання саме Gemini обумовлене в першу чергу зручним API і легкістю інтеграції, хоча для подібних задач можна використати й інші моделі штучного інтелекту. Для більшості сучасних моделей задача семантичного розбору й розпізнавання іменованих сутностей (наприклад, марки і моделі авто) є досить неважкою.

2.3 Архітектура серверної частини системи

Серверна частина застосунку створена за допомогою фреймворку Spring Boot і використовує типову для нього трирівневу модель архітектури (three-layer architecture). В такій парадигмі програмний код поділяється на шари: презентаційний шар, шар бізнес-логіки та шар доступу до даних. В Spring Boot такі класи носять відповідні назви: контролер (controller), сервіс (service) та репозиторій (repository). Контролери відповідають за приймання запитів від фронтенду і формування відповідей, сервіси містять логіку роботи застосунку і здійснюють операції з даними, репозиторії – відповідають за отримання інформації

з бази даних. Важливо також те, що в кодї ці класи мають використовуватись в ієрархічному порядку, тобто контроллер не може отримати доступ до репозиторію напряду, а лише через відповідний сервіс. Це гарантує структурованість коду і підвищує безпеку застосунку через ізоляцію даних.

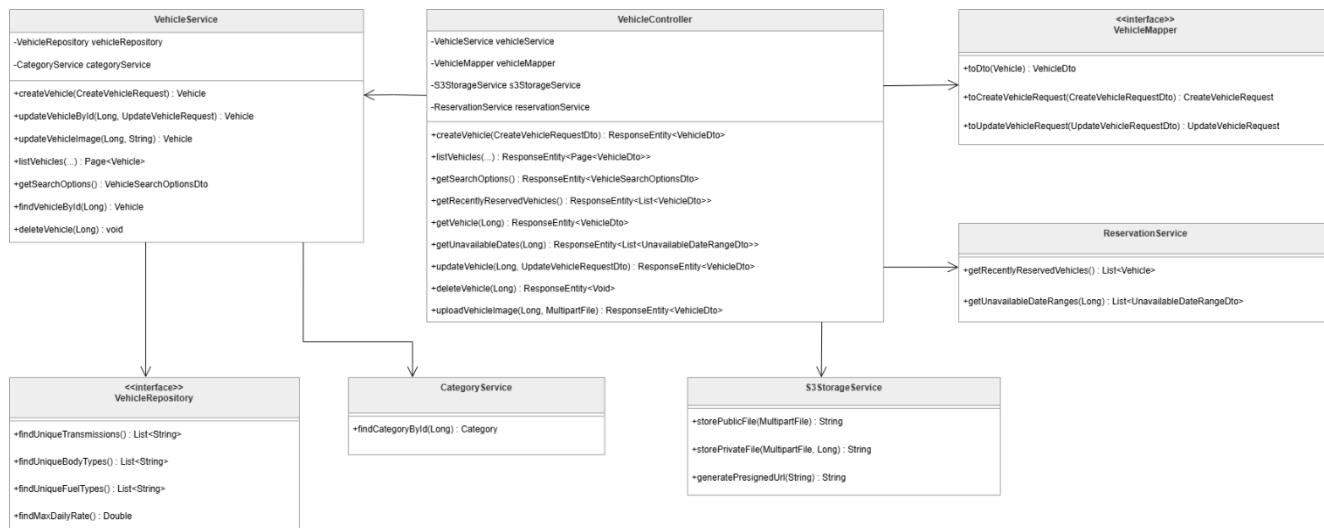


Рис 2.4 Діаграма класів роботи з автомобілями

На рисунку 2.4 зображено діаграму класів, що відповідають за логіку роботи з автомобілями. Всього в системі 4 основні сутності – автомобілі (Vehicles), категорії (Categories), користувачі (Users), бронювання (Reservations). Візуально вони продемонстровані на рисунку 2.5 на діаграмі класів сутностей.

Крім основних класів сутностей, на діаграмі також зображені 2 типи перелічення – UserRole та ReservationStatus. Перший відповідає за роль користувача в системі (клієнт, менеджер), другий – за статус бронювання (в обробці, підтверджено, скасовано).

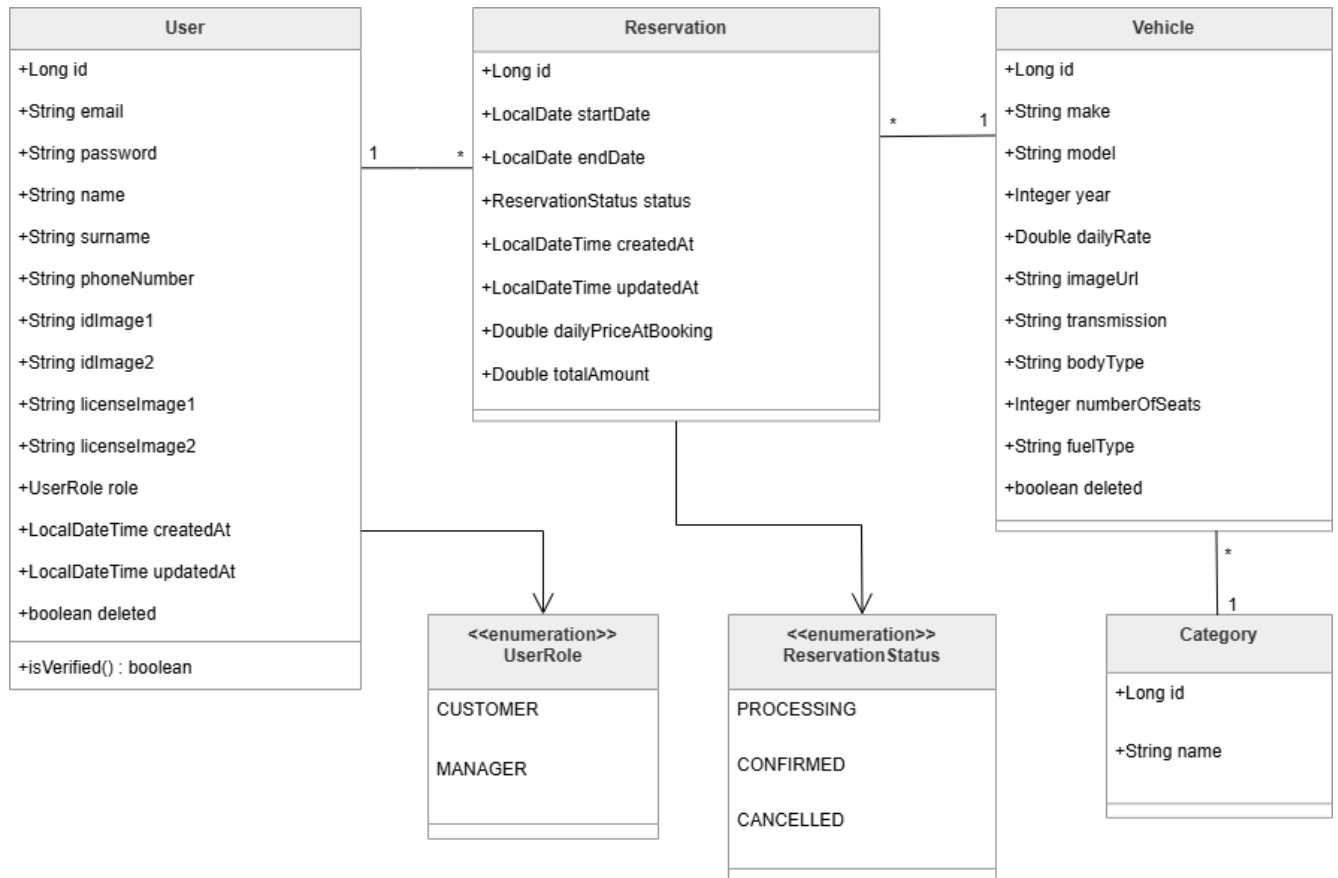


Рис 2.5 Діаграма класів сутностей

2.4 Архітектура клієнтської частини системи

Клієнтська частина реалізована з використанням TypeScript, React і Next. Фреймворк Next пропонує зручний механізм маршрутизації, що базується на структуруванні директорій і неймінгу файлів. Завдяки цьому, вже на етапі проектування можливо передбачити майбутню структуру сайту. В результаті маємо три мапи сайту (залежно від автентифікації та авторизації користувача) з усіма доступними кінцевому користувачу сторінками (див. рис. 2.6-2.8).

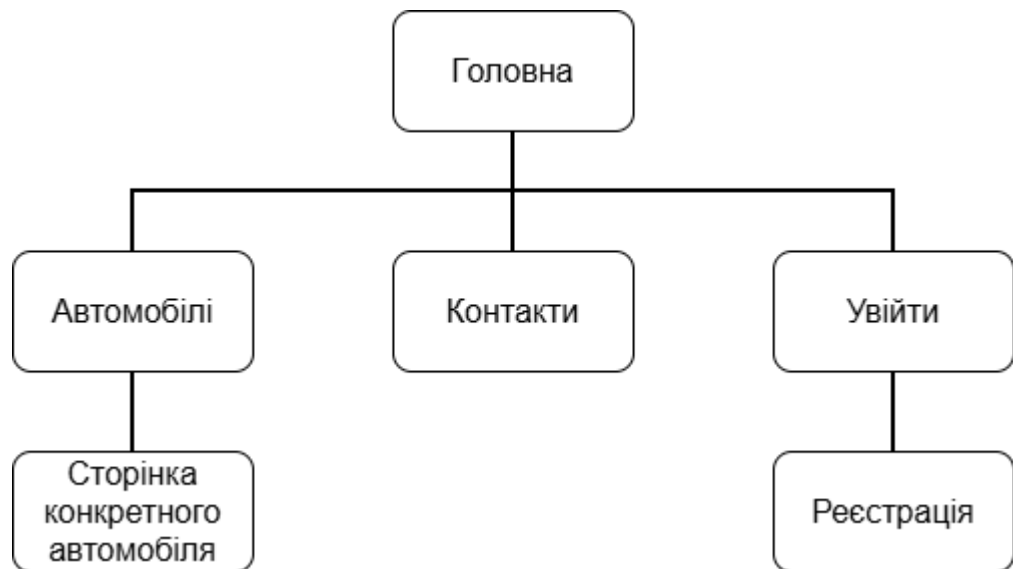


Рис 2.6 Мапа сайту для неавтентифікованого користувача

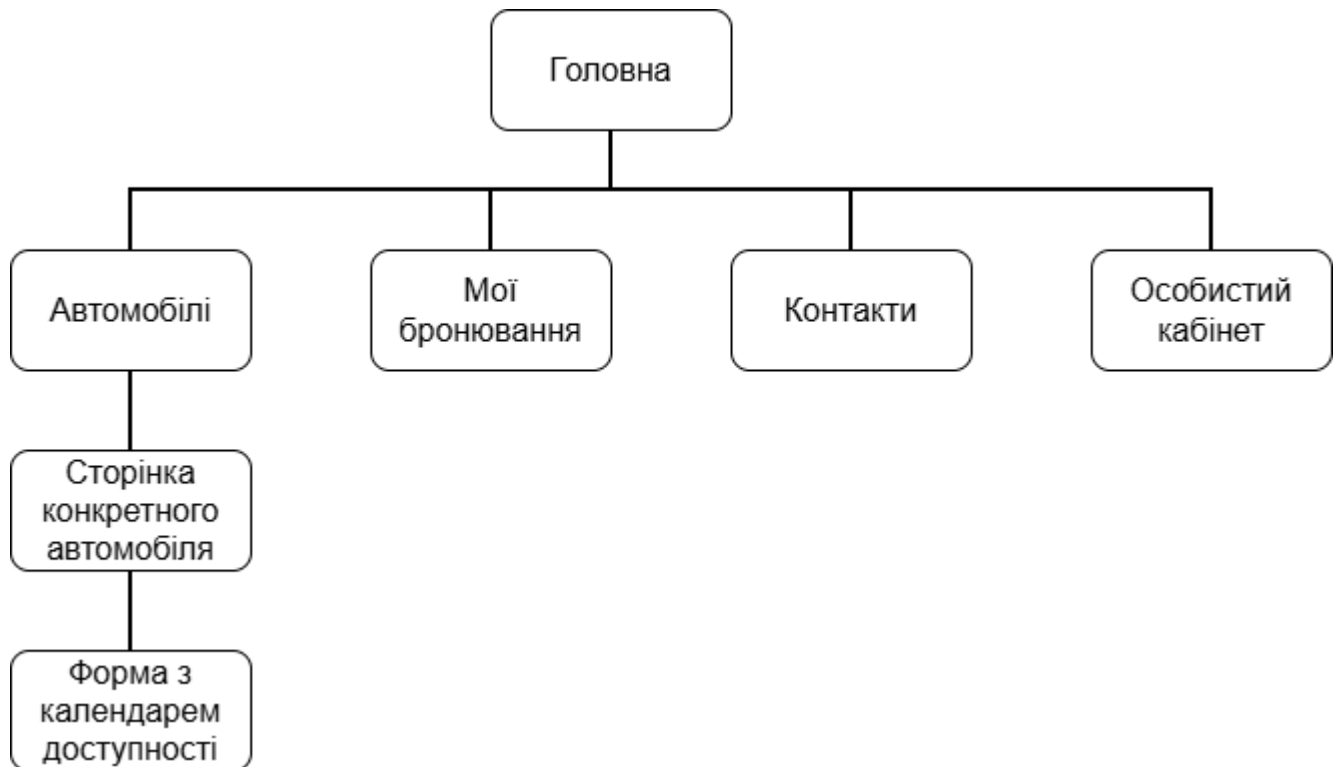


Рис 2.7 Мапа сайту для клієнта

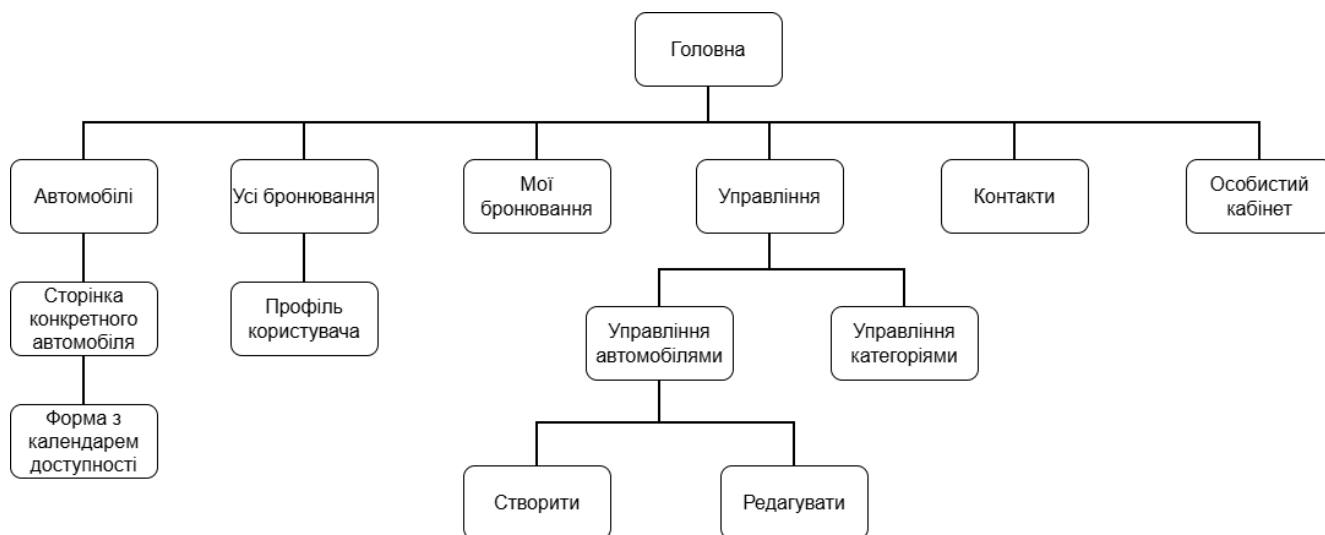


Рис. 2.8 Мапа сайту для менеджера

Спроектowana структура дає уявлення про кількість та призначення майбутніх сторінок і значно полегшить процес розробки.

З точки зору безпеки і розділення прав, видимість сторінок на панелі навігації залежить від вмісту JWT токена користувача. Для зручного поширення стану аутентифікації використовується механізм контексту бібліотеки React. Це дозволяє один раз перевірити вміст токена, а надалі – з будь якого компоненту застосунку безперешкодно отримувати доступ до інформації щодо ролі користувача і показувати на сторінці лише потрібні елементи.

2.5 Архітектура бази даних

База даних містить чотири таблиці, які дзеркально повторюють структуру класів сутностей системи: автомобілі (vehicles), категорії (categories), користувачі (users), бронювання (reservations). На рисунку 2.9 зображена даталогічна модель бази даних, або ER-діаграма бази даних. На ній зображені таблиці, поля, типи полів та зв'язки між таблицями у базі даних.

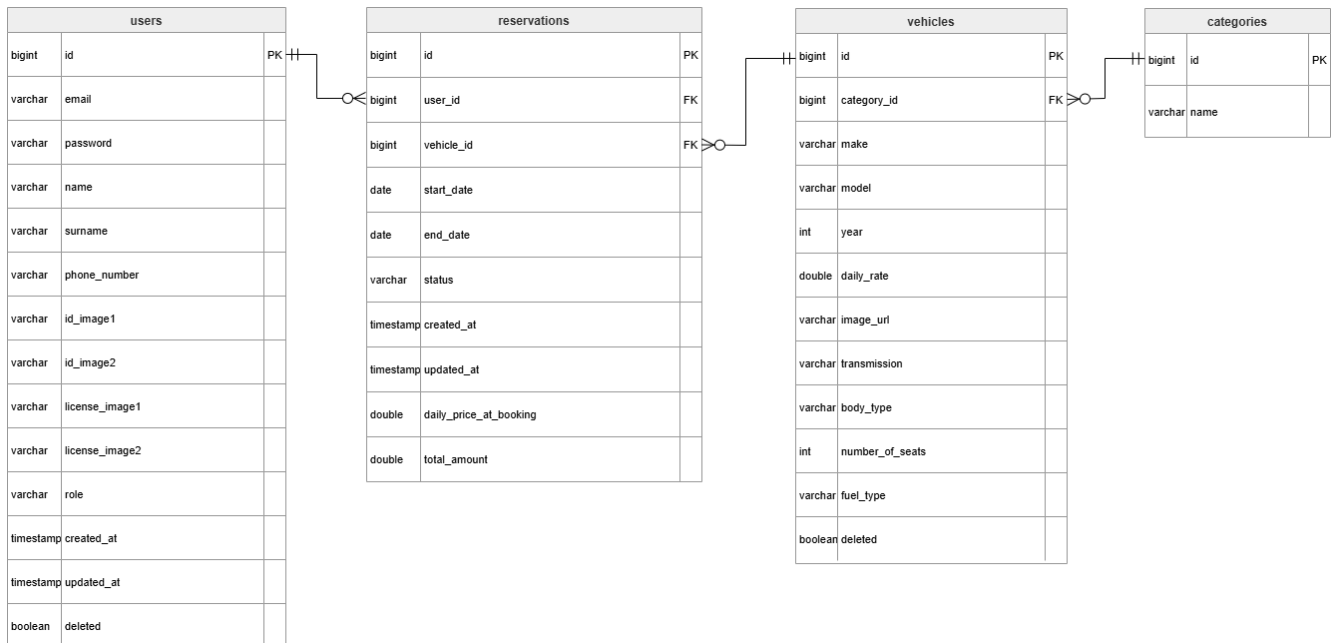


Рис. 2.9 ER-діаграма бази даних

Важливим моментом при проєктуванні баз даних є забезпечення цілісності даних в таблицях. На діаграмі видно, що таблиця `reservations` має зв'язки один-до-багатьох з таблицями `users` та `vehicles`. Таким чином, таблиця `reservations` стає проміжною таблицею у формуванні зв'язку багато-до-багатьох між `users` та `vehicles`. Наслідком цього стає залежність таблиці від наявності посилань на відповідних користувачів та автомобілі. Якщо ж, наприклад, менеджер вирішить видалити автомобіль (який був заброньований хоч раз) з каталогу, то посилання на первинний ключ цього автомобіля в таблиці бронювань стане недійсним і цілісність даних буде втрачено. Один з варіантів вирішення такої проблеми – дозволити каскадне видалення і видаляти всі пов'язані бронювання разом з автомобілем.

Але такий підхід суперечить бізнес-логіці системи, адже ми втратимо історичність даних – користувачі не побачать своїх минулих бронювань в особистому кабінеті, а менеджери не зможуть, наприклад, створити коректний звіт з доходами за місяць. Тому при проєктуванні обираємо інший шлях – механізм м'якого видалення. Для його імплементатії створюємо поле `deleted` для таблиць `users` та `vehicles`. Тепер, при видаленні авто чи користувача, просто ставимо

необхідний прапорець в БД і не показуємо такі об’єкти там, де це не потрібно, але зберігаємо історичність і цілісність даних.

2.6 Алгоритм роботи модуля семантичного пошуку

Важливою інноваційною компонентою розроблюваної системи є впровадження семантичного пошуку, що працюватиме за допомогою штучного інтелекту. Класичні пошукові рядки в застосунках дозволяють користувачам шукати за конкретними назвами товарів чи послуг. Але іноді зручніше описати свою потребу природною мовою – неначе пояснюєш, що саме шукаєш, іншій людині. Такий підхід може бути актуальним і в контексті оренди авто. Наприклад, запит може звучати як “дизельне авто для пересічної місцевості на автоматі”, або “недорога машина на механіці”. Саме такі запити користувачів будемо обробляти за допомогою великої мовної моделі Gemini, яка буде виконувати роль своєрідного NLP-аналізатора.

Найпершою інтуїтивною ідеєю для реалізації такого функціоналу, було би звертатися до API мовної моделі, задавши промпт на фільтрацію, і передаючи із запитом користувача усі доступні в каталозі авто. Проте, такий спосіб буде значно гальмувати процес обробки запитів, якщо каталог доволі великий і постійно поповнюється. Також, при такому підході значно зростає ризик галюцинації ШІ – мовна модель може просто “вигадати” результат, сумістивши характеристики різних автомобілів. Тому обрано інший підхід – разом із запитом користувача будемо передавати всі доступні параметри фільтрації (іншими словами, всі доступні варіанти характеристик авто), а на виході очікувати JSON-файл з конкретним параметром для кожного фільтру (для кожної характеристики). Задачу ж самої фільтрації залишаємо надійній реляційній БД PostgreSQL, будуючи SQL запити на основі отриманого JSON-файлу. Спрощена схема процесу зображена на рисунку 2.10.

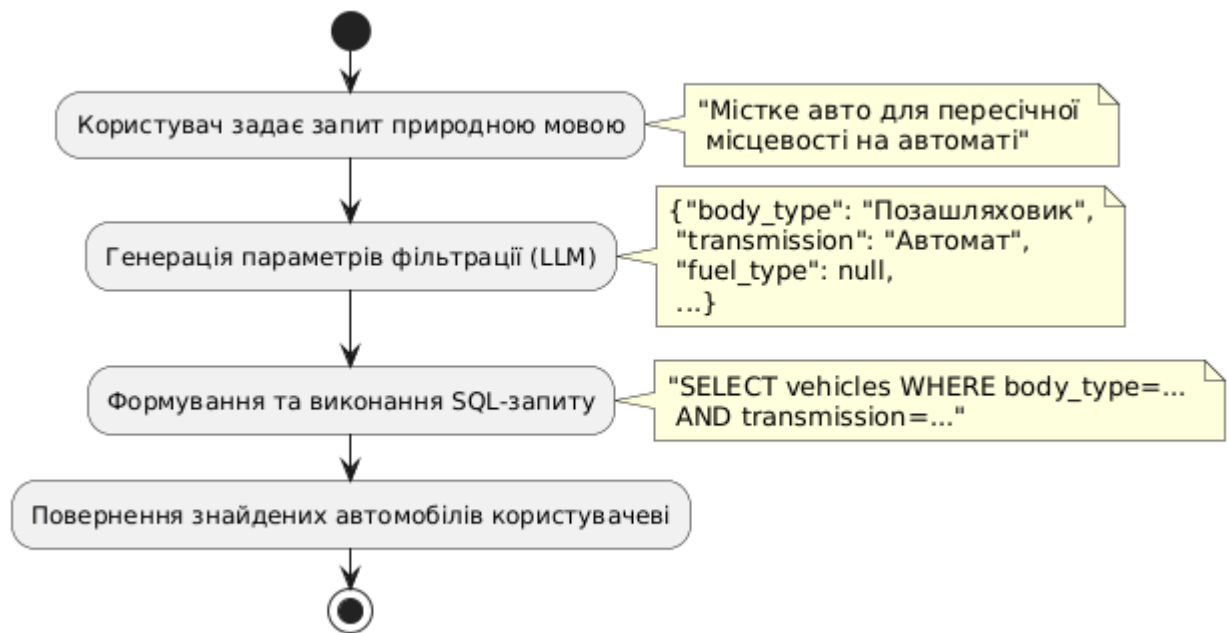


Рис. 2.10 Спрощена діаграма діяльності семантичного пошуку

При проектуванні такого нестандартного рішення, маємо враховувати і попередній досвід користувачів. Якщо клієнт захоче скористатися пошуком, але введе в нього конкретну марку чи модель авто, то система все одно повинна видати правильний результат.

Насправді, більшість сучасних мовних моделей ШІ мають механізм розпізнавання іменованих сутностей (named entity recognition), тому наша система зможе обробити і такі запити (наприклад, “toyota corolla”), і навіть комбіновані запити на кшталт “toyota corolla на автоматі”. Єдине, що доведеться зробити при розробці – це включити до бажаного вихідного JSON-файлу, який отримуємо від ШІ, поля `make` (марка) та `model` (модель).

Діаграма послідовності на рисунку 2.11 зображає детальний процес взаємодії окремих частин застосунку під час використання клієнтом компонента семантичного пошуку.

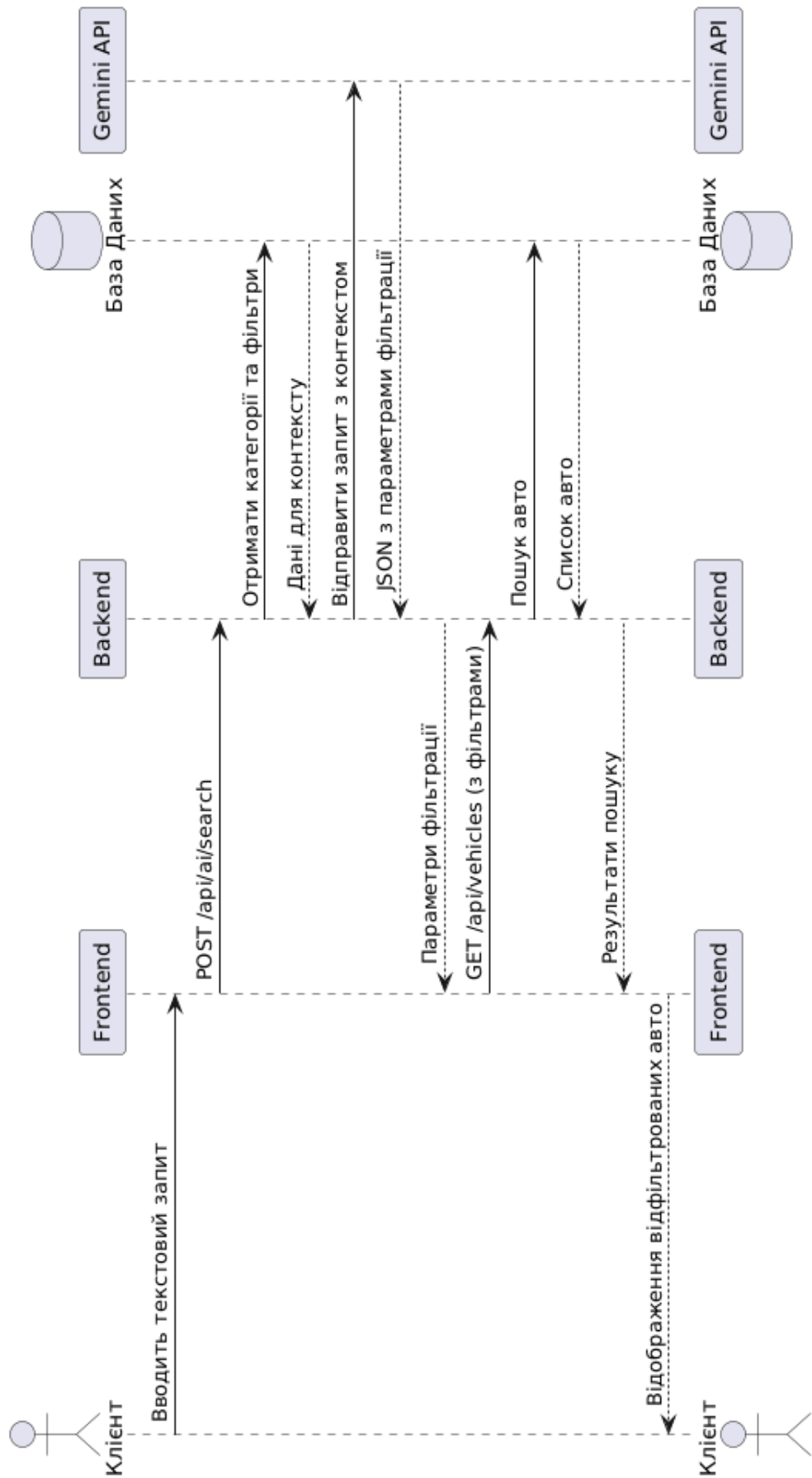


Рис. 2.11 Діаграма послідовності роботи семантичного пошуку

3 ОПИС РЕАЛІЗАЦІЇ СИСТЕМИ

3.1 Реалізація серверної частини

Наступним етапом розробки є програмна реалізація серверної частини. Для демонстрації роботи різних шарів застосунку в цьому розділі буде продемонстровано код тих класів, що відповідають за логіку роботи з категоріями автомобілів, оскільки вони найменш громіздкі і добре показують логіку взаємодії різних класів у системі.

Перший представлений клас – клас доменної моделі або сутності (див. рис. 3.1). В першу чергу він відповідає за представлення об'єкту бізнес логіки, тобто категорії транспортного засобу, а також описує як цей об'єкт буде зберігатися в базі даних. Клас категорії містить поля `id`, `name` та `vehicles`. Останнє, зокрема, використовується для зв'язку з сутністю `Vehicle`.

Над самим класом та його полями є деякі анотації, кожна з яких відповідає за певний функціонал. Умовно їх можна поділити на 2 групи: JPA анотації (частина Spring Boot) та Lombok анотації (генерація шаблонного коду). Ось за що відповідають деякі з них:

- `@Entity`: Вказує фреймворку Hibernate/JPA, що цей клас є сутністю і його екземпляри потрібно зберігати в базі даних.
- `@Table(name = "categories")`: Вказує назву таблиці в базі даних, з якою пов'язаний цей клас.
- `@Id`: Позначає поле `id` як первинний ключ.
- `@Column(nullable = false, unique = true)`: Встановлює обмеження на рівні схеми БД для поля `name`. Воно не може бути порожнім та повинно бути унікальним для кожної категорії.
- `@OneToMany(mappedBy = "category")`: Описує зв'язок “один-до-багатьох”. Одна категорія може мати багато транспортних засобів (`vehicles`). Параметр

mappedBy вказує, що фізично цей зв'язок контролюється полем category в класі Vehicle.

- @NoArgsConstructor та @AllArgsConstructor: Автоматично генерують порожній конструктор (обов'язкова вимога JPA) та конструктор з усіма полями класу.
- @Getter і @Setter: Генерують методи доступу та зміни (гетери і сетери) для всіх полів.

```

Category.java

@Entity
@Table(name = "categories")
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
@Builder
public class Category {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Long id;

    @Column(nullable = false, unique = true)
    private String name;

    @Builder.Default
    @OneToMany(mappedBy = "category")
    private List<Vehicle> vehicles = new ArrayList<>();

    @Override
    public boolean equals(Object o) {
        if (o == null || getClass() != o.getClass()) return false;
        Category category = (Category) o;
        return Objects.equals(id, category.id) && Objects.equals(name, category.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(id, name);
    }
}

```

Рис. 3.1 Код класу Category.java



```
CategoryRepository.java

@Repository
public interface CategoryRepository extends JpaRepository<Category, Long> {

    boolean existsByNameIgnoreCase(String name);

}
```

Рис. 3.2 Код інтерфейсу CategoryRepository.java

Наступний шар – це шар доступу до даних, представлений інтерфейсом репозиторію (див. рис. 3.2). В Spring Boot такі інтерфейси позначаються анотацією `@Repository`. Цей шар інкапсулює всі операції з базою даних. Завдяки наслідуванню від `JpaRepository`, нам не потрібно писати базові SQL-запити для CRUD-операцій (створення, читання, оновлення, видалення), вони автоматично створюються за нас. Але якщо потрібна специфічна логіка, то можна описати її в анотації `@Query`, або скористатись синтаксичним парсером, вбудованим в Spring Data, і просто вказати назву необхідного методу за певною логікою, описаною в документації. Так, Spring Boot “зрозуміє” призначення нашого методу `existsByNameIgnoreCase` навіть без вказання конкретної реалізації.

Далі – сервісний шар, який по своїй суті є серцем застосунку (див. рис. 3.3). Саме тут зосереджена вся бізнес-логіка та правила системи. Сервіс бере дані з бази через репозиторій, обробляє їх і гарантує, що система працює за правилами. Класи сервісного шару позначаються анотацією `@Service`. Для того, щоб отримати доступ до методів репозиторію, використовується механізм `dependency injection`. Це означає, що екземпляр репозиторію не створюється всередині класу, а передається ззовні, що гарантує нам Spring Boot.

```

CategoryService.java

@Service
@RequiredArgsConstructor
public class CategoryService {

    private final CategoryRepository categoryRepository;

    public List<Category> listCategories() {
        return categoryRepository.findAll();
    }

    @Transactional
    public Category createCategory(Category category) {
        String categoryName = category.getName();
        if (categoryRepository.existsByNameIgnoreCase(categoryName)) {
            throw new IllegalArgumentException("Category already exists with name: " +
categoryName);
        }
        return categoryRepository.save(category);
    }

    @Transactional
    public void deleteCategory(Long id) {
        Optional<Category> category = categoryRepository.findById(id);
        if (category.isPresent()) {
            if (!category.get().getVehicles().isEmpty()) {
                throw new IllegalStateException("Category has vehicles associated with it");
            }
            categoryRepository.deleteById(id);
        }
    }

    public Category findCategoryById(Long categoryId) {
        return categoryRepository.findById(categoryId)
            .orElseThrow(() -> new EntityNotFoundException("Category not found by id: " +
categoryId));
    }
}

```

Рис. 3.3 Код класу CategoryService.java

В прикладі з сервісом категорій, методи лише перевіряють деякі умови і викликають виключення, але бізнес-логіка може бути скільки завгодно складною. Анотація `@Transactional` над методами гарантує, що операція або виконається повністю, або відкотиться у разі збою.

```

CategoryController.java

@RestController
@RequestMapping(path = "/api/categories")
@RequiredArgsConstructor
public class CategoryController {

    private final CategoryService categoryService;
    private final CategoryMapper categoryMapper;

    @GetMapping
    public ResponseEntity<List<CategoryDto>> listCategories() {
        List<CategoryDto> categories = categoryService
            .listCategories()
            .stream()
            .map(categoryMapper::toDto)
            .toList();
        return ResponseEntity.ok(categories);
    }

    @PostMapping
    public ResponseEntity<CategoryDto> createCategory(
        @Valid @RequestBody CreateCategoryRequestDto createCategoryRequestDto) {
        Category categoryToCreate = categoryMapper.toEntity(createCategoryRequestDto);
        Category savedCategory = categoryService.createCategory(categoryToCreate);
        return new ResponseEntity<>(categoryMapper.toDto(savedCategory), HttpStatus.CREATED);
    }

    @DeleteMapping(path =("/{id}")
    public ResponseEntity<Void> deleteCategory(@PathVariable Long id) {
        categoryService.deleteCategory(id);
        return new ResponseEntity<>(HttpStatus.NO_CONTENT);
    }
}

```

Рис. 3.4 Код класу CategoryController.java

Далі – презентаційний шар, або шар контролерів (див. рис. 3.4). Він відповідає за обробку HTTP-запитів від клієнтів і формування HTTP-відповідей. Для того, щоб позначити клас як контролер, використовується анотація `@RestController`. `@RequestMapping(path = "/api/categories")` визначає базовий маршрут (URL), за яким будуть доступні всі методи цього класу. Далі, кожен метод може доповнювати цей URL, як `deleteCategory`. Кожен метод у контролері відповідає за обробку конкретного типу HTTP-запиту: GET, POST, DELETE. Це також досягається використанням анотацій: `@GetMapping`, `@PostMapping`, `@DeleteMapping`.

У контролері ми бачимо класи на кшталт CategoryDto та CreateCategoryRequestDto, а також CategoryMapper. Це дуже важливий архітектурний підхід, адже сутності бази даних (entity) не можна віддавати “назовні” клієнту. Вони можуть містити паролі, службові поля БД, або викликати нескінченні цикли. Тому використовуються DTO (data transfer object) – прості класи без логіки, створені спеціально для передачі інформації між фронтендом і бекендом. Вони містять лише те, що потрібно побачити або відправити користувачу. Mapper, в свою чергу, відповідає за перетворення даних з сутності в DTO і навпаки.

Кінцевим продуктом реалізації серверної частини є RESTful API, яке відповідатиме за приймання запитів від клієнтської частини та формуватиме відповіді. Всі ендпоінти цього API та їх призначення представлено в таблиці 3.1.

Таблиця 3.1

Перелік ендпоінтів API

HTTP метод	Ендпоінт	Призначення
POST	/api/auth/login	Авторизація користувача та отримання JWT токена.
POST	/api/auth/send-code	Відправка коду верифікації на електронну пошту для реєстрації.
POST	/api/auth/register	Реєстрація нового користувача (потребує перевірки коду).
GET	/api/categories	Отримання списку всіх категорій транспортних засобів.
POST	/api/categories	Створення нової категорії.
DELETE	/api/categories/{id}	Видалення категорії за її ідентифікатором (ID).
POST	/api/vehicles	Додавання нового транспортного засобу.

Продовження таблиці 3.1

Перелік ендпоінтів API

HTTP метод	Ендпоінт	Призначення
GET	/api/vehicles	Отримання списку авто з підтримкою пагінації та фільтрації (за категорією, маркою, моделлю тощо).
GET	/api/vehicles/search-options	Отримання доступних опцій для фільтрації авто (унікальні марки, моделі, типи пального тощо).
GET	/api/vehicles/recently-reserved	Отримання списку нещодавно заброньованих транспортних засобів.
GET	/api/vehicles/{id}	Отримання детальної інформації про конкретний транспортний засіб за ID.
GET	/api/vehicles/{id}/unavailable-dates	Отримання списку недоступних (заброньованих) дат для транспортного засобу.
PUT	/api/vehicles/{id}	Оновлення інформації про існуючий транспортний засіб.
DELETE	/api/vehicles/{id}	Видалення транспортного засобу.
POST	/api/vehicles/{id}/image	Завантаження зображення для авто (зберігається в AWS S3).
GET	/api/users/me	Отримання профілю поточного авторизованого користувача.
GET	/api/users/{id}	Отримання даних користувача за ID (для менеджерів).
PUT	/api/users/me	Оновлення даних профілю поточного користувача.
POST	/api/users/me/documents/{documentType}	Завантаження документів користувача (паспорт, водійське посвідчення) за типом.
DELETE	/api/users/{id}	Видалення/блокування користувача за ID (тільки для менеджерів).
POST	/api/reservations	Створення нового бронювання.
GET	/api/reservations	Отримання списку всіх бронювань (з пагінацією).

Продовження таблиці 3.1

Перелік ендпоінтів API

HTTP метод	Ендпоінт	Призначення
GET	/api/reservations/my	Отримання списку бронювань поточного авторизованого користувача.
GET	/api/reservations/{id}	Отримання детальної інформації про конкретне бронювання.
PATCH	/api/reservations/{id}/confirm	Підтвердження бронювання.
PATCH	/api/reservations/{id}/cancel	Скасування бронювання.
DELETE	/api/reservations/{id}	Видалення запису про бронювання.
GET	/api/files/public/vehicles/images/{fileName}	Отримання публічного посилання на зображення транспортного засобу (редирект на S3).
GET	/api/files/private/users/{userId}/documents/{fileName}	Отримання захищеного посилання на приватний документ користувача (редирект на S3).
POST	/api/ai/parse-query	Обробка текстового запиту за допомогою AI для інтелектуального пошуку транспортних засобів.

3.2 Реалізація клієнтської частини

Реалізація клієнтської частини веб-застосунку виконана з використанням сучасного фреймворку Next.js та бібліотеки React. Архітектура фронтенду побудована за модульним принципом, де чітко розділені шари відображення, управління станом та взаємодії з сервером.

Взаємодія з бекендом абстрагована в окремий шар API-сервісів, що дозволяє ізолювати логіку HTTP-запитів від візуальних компонентів. Наприклад, у файлі `lib/api/reservations.ts` (див. рис. 3.5) зібрані всі методи для роботи з життєвим циклом бронювань. Замість написання запитів безпосередньо у компонентах, застосовується попередньо налаштований екземпляр Axios (імпортований як `apiClient`), який відповідає за автоматичне додавання заголовків авторизації та обробку глобальних помилок.

Цей файл містить набір асинхронних функцій, таких як `getMy` для отримання списку бронювань поточного користувача або `create` для оформлення нової оренди.

Для ефективного управління асинхронним станом та кешуванням отриманих даних на стороні клієнта інтегровано бібліотеку TanStack React Query. Файл `lib/hooks/useApi.ts` (див. рис. 3.6) відіграє роль “містка” між чистими API-запитами та React-компонентами. У ньому оголошуються власні кастомні хуки. Зокрема, хук `useMyReservations` використовує `useQuery` для ініціалізації запиту, автоматичного кешування відповіді та надання компонентам зручних прапорців стану, таких як `isLoading` або `error`. Для модифікації даних (створення, оновлення, видалення) застосовуються хуки на базі `useMutation`, такі як `useCreateReservation`.

```
lib/api/reservations.ts

import { apiClient } from './client';
import { ReservationDto, CreateReservationRequestDto } from '../types/reservation';
import { PageResponse } from '../types/api';

export const reservationsApi = {
  ...

  getMy: async (params?: Record<string, any>): Promise<PageResponse<ReservationDto>> => {
    const response = await apiClient.get<PageResponse<ReservationDto>>('/reservations/my', {
      params });
    return response.data;
  },

  ...

  confirm: async (id: number): Promise<ReservationDto> => {
    const response = await apiClient.patch<ReservationDto>(`/reservations/${id}/confirm`);
    return response.data;
  },

  ...
};
```

Рис. 3.5 Частина коду із файлу `lib/api/reservations.ts`

Побудова користувацького інтерфейсу базується на композиції ізольованих компонентів. Прикладом є файл `ReservationForm.tsx`, який відповідає за інтерактивний процес оформлення оренди транспортного засобу. Для управління внутрішнім станом форми та її валідації застосовується зв'язка бібліотек React

Hook Form та Zod. Вона дозволяє декларативно описати правила перевірки вхідних даних (наприклад, перевірку наявності дат) ще до того, як запит буде відправлено на сервер. Компонент також інтегрований з UI-бібліотекою `react-day-picker` для візуального вибору діапазону дат. При цьому форма використовує хук `useVehicleUnavailableDates`, щоб динамічно блокувати в календарі ті дні, коли автомобіль вже заброньований іншими користувачами. Після коректного заповнення форми викликається функція `onSubmit`, яка ініціює мутацію для відправки підготовлених даних на бекенд.

```
lib/hooks/useApi.ts

export const useMyReservations = (params?: Record<string, any>) => {
  return useQuery({
    queryKey: ['reservations', 'my', params],
    queryFn: () => reservationsApi.getMy(params),
  });
};

export const useReservation = (id: number | null) => {
  return useQuery({
    queryKey: ['reservations', id],
    queryFn: () => reservationsApi.getById(id!),
    enabled: !!id,
  });
};

export const useCreateReservation = () => {
  const queryClient = useQueryClient();
  return useMutation({
    mutationFn: (data: CreateReservationRequestDto) => reservationsApi.create(data),
    onSuccess: () => {
      queryClient.invalidateQueries({ queryKey: ['reservations'] });
    },
  });
};
```

Рис. 3.6 Приклад кастомних хуків

На найвищому рівні ієрархії фронтенду знаходяться файли сторінок (pages), які в Next.js відповідають за маршрутизацію та компонування загальної структури екрану. У файлі `app/reservations/page.tsx` (див. рис. 3.7) реалізована сторінка історії бронювань користувача. Оскільки сторінка потребує інтерактивності (наприклад, зміни сторінок у пагінації), вона оголошена як клієнтський компонент за

допомогою директиви 'use client'. Доступ до неї захищений обгорткою `ProtectedRoute`, яка гарантує, що контент побачить лише авторизований користувач. Сама сторінка виконує роль контейнера: вона керує локальним станом сторінки пагінації за допомогою стандартного `useState` та передає це значення у хук `useMyReservations`. Отриманий масив бронювань, разом зі станами завантаження, передається як `props` до дочірнього презентаційного компонента `ReservationList`. Для навігації по результатах використовується компонент `Pagination`. Такий патерн гарантує, що головний файл сторінки залишається чистим і відповідає виключно за оркестрацію потоків даних та відображення високорівневого макета, делегуючи деталі рендерингу вкладеним компонентам [20].

```

app/reservations/page.tsx

'use client';

import React, { useState } from 'react';
import { ProtectedRoute } from '@components/layout/ProtectedRoute';
import { useMyReservations } from '@lib/hooks/useApi';
import { ReservationList } from '@components/reservations/ReservationList';
import { Pagination } from '@components/common/Pagination';

export default function ReservationsPage() {
  const [page, setPage] = useState<number>(0);
  const { data: reservationsPage, isLoading, error } = useMyReservations({ page });

  return (
    <ProtectedRoute>
      <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 py-8">
        <h1 className="text-3xl font-bold text-gray-900 mb-6">Мої бронювання</h1>
        <ReservationList
          reservations={reservationsPage?.content}
          isLoading={isLoading}
          error={error}
        />
        {reservationsPage && (
          <Pagination
            currentPage={reservationsPage.number}
            totalPages={reservationsPage.totalPages}
            onPageChange={setPage}
          />
        )}
      </div>
    </ProtectedRoute>
  );
}

```

Рис. 3.7 Приклад сторінки `app/reservations/page.tsx`

3.3 Результати та демонстрація

Логічним завершенням етапів проєктування та програмної реалізації є готова до експлуатації інформаційна система. У даному підрозділі представлено результати практичного виконання проєкту. Для підтвердження працездатності розробленого застосунку нижче наведено візуальну демонстрацію його ключових функціональних можливостей, екранів користувацького інтерфейсу та сценаріїв взаємодії.

3.3.1 Головна сторінка

Головна сторінка є точкою входу в систему та слугує для привернення уваги користувача (див. рис. 3.8). Вона містить привітальний текст та заклик до дії у вигляді кнопки “Переглянути автомобілі”, яка направляє користувача безпосередньо до каталогу. Також тут знаходиться динамічна секція, яка відображає список транспортних засобів, що були заброньовані останніми. Це стимулює інтерес клієнта та демонструє активність платформи.

3.3.2 Каталог автомобілів

Це основна сторінка для пошуку та вибору транспортних засобів (див. рис. 3.9). Автомобілі представлені у вигляді карток з основними характеристиками та фотографіями. Панель фільтрів надає можливість ручного пошуку за: категорією авто, типом трансмісії, типом кузова, типом пального, мінімальною кількістю місць та максимальною добовою вартістю оренди. Також користувач може скористатися інтелектуальним пошуком, який дозволяє шукати автомобілі за допомогою запитів природною мовою і автоматично застосовує фільтри.

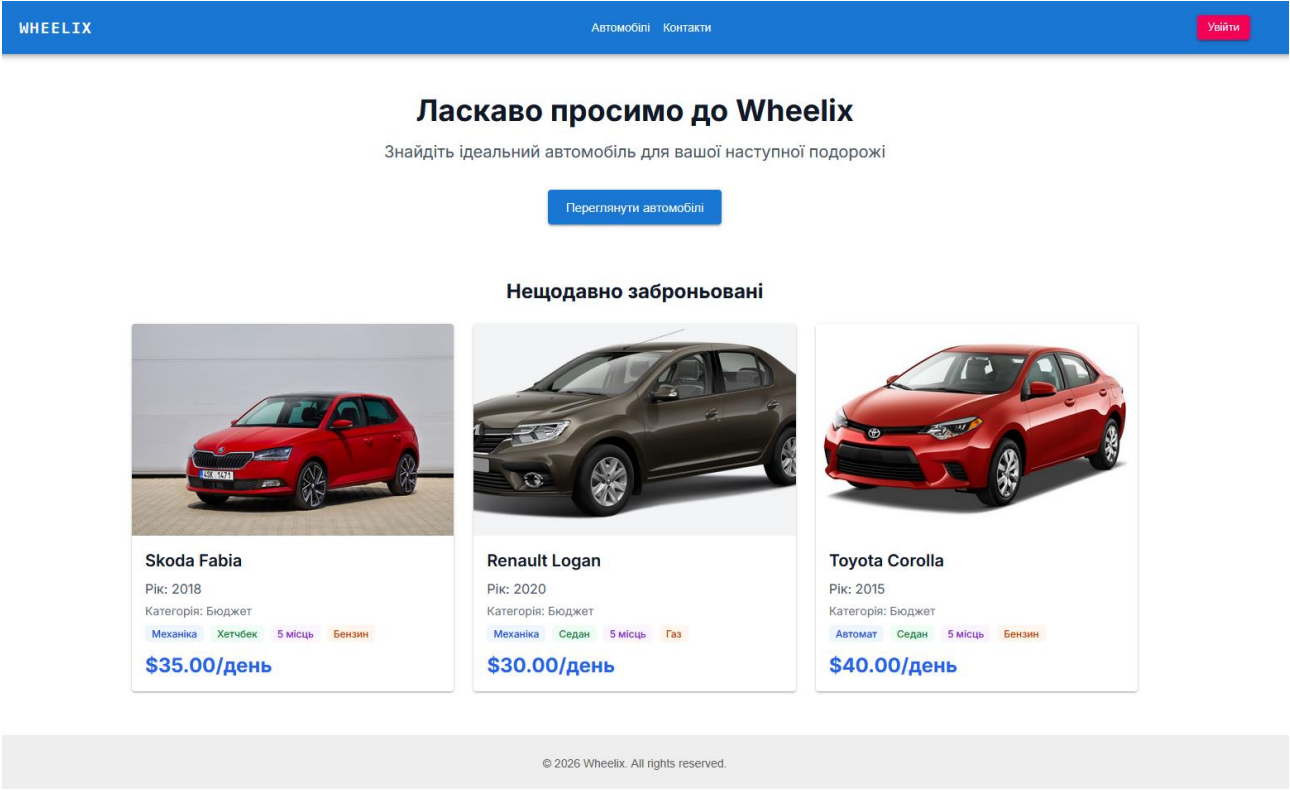


Рис. 3.8 Головна сторінка

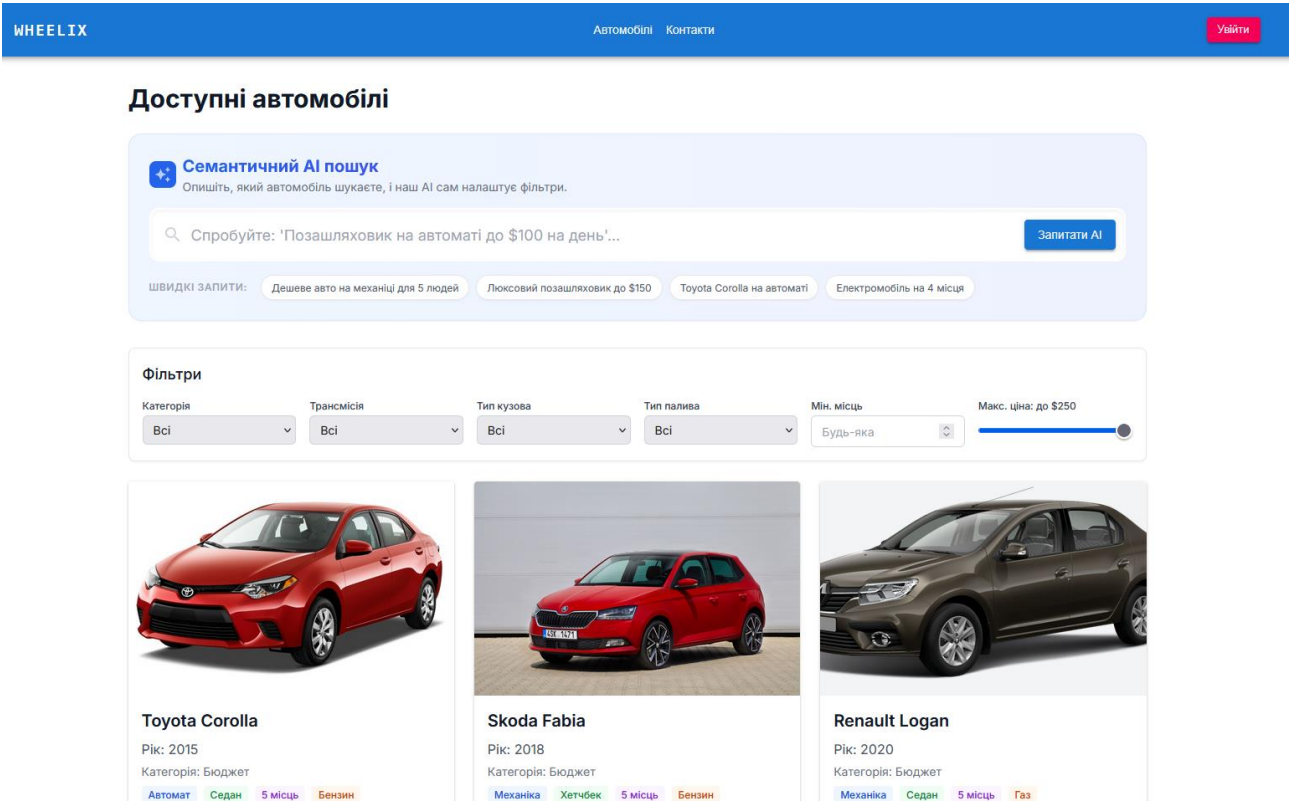


Рис. 3.9 Каталог автомобілів

3.3.3 Сторінка автомобіля

При натисканні на конкретний автомобіль з каталогу, користувач переходить на сторінку цього авто (див. рис. 3.10). Тут відображається інформація про транспортний засіб, а також кнопка “Забронювати цей автомобіль”. Якщо користувач не авторизований, система автоматично перенаправляє його на сторінку входу при натисканні на неї.

Для авторизованого користувача відкривається модальне вікно форми бронювання, де він може обрати дати початку та завершення оренди (див. рис. 3.11). Якщо користувач не вказав контактні дані в особистому кабінеті, то над формою є попередження про необхідність це зробити і посилання на сторінку профілю.

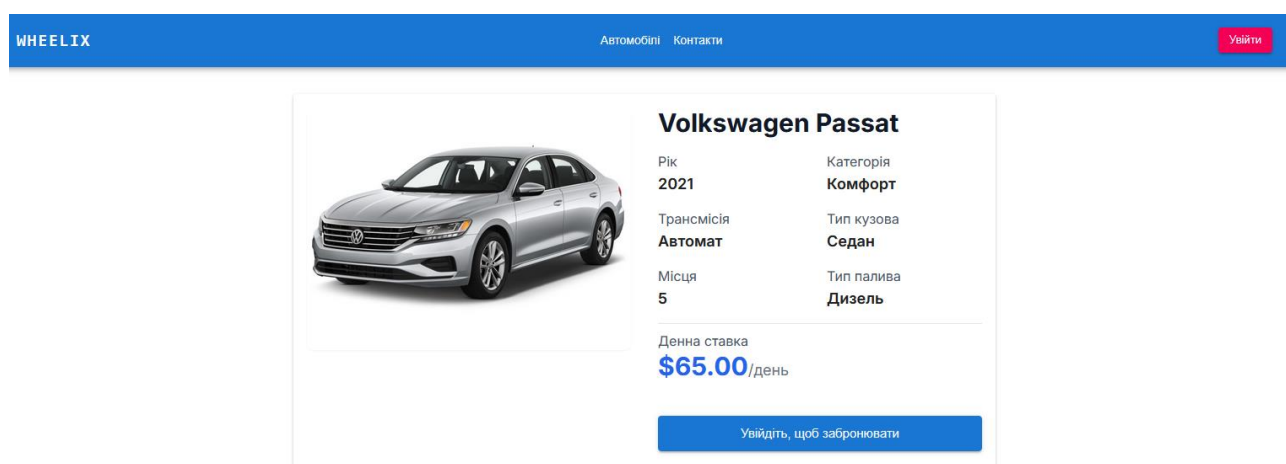


Рис. 3.10 Сторінка автомобіля

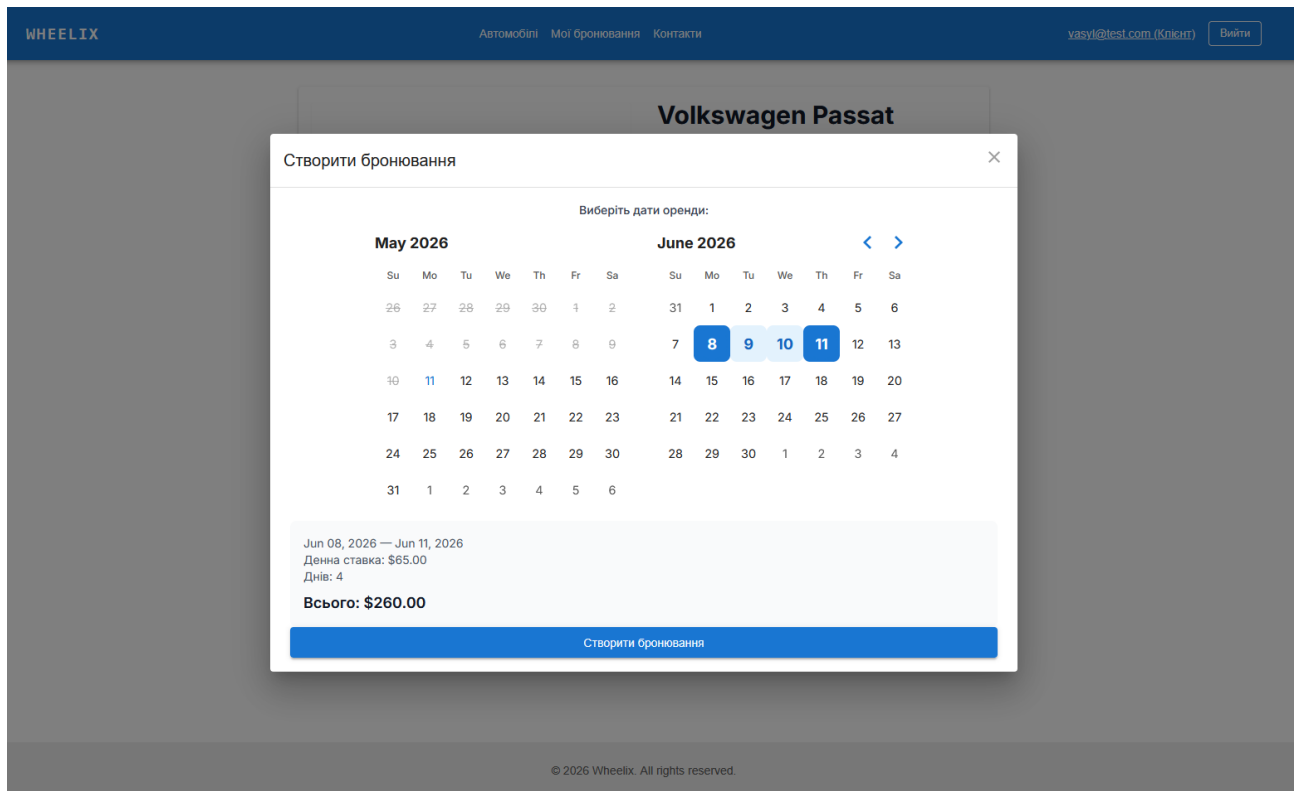


Рис. 3.11 Форма вибору дат для бронювання

3.3.4 Особистий кабінет користувача

Це сторінка керування персональними даними клієнта (див. рис. 3.12). Тут користувачу доступні перегляд та редагування контактних даних (ім'я, прізвище, номер телефону). Нижче знаходиться секція для завантаження сканів або фотографій документів, що посвідчують особу (див. рис. 3.13). Документи зберігаються у хмарному сховищі AWS S3.

Також система відображає поточний статус акаунту (підтверджено/не підтверджено) за допомогою таблички відповідного кольору у верхній частині профілю. Вона інформує клієнта про те, чи має він право здійснювати бронювання.

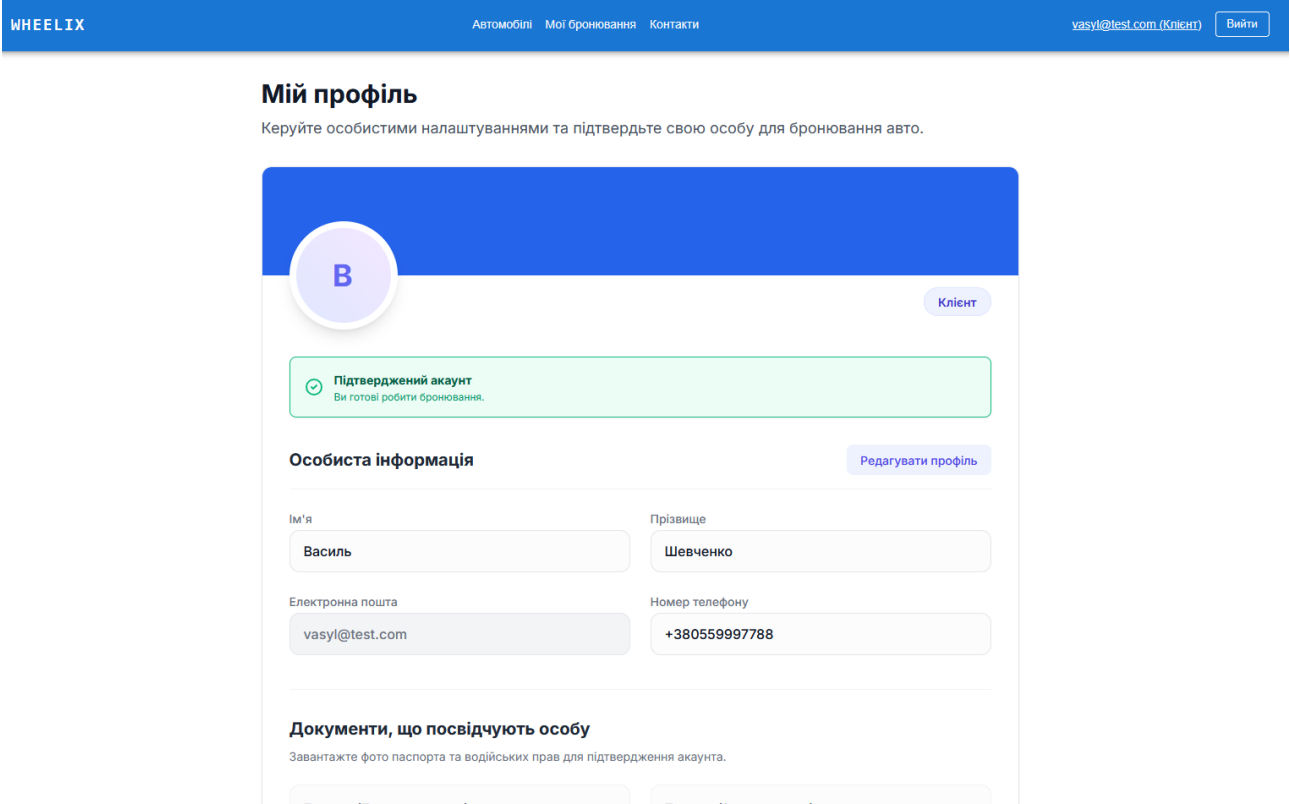


Рис. 3.12 Сторінка особистого кабінету (частина 1)

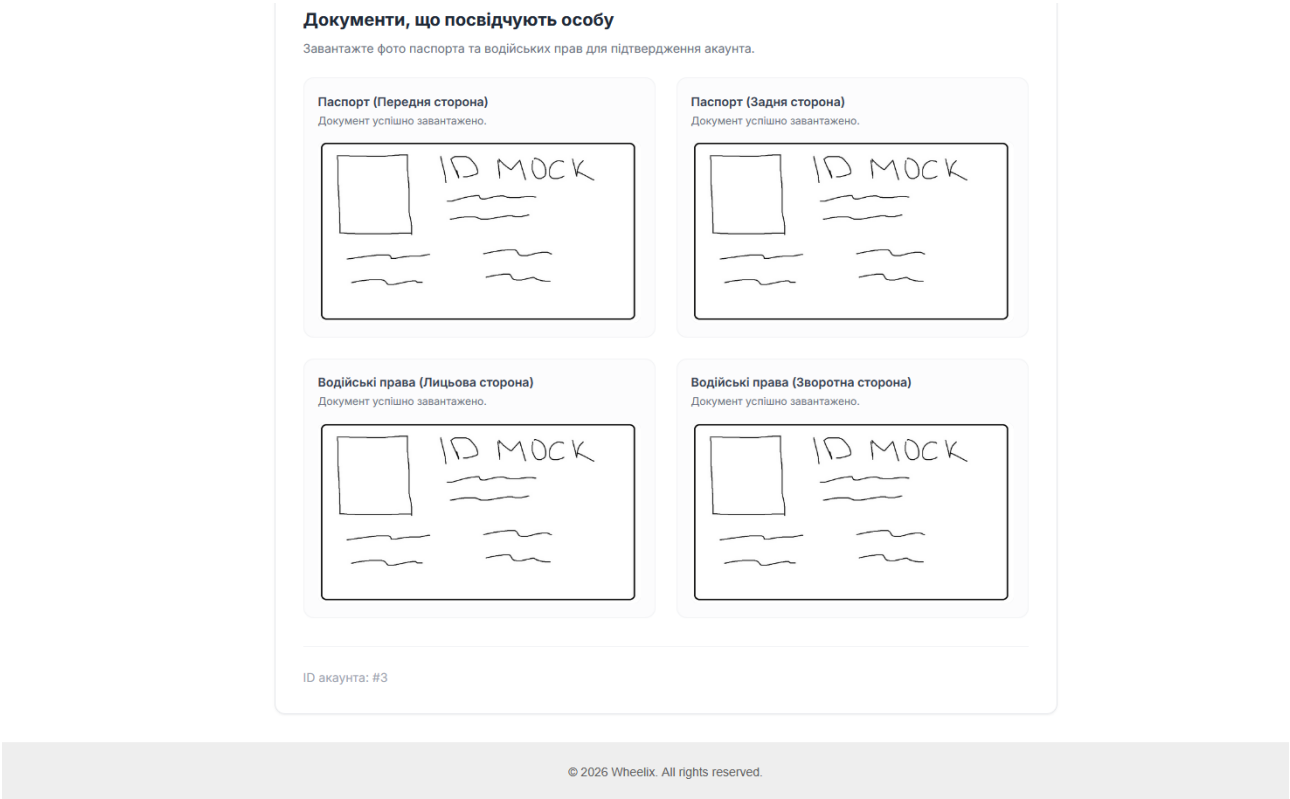


Рис. 3.13 Сторінка особистого кабінету (частина 2)

3.3.5 Мої бронювання

Це розділ для клієнтів, де вони можуть відстежувати історію своїх замовлень (див. рис. 3.14). Тут знаходяться картки з інформацією про орендовані авто, терміни оренди, загальну вартість та поточний статус бронювання. Вартість відображається на основі цін, що були актуальні саме в момент створення бронювання.

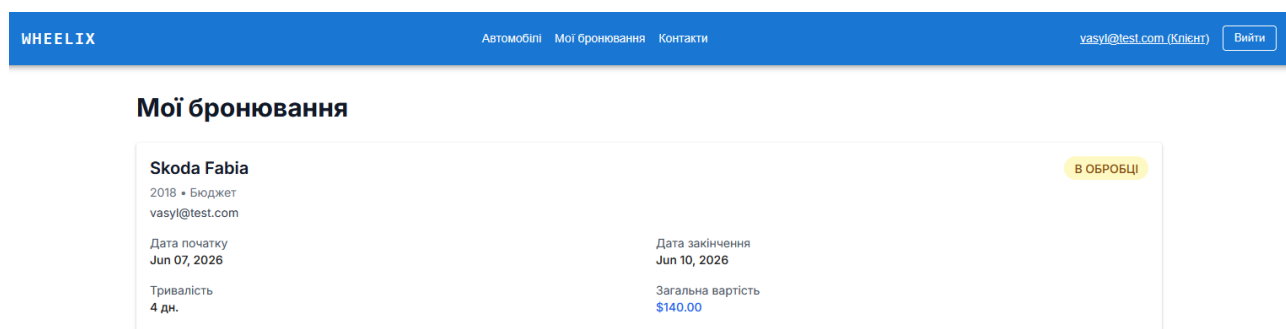


Рис. 3.14 Сторінка бронювань користувача

3.3.6 Контакти

Це інформаційна сторінка, яка забезпечує клієнтів даними для зв'язку з компанією. Тут відображається фізична адреса компанії, контактні номери телефонів, електронні пошти технічної підтримки та графік роботи офісу.

3.3.7 Управління автопарком

Доступно тільки менеджеру. Забезпечує повний CRUD функціонал для автопарку (див. рис. 3.15). Менеджер може додати новий транспортний засіб, завантажити його фото, відредагувати ціну чи характеристики (див. рис. 3.16), або видалити його з системи.

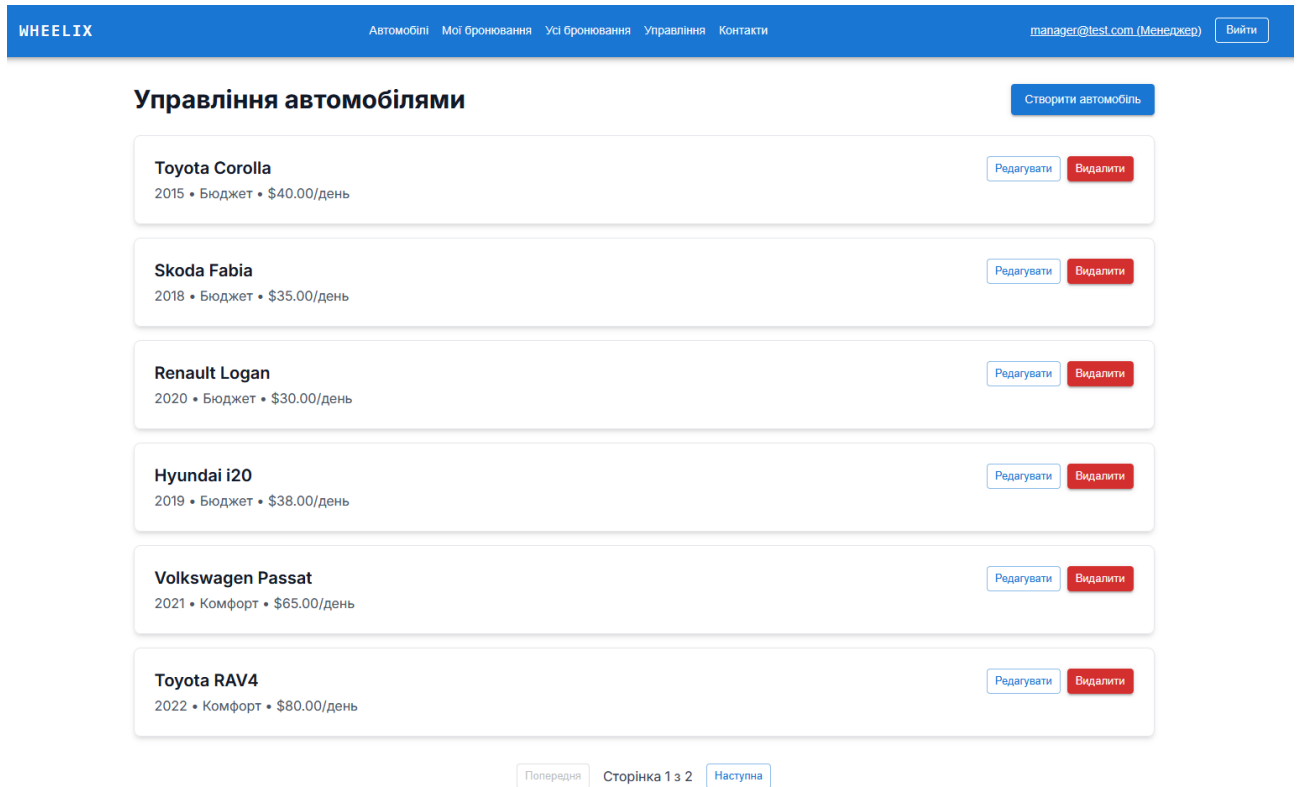


Рис. 3.15 Сторінка управління автопарком

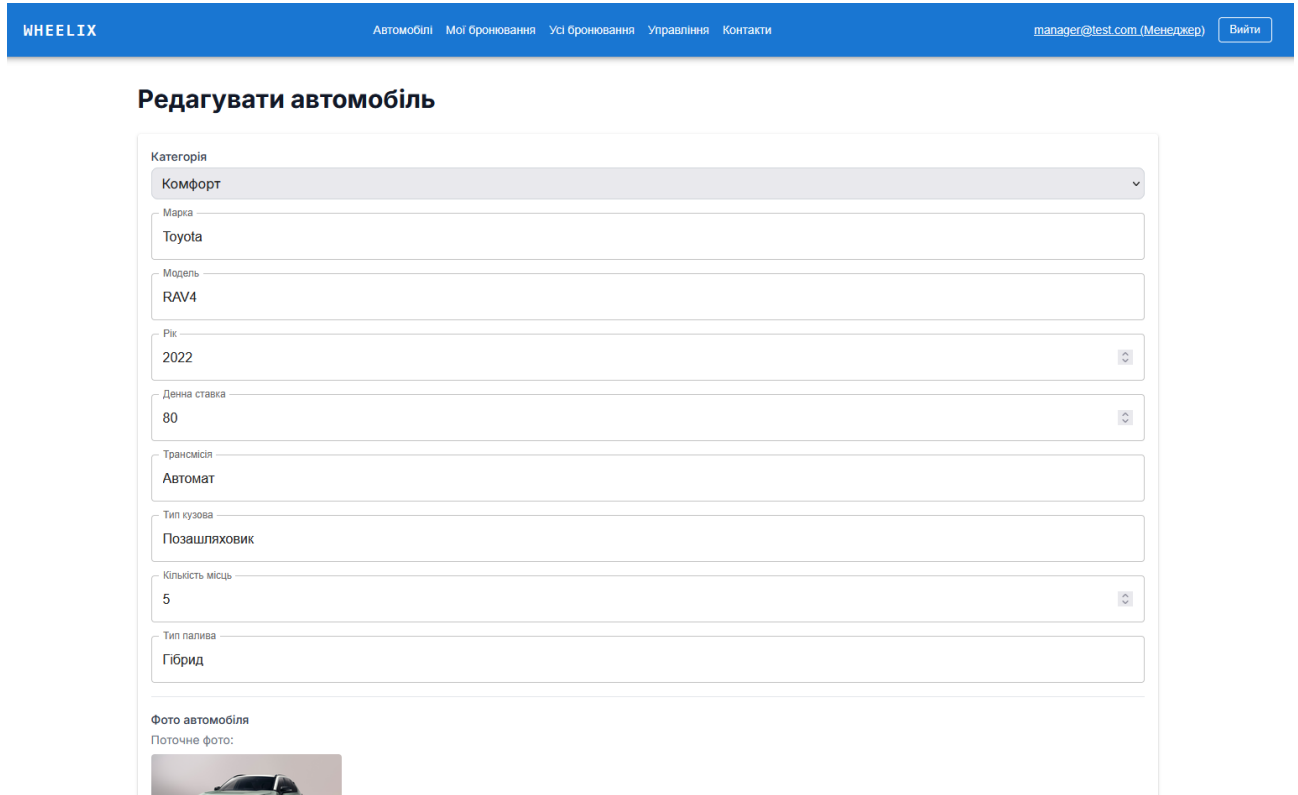


Рис. 3.16 Форма редагування характеристик авто

3.3.8 Управління категоріями

Доступно тільки менеджеру. Надає можливість переглядати, створювати нові та видаляти існуючі категорії автомобілів (див. рис. 3.17).

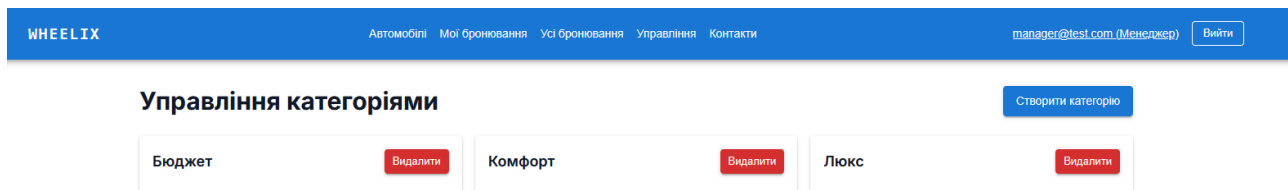


Рис. 3.17 Сторінка управління категоріями

3.3.9 Управління бронюваннями

Доступно тільки менеджеру. Забезпечує централізований перегляд усіх бронювань у системі (див. рис. 3.18). Менеджер має повноваження підтверджувати нові заявки від клієнтів, скасовувати їх або повністю видаляти записи.

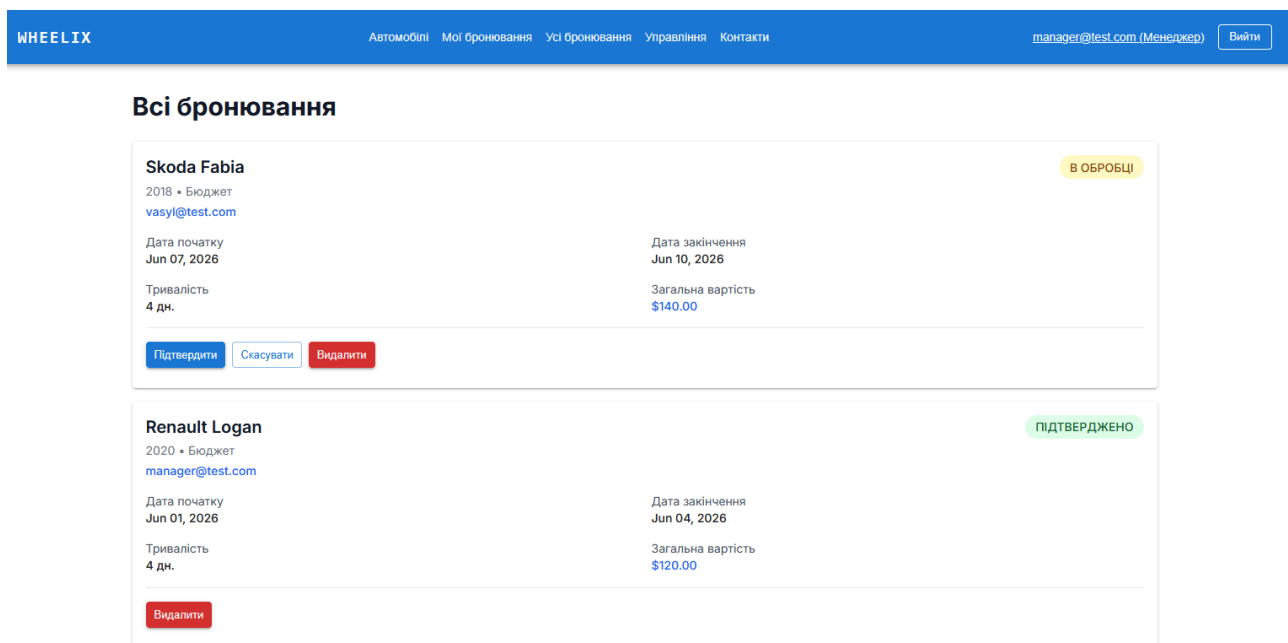


Рис. 3.18 Сторінка управління бронюваннями

3.3.10 Управління користувачами

Адміністратор може переглядати профілі клієнтів і перевіряти завантажені ними документи (паспорт і водійське посвідчення) для верифікації (див. рис. 3.19). Також реалізована можливість блокування користувачів за порушення правил сервісу.

3.3.11 Реєстрація та вхід

Стандартні форми для створення нового акаунту та авторизації існуючих клієнтів за допомогою електронної пошти та пароля (див. рис. 3.20).

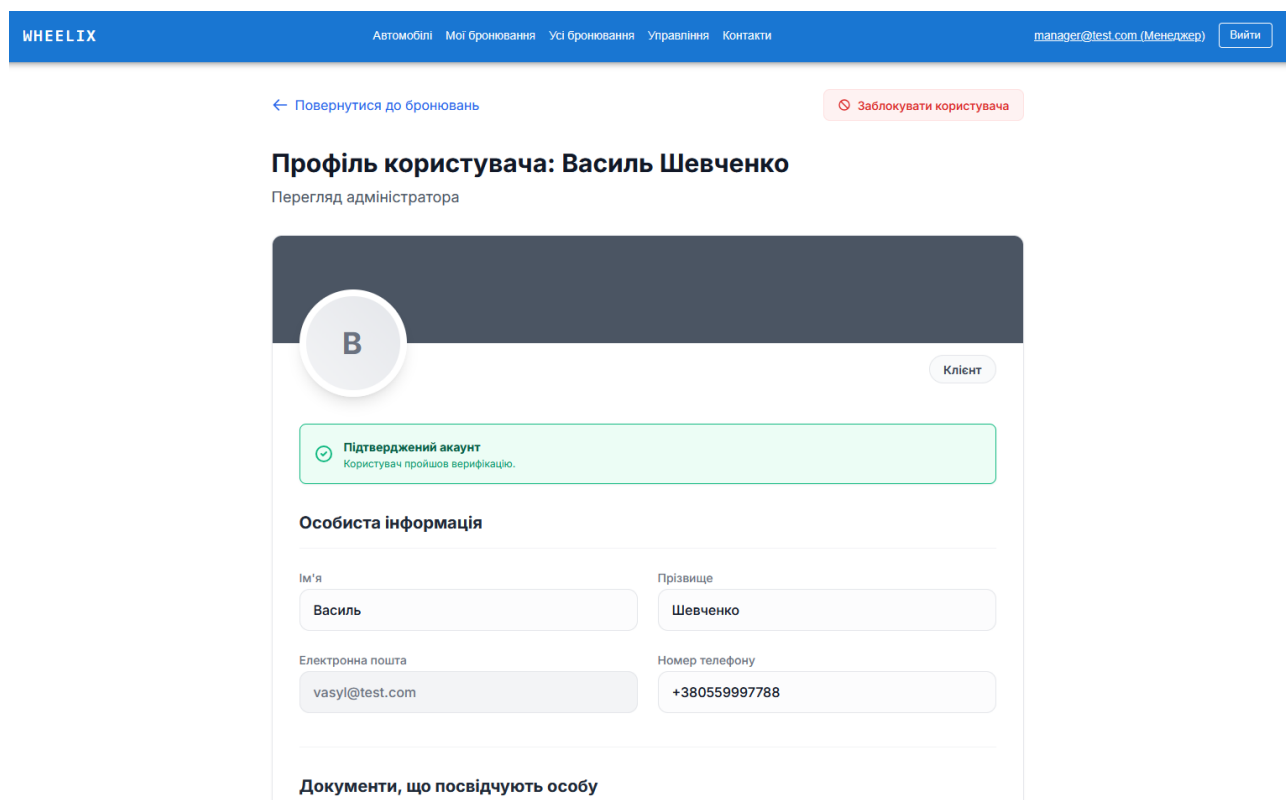


Рис. 3.19 Перегляд профілю клієнта менеджером

WHEELIX

Автомобілі Контакти

Увійти

Вхід

Електронна пошта

you@example.com

Пароль

Увійти

Немає акаунта? [Зареєструватися](#)

Рис. 3.20 Сторінка входу

4 ТЕСТУВАННЯ СИСТЕМИ

4.1 Обґрунтування видів тестування

Ключовим етапом в розробці програмного забезпечення є тестування. Воно дозволяє перевірити, чи відповідає кінцевий продукт заданим функціональним і нефункціональним вимогам. Зокрема, важливими є критерії стабільності, безпеки та продуктивності кінцевого продукту. Успішне тестування програми дозволяє усунути дефекти до її введення в експлуатацію і зменшити ймовірність виникнення збоїв у майбутньому. Щоб гарантувати високу якість продукту, тестування повинно оцінювати систему всебічно: перевіряти коректність роботи коду, взаємодію з користувачем та поведінку застосунку під навантаженням.

Для ретельного тестування розробленої системи буде використано комплексний підхід і застосовано три методи тестування: модульне тестування, ручне тестування та навантажувальне тестування.

Поєднання запропонованих методів допоможе мінімізувати критичні помилки протягом життєвого циклу програми. Модульне тестування закладає основу, підтверджуючи відсутність помилок у бізнес-логіці на найнижчому рівні, та захищає всю систему від регресійних помилок під час будь-яких майбутніх модифікацій вихідного коду. Метод ручного тестування надає можливість оцінити продукт з точки зору реального користувача.

Під час ручного тестування можна перевірити зручність використання графічного інтерфейсу користувача та виявити неочевидні логічні недоліки, які неможливо передбачити за допомогою автоматизованого тестування. Навантажувальне тестування, в свою чергу, гарантує, що архітектура проєкту має здатність витримувати наплив користувачів та забезпечити при цьому прийнятний час відгуку.

4.2 Модульне тестування

Для забезпечення високої надійності, стабільності та якості програмного забезпечення, в першу чергу в рамках розробки бекенд-частини застосунку було проведено модульне тестування.

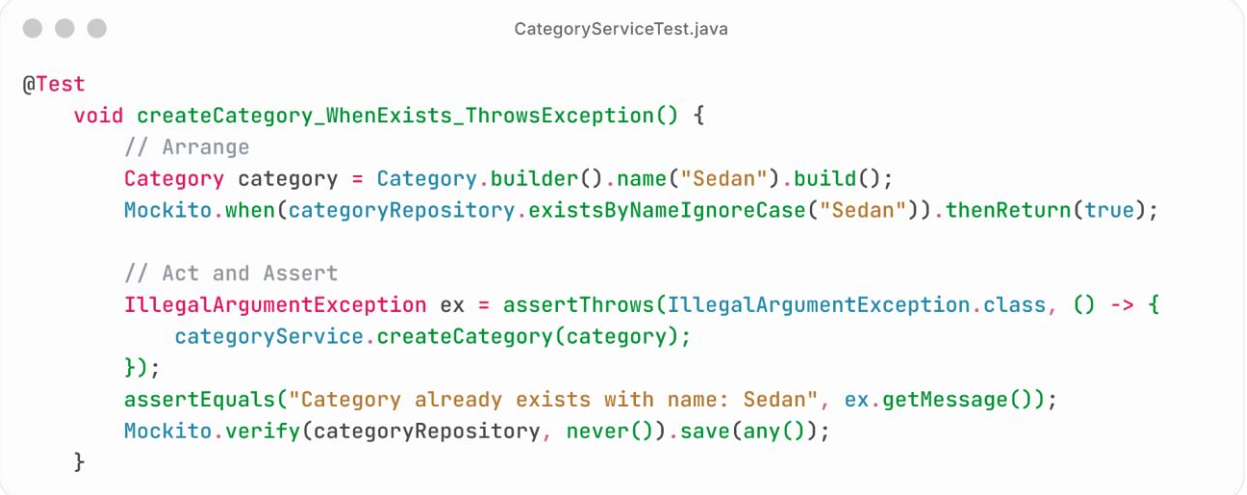
Модульне тестування – це рівень тестування програмного забезпечення, під час якого індивідуальні компоненти програми тестуються в повній ізоляції від решти системи. Метою цього процесу є перевірка того, що кожен окремий метод або клас працює саме так, як було задумано розробником, задовольняючи всі бізнес-вимоги та правильно обробляючи крайні випадки і помилки [21]. Для реалізації модульного тестування у проєкті використовується фреймворк JUnit 5, а для ізоляції компонентів від бази даних та інших зовнішніх залежностей – бібліотека Mockito. Завдяки Mockito створюються так звані “моки” – об’єкти-заглушки, які імітують поведінку реальних класів. Це дозволяє тестувати бізнес-логіку блискавично швидко, без необхідності підіймати реальну базу даних.

Усі написані тести відповідають традиційній методології розробки тестів AAA (Arrange-Act-Assert). Цей підхід структурує кожен тест на три чіткі, логічні блоки, роблячи код тесту читабельним та зрозумілим:

1. Arrange (Підготовка): На цьому етапі створюється оточення для тесту. Ініціалізуються вхідні дані, створюються тестові об’єкти та налаштовується поведінка моків.
2. Act (Дія): Тут відбувається виклик одного конкретного методу системи, який ми зараз тестуємо, з використанням даних, підготовлених на попередньому етапі.
3. Assert (Перевірка): На фінальному етапі ми порівнюємо отриманий результат роботи методу з тим, що очікували отримати. Також тут може перевірятися, чи викликав метод певні залежності (наприклад, чи спробував він зберегти дані в репозиторій).

Чудовим прикладом застосування цього підходу є тестування сервісу

управління категоріями транспортних засобів. Розглянемо тест, який перевіряє, що система не дозволить створити категорію, якщо така назва вже існує в базі даних (див. рис. 4.1).



```

CategoryServiceTest.java

@Test
void createCategory_WhenExists_ThrowsException() {
    // Arrange
    Category category = Category.builder().name("Sedan").build();
    Mockito.when(categoryRepository.existsByNameIgnoreCase("Sedan")).thenReturn(true);

    // Act and Assert
    IllegalArgumentException ex = assertThrows(IllegalArgumentException.class, () -> {
        categoryService.createCategory(category);
    });
    assertEquals("Category already exists with name: Sedan", ex.getMessage());
    Mockito.verify(categoryRepository, never()).save(any());
}

```

Рис. 4.1 Приклад модульного тестування

У цьому прикладі блок Arrange створює об'єкт категорії та програмує мок-репозиторій імітувати ситуацію, коли категорія з такою назвою вже є. У блоках Act та Assert ми викликаємо метод createCategory і стверджуємо, що він обов'язково повинен викинути виняток IllegalArgumentException. Після цього ми додатково переконуємося, що текст помилки коректний, і що програма не дійшла до рядка коду, який зберігає дані (never().save), тим самим захистивши цілісність нашої системи.

Такий підхід покриває всі критичні вузли бізнес-логіки застосунку, гарантуючи, що подальші зміни в коді не зламують вже існуючий та перевірений функціонал.

4.3 Ручне тестування

Незважаючи на наявність автоматизованих модульних тестів, які перевіряють роботу окремих функцій, важливою складовою забезпечення якості є

ручне тестування.

Ручне тестування – це процес перевірки програмного забезпечення, під час якого тестувальник або розробник бере на себе роль звичайного кінцевого користувача. Метою є використання всіх функцій додатку (клікання кнопок, заповнення форм, переходи між сторінками) для виявлення логічних помилок, проблем з інтерфейсом користувача (UI) та користувацьким досвідом (UX), які неможливо або занадто складно покрити автоматизованими скриптами. Ручне тестування дозволяє переконатися, що система в цілому (фронтенд + бекенд + база даних) працює коректно і відповідає початковим вимогам бізнесу.

Для систематизації цього процесу використовуються тест-кейси. Тест-кейс – це детальний опис одного конкретного сценарію перевірки, який включає:

1. Передумови (Preconditions): Що має бути виконано до початку тесту (наприклад, “користувач авторизований”).
2. Кроки (Steps): Послідовні дії, які треба зробити.
3. Очікуваний результат (Expected Result): Як саме має відреагувати система, якщо вона працює правильно.

Нижче наведено приклади ключових тест-кейсів, розроблених для перевірки нашої системи оренди автомобілів.

Тест-кейс 1. Оформлення нового бронювання автомобіля

Мета: Перевірити, що клієнт може успішно орендувати вільний автомобіль.

Передумови:

1. Користувач зареєстрований, авторизований та має підтверджений профіль (завантажені документи).
2. У системі є автомобіль Toyota Camry, який вільний на обрані дати.

Кроки:

1. Відкрити сторінку каталогу автомобілів.
2. Обрати Toyota Camry та перейти на сторінку деталей.
3. Натиснути кнопку “Забронювати цей автомобіль”.
4. У компоненті календаря обрати дати оренди: з завтрашнього дня терміном

на 3 дні.

5. Перевірити, що система автоматично розрахувала правильну загальну вартість (ціна за день помножена на 3).
6. Натиснути кнопку “Створити бронювання”.

Очікуваний результат:

1. Форма успішно відправляє дані.
2. Користувача перенаправляє на сторінку “Мої бронювання” (/reservations), де з’являється новий запис зі статусом “В обробці”.

Тест-кейс 2. Блокування недоступних дат у календарі

Мета: Перевірити роботу логіки перетину дат (захист від подвійного бронювання).

Передумови:

1. Користувач авторизований.
2. Автомобіль “Renault Logan” вже заброньований іншим клієнтом на дати з 15 по 20 число поточного місяця.

Кроки:

1. Відкрити сторінку автомобіля “Renault Logan”.
2. Подивитися на календар вибору дат.
3. Спробувати обрати дати оренди з 18 по 22 число.

Очікуваний результат:

1. Дні з 15 по 20 візуально заблоковані (сірого кольору, перекреслені).
2. Система фізично не дозволяє клікнути на ці дати. Якщо користувач спробує підмінити запит через інструменти розробника, бекенд має відхилити запит з помилкою валідації.

Тест-кейс 3. Обмеження для непідтверджених користувачів

Мета: Перевірити дотримання бізнес-правил щодо верифікації профілю.

Передумови:

1. Користувач авторизований, але його профіль не верифікований (статус isVerified: false).

Кроки:

1. Перейти на сторінку будь-якого автомобіля.
2. Натиснути кнопку “Забронювати цей автомобіль”.
3. У компоненті календаря обрати будь-які доступні дати оренди.

Очікуваний результат:

1. Кнопка “Створити бронювання” недоступна (сірого кольору).
2. Над формою відображається жовтий інформаційний блок з попередженням: “Ви повинні заповнити свій профіль та завантажити необхідні документи, щоб забронювати автомобіль”, та посиланням на сторінку профілю.

4.4 Тестування під навантаженням

Для підтвердження відповідності розробленої системи вимогам щодо продуктивності та надійності було проведено тестування під навантаженням. Головною метою цього етапу є перевірка здатності серверної частини стабільно обробляти інтенсивний потік вхідних HTTP-запитів від заданої кількості віртуальних користувачів безперервно.

Проведення таких тестів дозволяє емпірично виміряти фактичну пропускну здатність застосунку та переконатися у відсутності критичної деградації швидкодії чи відмов в обслуговуванні під час пікових навантажень.

У якості інструменту для генерації запитів використано ПЗ Postman. Першочергово – створено колекцію із запитів, що будуть надсилатися, а саме: отримання списку авто (GET api/vehicles/), отримання списку категорій (GET api/categories/) і перегляд профілю користувача (GET api/users/me/). Далі, використано інструмент Collection Runner у конфігурації Performance для безпосереднього тестування.

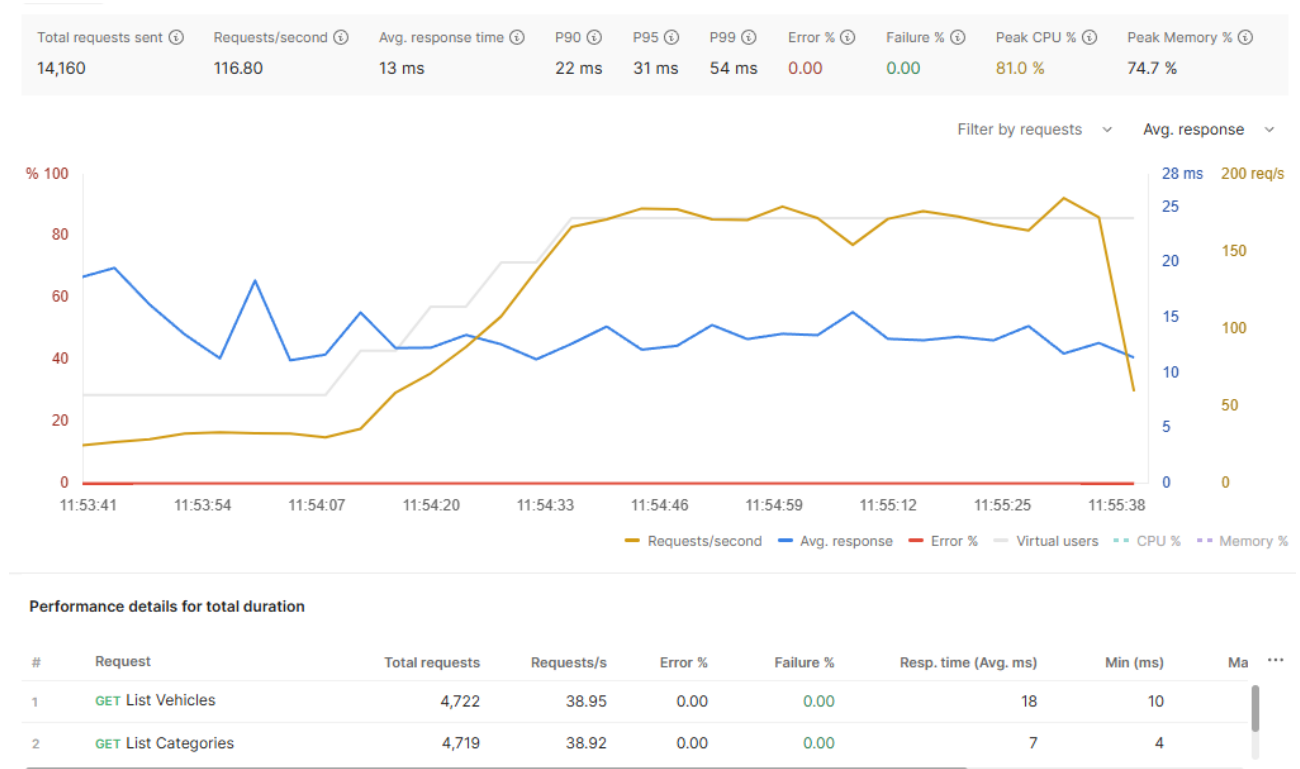


Рис 4.2 Результат тестування під навантаженням

В результаті тесту (див. рис. 4.2) отримали значення кількості запитів, які система може обробити за секунду – 116, а також середній час відповіді на запит – 13мс.

Таким чином, застосунок відповідає визначеним під час проєктування нефункціональним вимогам.

ВИСНОВКИ

Результатом виконаної кваліфікаційної роботи стала веб-орієнтована інформаційна система оренди автомобілів “Wheelix”. Застосунок дозволяє користувачам здійснювати зручний пошук та вибір транспортних засобів, автоматизує процеси оренди, адміністрування, а також гарантує безпечне і надійне збереження даних.

Під час виконання кваліфікаційної роботи виконано такі завдання:

1. Проведено аналіз предметної галузі, виявлено ключові проблеми та потреби користувачів у сфері прокату транспорту, порівняно існуючі програмні рішення, визначено їх обмеження. На основі зібраних даних, розроблено вимоги до програмного забезпечення, зокрема функціональні сценарії користувачів, а також нефункціональні вимоги до надійності та продуктивності.
2. Проведено огляд мов програмування, бібліотек, фреймворків та інших веб-технологій, що можуть бути використані для розробки системи.
3. Спроектовано архітектуру застосунку. Описано поділ системи на серверну та клієнтську частини. Визначено доменні сутності та їх взаємозв'язки, описано структуру бази даних. Створено відповідні UML-діаграми.
4. Програмно реалізовано повноцінний вебзастосунок з використанням сучасного технологічного стеку. Серверну частину побудовано на базі мови Java та фреймворку Spring Boot, а клієнтський інтерфейс – з використанням TypeScript, React та Next.js. Забезпечено збереження даних у реляційній базі PostgreSQL. Крім того, успішно реалізовано інтеграцію із зовнішніми сервісами: підключено Gemini API для забезпечення семантичного пошуку та AWS S3 для безпечного хмарного зберігання фотографій і документів.
5. Проведено комплексне тестування системи. Застосовано методи ручного, модульного та навантажувального тестування. Результати тестів свідчать про коректність роботи застосунку і відповідність функціональним та

нефункціональним вимогам.

Таким чином, поставлені в межах кваліфікаційної роботи завдання успішно виконано. Розроблена інформаційна система “Wheelix” може стати основою для подальшого розвитку та впровадження в реальні бізнес-процеси у сфері оренди транспорту.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Грицюк В.В., Яскевич В.О. Огляд технічних засобів для створення сучасного веб рішення у сфері оренди транспорту. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С.54-55.
2. Грицюк В.В., Яскевич В.О. Впровадження семантичного пошуку для системи оренди авто за допомогою великої мовної моделі. Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 23 квітня 2026 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2026. С.60-63.

ПЕРЕЛІК ПОСИЛАНЬ

1. Transport Trends and Economics 2018–2019 - Mobility as a Service. UNECE.
URL: <https://unece.org/transport/publications/transport-trends-and-economics-2018-2019-mobility-service>.
2. Trends, Challenges Redefine Car Rental Industry. *AutoRentalNews*. 14.01.2026.
URL: <https://www.autorentalnews.com/articles/trends-challenges-redefine-car-rental-industry>.
3. *Ukr-Prokat*. URL: <https://ukr-prokat.com/>.
4. *BLS*. URL: <https://bls.ua/ua>.
5. *Megarent*. URL: <https://megarent.ua/ua/>.
6. *7cars*. URL: <https://7cars.com.ua/ua/>.
7. Functional and Non-Functional Requirements Simply Put!: Simple Requirements Decomposition / Drill-Down Techniques for Defining IT Application ... Qualities. *Createspace Independent Publishing Platform*, 2016. 68 с.
8. Getting started | IntelliJ IDEA. *IntelliJ IDEA Help*.
URL: <https://www.jetbrains.com/help/idea/>.
9. Microsoft. Documentation for Visual Studio Code. *Visual Studio Code - The open source AI code editor | Your home for multi-agent development*.
URL: <https://code.visualstudio.com/docs>.
10. Java Documentation - Get Started. *Oracle Help Center*.
URL: <https://docs.oracle.com/en/java/>.
11. The starting point for learning TypeScript. *TypeScript: JavaScript With Syntax For Types*. URL: <https://www.typescriptlang.org/docs/>.
12. Spring Boot :: Spring Boot. *Spring | Home*. URL: <https://docs.spring.io/spring-boot/>.
13. React. URL: <https://react.dev/>.
14. Vercel. Next.js Docs | Next.js by Vercel - The React Framework.

URL: <https://nextjs.org/docs>.

15. PostgreSQL: Documentation. *PostgreSQL: The world's most advanced open source database*. URL: <https://www.postgresql.org/docs/>.
16. AWS S3 documentation. *Amazon*. URL: <https://docs.aws.amazon.com/s3/>.
17. Git documentation. *Git*. URL: <https://git-scm.com/doc>.
18. Gemini generateContent API | *Google AI for Developers*. URL: <https://ai.google.dev/gemini-api/docs>.
19. Пірог О.В. Безпека вебдодатків : навч. посібн. / О.В. Пірог. – Електронні дані. – Житомир : Житомирська політехніка, 2025. – 290 с. URL: <http://eztuir.ztu.edu.ua/123456789/8813>
20. Яскевич В. О. JavaScript-бібліотека React : навч. посіб. для студентів спеціальності «Комп'ютерні науки». Київ : Київський університет імені Бориса Грінченка, 2023.
21. Якість та тестування інформаційних систем: навч. посібн. / О.А. Золотухіна та ін. Київ: ННІТ ДУТ, 2020. 128с.

ДОДАТОК А. ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ



ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



Розробка веб-орієнтованої інформаційної системи для подового оренди автомобілів

Виконав студент 4 курсу
групи ПД-41
Грицюк Владислав Віталійович
Керівник роботи
кандидат технічних наук, доцент Яскевич Владислав Олександрович
Київ – 2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – покращення якості процесів пошуку, бронювання, обліку та управління послугами прокату авто шляхом проєктування та розробки веб-орієнтованої інформаційної системи.

Об'єкт дослідження – процеси пошуку, бронювання, обліку та управління послугами подового прокату авто.

Предмет дослідження – методи, засоби і технології веб-програмування, що застосовуються для забезпечення процесів пошуку, бронювання, обліку та управління подовим прокатом авто.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Провести аналіз предметної галузі оренди авто та ознайомитись з основними бізнес-процесами у галузі.
2. Ознайомитись з існуючими сервісами оренди, представленими на ринку України, визначити їх переваги та недоліки.
3. Визначити функціональні та нефункціональні вимоги до застосунку.
4. Провести огляд технічних засобів для розробки системи.
5. Спроектувати архітектуру клієнтської та серверної частин системи.
6. Програмно реалізувати вебзастосунок для оренди авто, що забезпечить зручний пошук, бронювання, адміністрування та безпечне зберігання даних.
7. Провести комплексне тестування системи і визначити чи відповідає вона поставленим вимогам.

3

АНАЛІЗ АНАЛОГІВ

	Укр-Прокат	BLS	Megarent	7cars	Wheelix
Перегляд каталогу та онлайн бронювання	+	+	+	+	+
Особистий кабінет з історією бронювання	—	+	—	+	+
Попередня віддалена верифікація документів	—	+	—	—	+
Інтерактивний календар доступності	—	—	—	—	+
Семантичний AI-пошук	—	—	—	—	+
Технологічний стек та архітектура	Застарілі монолітні рішення (CMS, PHP)				Сучасна клієнт-серверна архітектура (React/Next.js + Spring Boot)

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги (клієнт):

1. Реєстрація та вхід.
2. Додавання та редагування особистих даних.
3. Додавання та редагування документів.
4. Перегляд доступних автомобілів і характеристик.
5. Фільтрація авто за заданими характеристиками.
6. Пошук за допомогою запитів природною мовою.
7. Перегляд календаря доступності для кожного авто.
8. Створення заявок на бронювання.

Функціональні вимоги (менеджер):

1. Управління автопарком.
2. Розгляд заявок на бронювання.
3. Перегляд профілів користувачів, в тому числі прикріплених документів.
4. Можливість заблокувати користувача.

Нефункціональні вимоги:

1. Поділ на ролі: клієнт та менеджер.
2. Автентифікація та авторизація за допомогою JWT.
3. Хешування паролів за алгоритмом bcrypt.
4. Зберігання фото документів в файловому сховищі AWS S3, підписані посилання для доступу.
5. Валідація вхідних даних.
6. Час відгуку на стандартні запити – до 2 секунд; обробка 50 запитів на секунду.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



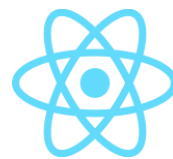
Java



Spring Boot



TypeScript



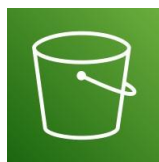
React



Next.js



PostgreSQL



AWS S3



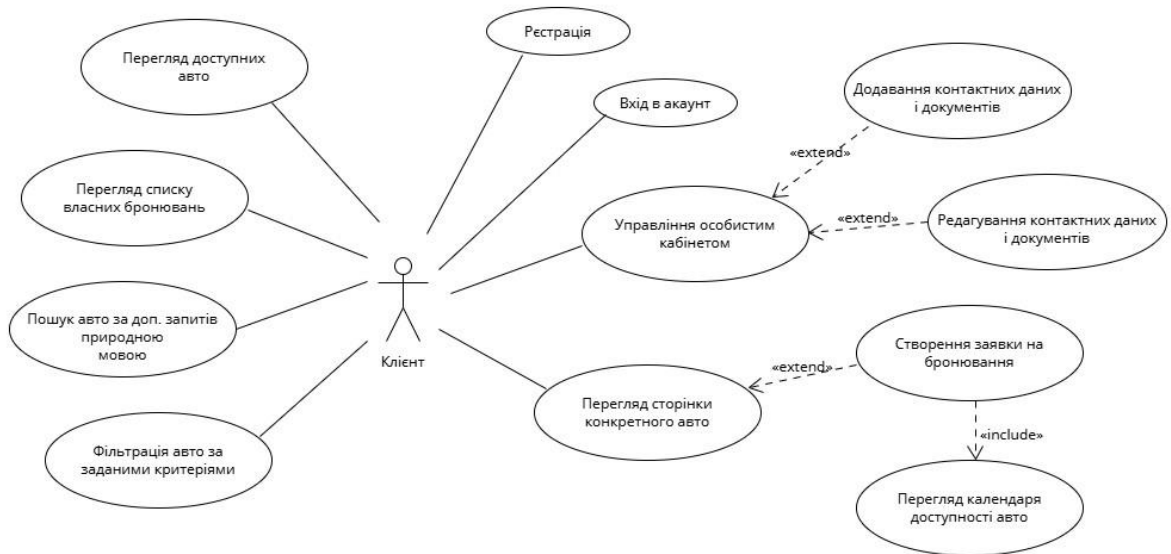
Gemini API



Git

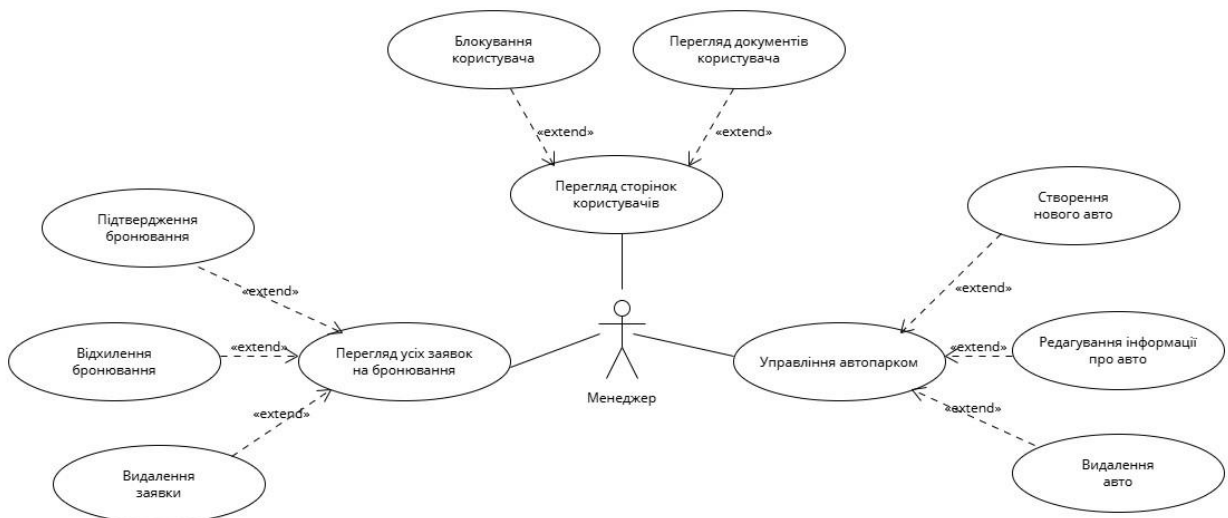
6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ (КЛІЄНТ)



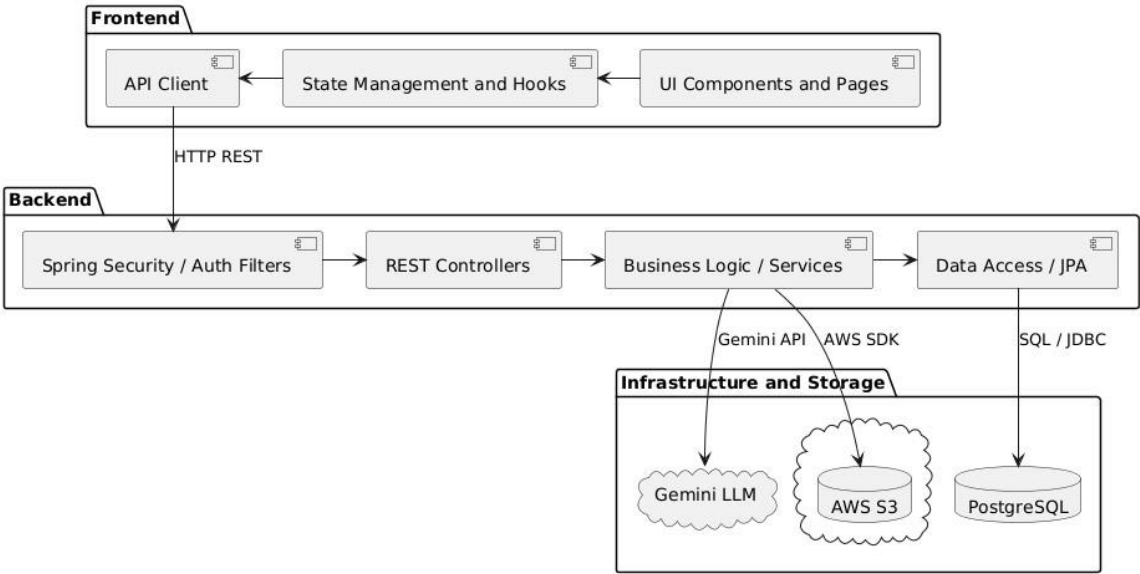
7

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ (МЕНЕДЖЕР)



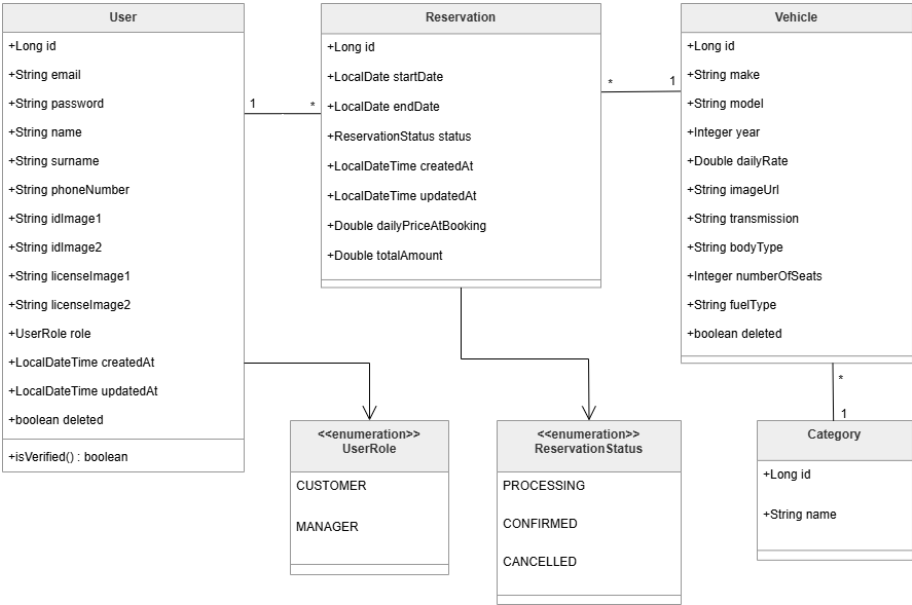
8

ДІАГРАМА КОМПОНЕНТІВ АРХІТЕКТУРИ



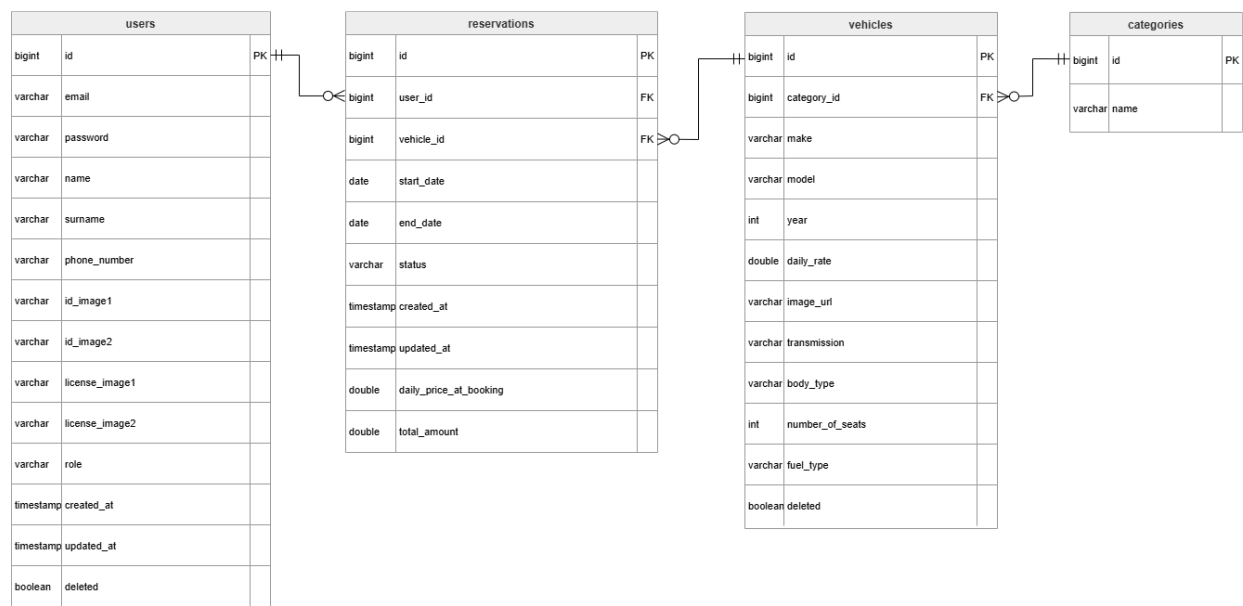
9

ДІАГРАМА КЛАСІВ СУТНОСТЕЙ

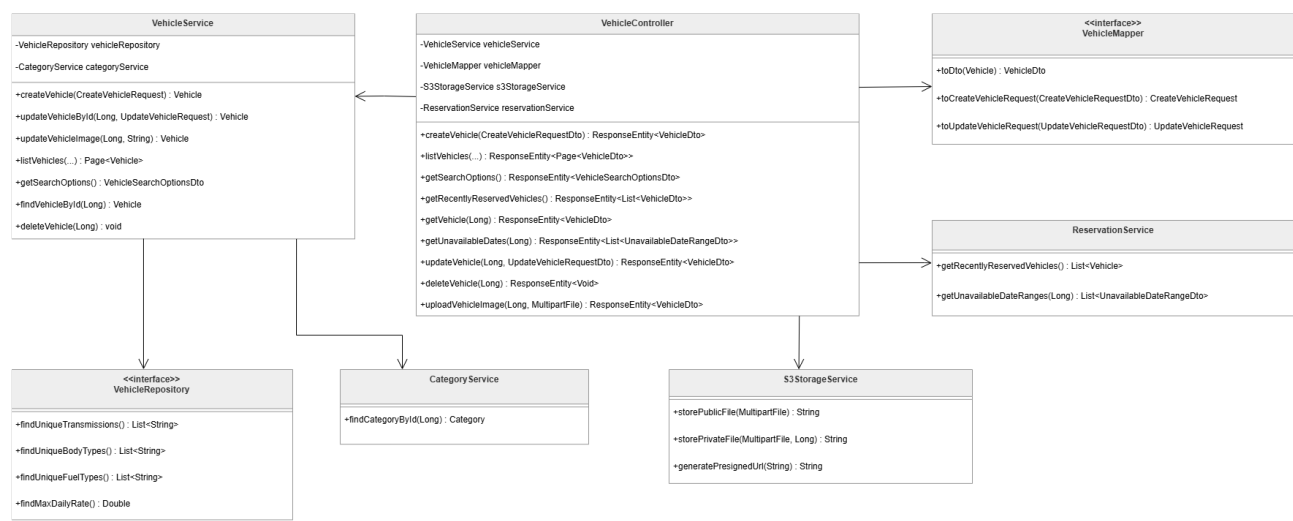


10

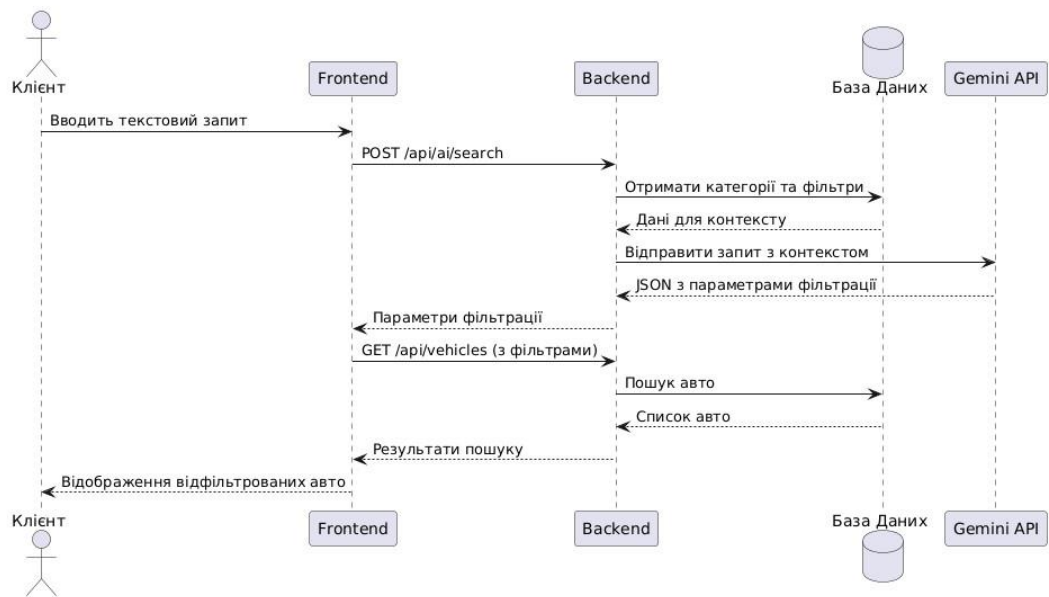
ER-ДІАГРАМА БАЗИ ДАНИХ



ДІАГРАМА КЛАСІВ ЛОГІКИ

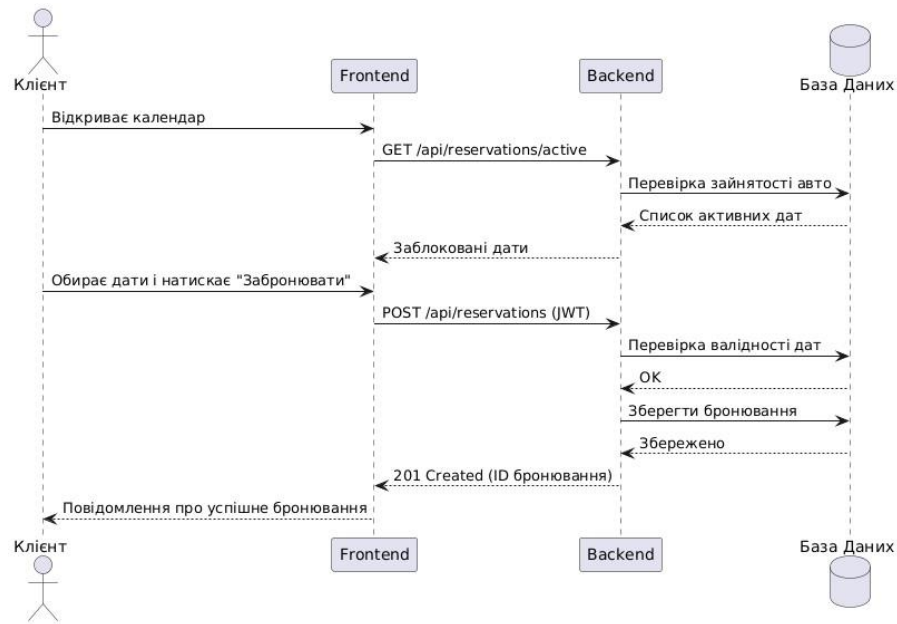


ДІАГРАМА ПОСЛІДОВНОСТІ СЕМАНТИЧНОГО ПОШУКУ



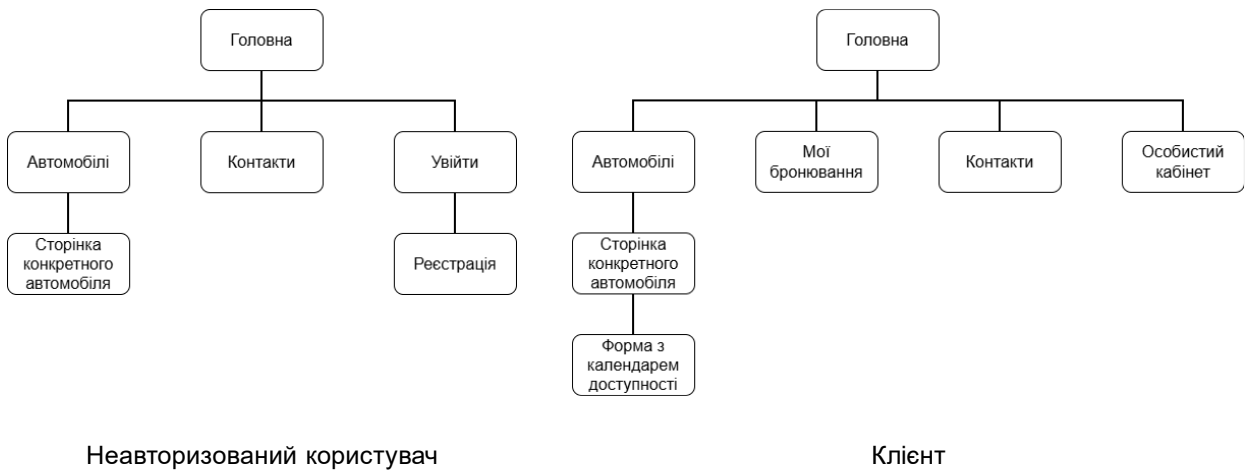
13

ДІАГРАМА ПОСЛІДОВНОСТІ БРОНЮВАННЯ АВТО



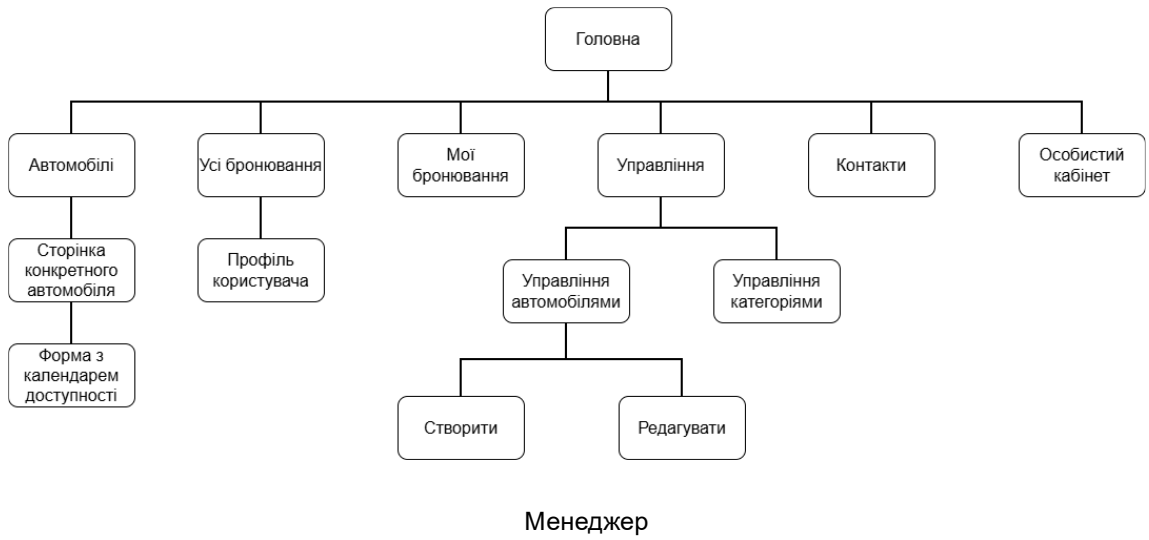
14

МАПА САЙТУ



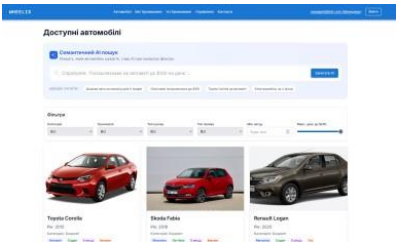
15

МАПА САЙТУ



16

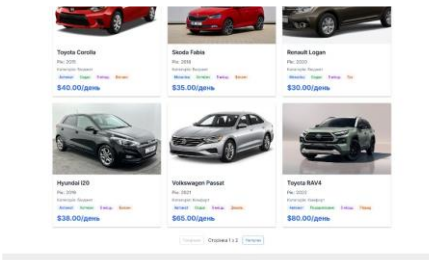
ЕКРАННІ ФОРМИ



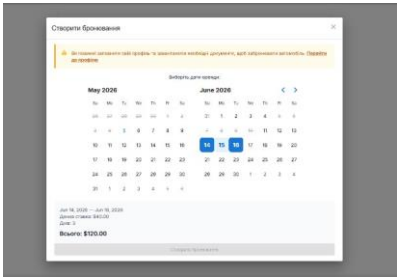
Сторінка каталогу



Сторінка конкретного авто



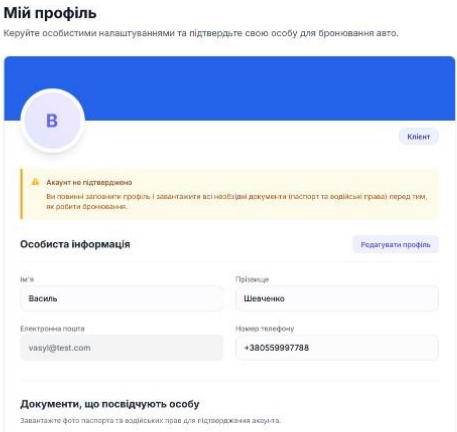
Сторінка каталогу



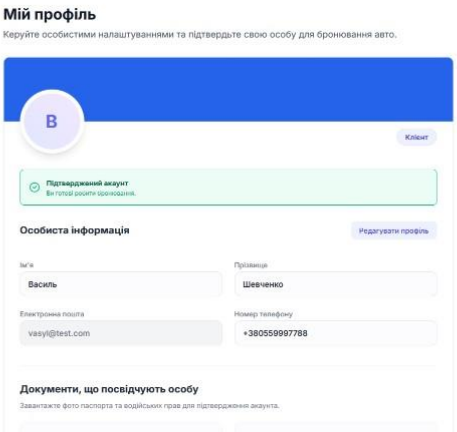
Процес вибору дат при бронюванні

17

ЕКРАННІ ФОРМИ



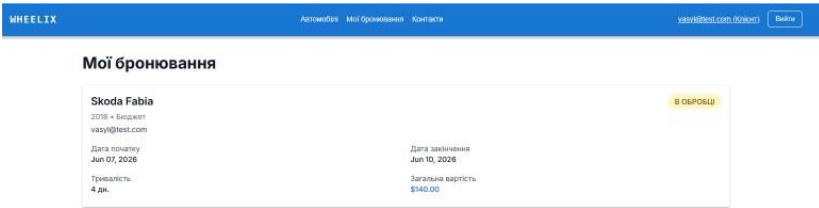
Особистий кабінет (не верифіковано)



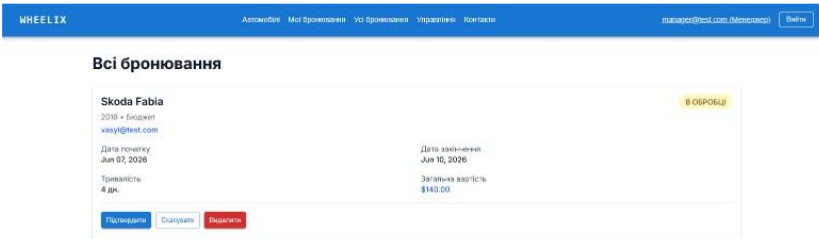
Особистий кабінет (верифіковано)

18

ЕКРАННІ ФОРМИ



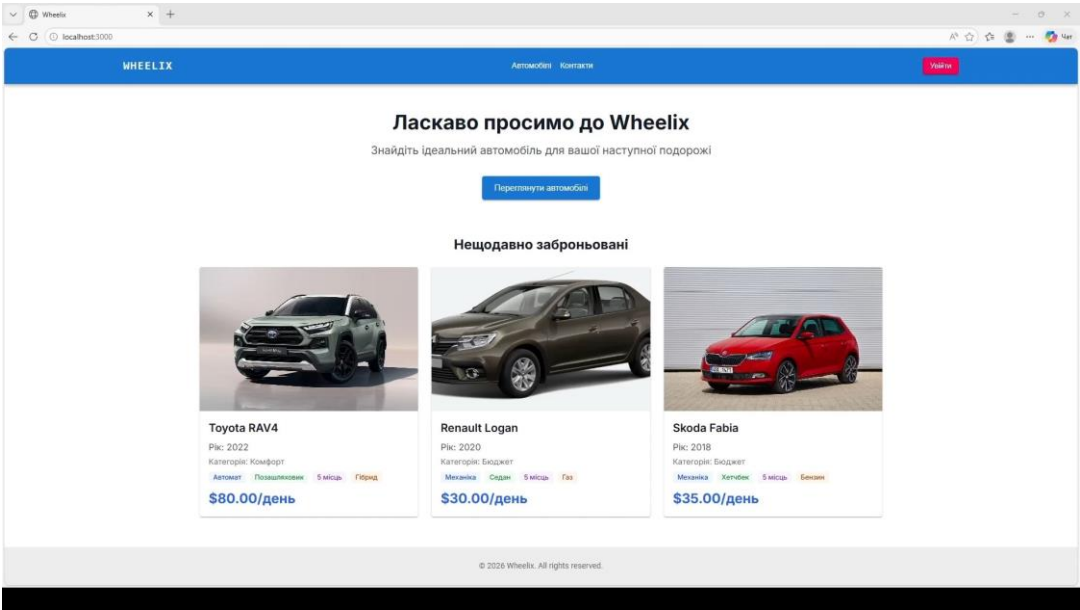
Мої бронювання (клієнт)



Всі бронювання (менеджер)

19

ДЕМОНСТРАЦІЯ РОБОТИ ЗАСТОСУНКУ



20

ТЕСТУВАННЯ ПЗ

1. Модульне тестування
2. Ручне тестування
 - Тест-кейс 1. Оформлення нового бронювання автомобіля
 - Тест-кейс 2. Блокування недоступних дат у календарі
 - Тест-кейс 3. Обмеження для непідтверджених користувачів
3. Навантажувальне тестування
 - 116 запитів на секунду (RPS)
 - Середній час відповіді – 13 мс

21

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Грицюк В.В., Яскевич В.О. Огляд технічних засобів для створення сучасного веб рішення у сфері оренди транспорту. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С. 54-55.
2. Грицюк В.В., Яскевич В.О. Впровадження семантичного пошуку для системи оренди авто за допомогою великої мовної моделі. Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 23 квітня 2026 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2026. С. 60-63.

22

ВИСНОВКИ

1. Досліджено предметну галузь оренди авто, порівняно аналоги: Укр-Прокат, BLS, Megarent, 7cars. На основі аналізу сформовані функціональні та нефункціональні вимоги.
2. Спроектовано архітектуру застосунку. Описано поділ системи на серверну та клієнтську частини. Визначено доменні сутності та їх взаємозв'язки, описано структуру бази даних. Створено відповідні UML-діаграми.
3. Визначено технологічний стек для реалізації системи. Для серверної частини використана мова Java та фреймворк Spring Boot, для клієнтського інтерфейсу – TypeScript, React та Next.js. Збереження даних відбувається у реляційній базі PostgreSQL. Підключено Gemini API для забезпечення семантичного пошуку та AWS S3 для безпечного хмарного зберігання фотографій.
4. Програмно реалізовано вебзастосунок на основі клієнт-серверної архітектури. Впроваджено ключовий функціонал: інтелектуальні механізми пошуку і бронювання автомобілів, створено особистий кабінет користувача, а також необхідні інструменти управління прокатом для адміністраторів.
5. Проведено комплексне тестування системи. Застосовано методи ручного, модульного та навантажувального тестування. Результати тестів свідчать про коректність роботи застосунку і відповідність поставленим вимогам.

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

```

package com.example.rental.controllers;

import
com.example.rental.domain.CreateVehicleRequest;
import
com.example.rental.domain.UpdateVehicleRequest;
import com.example.rental.domain.dtos.*;
import com.example.rental.domain.entities.Vehicle;
import com.example.rental.mappers.VehicleMapper;
import com.example.rental.services.ReservationService;
import com.example.rental.services.VehicleService;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.web.PageableDefault;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import
org.springframework.web.multipart.MultipartFile;
import
org.springframework.web.servlet.support.ServletUriCom
ponentsBuilder;

import java.util.List;

import com.example.rental.services.S3StorageService;

@RestController
@RequestMapping(path = "/api/vehicles")
@RequiredArgsConstructor
public class VehicleController {

    private final VehicleService vehicleService;
    private final VehicleMapper vehicleMapper;
    private final S3StorageService s3StorageService;
    private final ReservationService reservationService;

    @PostMapping
    public ResponseEntity<VehicleDto> createVehicle(
        @Valid @RequestBody
CreateVehicleRequestDto createVehicleRequestDto) {
        CreateVehicleRequest createVehicleRequest =
vehicleMapper.toCreateVehicleRequest(createVehicleRe
questDto);
        Vehicle createdVehicle =
vehicleService.createVehicle(createVehicleRequest);
        return new
ResponseEntity<>(vehicleMapper.toDto(createdVehicle)
, HttpStatus.CREATED);
    }

    @GetMapping
    public ResponseEntity<Page<VehicleDto>>
listVehicles(

```

```

        @RequestParam(name = "categoryId", required =
false) Long categoryId,
        @RequestParam(name = "transmission", required
= false) String transmission,
        @RequestParam(name = "bodyType", required =
false) String bodyType,
        @RequestParam(name = "fuelType", required =
false) String fuelType,
        @RequestParam(name = "minSeats", required =
false) Integer minSeats,
        @RequestParam(name = "maxRate", required =
false) Double maxRate,
        @RequestParam(name = "make", required =
false) String make,
        @RequestParam(name = "model", required =
false) String model,
        @PageableDefault(size = 6) Pageable pageable
    ) {
        Page<VehicleDto> vehicles = vehicleService
.listVehicles(categoryId, transmission,
bodyType, fuelType, minSeats, maxRate, make, model,
pageable)
        .map(vehicleMapper::toDto);
        return ResponseEntity.ok(vehicles);
    }

    @GetMapping(path = "/search-options")
    public ResponseEntity<VehicleSearchOptionsDto>
getSearchOptions() {
        return
ResponseEntity.ok(vehicleService.getSearchOptions());
    }

    @GetMapping(path = "/recently-reserved")
    public ResponseEntity<List<VehicleDto>>
getRecentlyReservedVehicles() {
        List<VehicleDto> vehicles =
reservationService.getRecentlyReservedVehicles()
        .stream()
        .map(vehicleMapper::toDto)
        .toList();
        return ResponseEntity.ok(vehicles);
    }

    @GetMapping(path =("/{id}")
    public ResponseEntity<VehicleDto>
getVehicle(@PathVariable("id") Long id) {
        Vehicle foundVehicle =
vehicleService.findVehicleById(id);
        return
ResponseEntity.ok(vehicleMapper.toDto(foundVehicle))
;
    }

    @GetMapping(path =("/{id}/unavailable-dates")

```

```

    public
    ResponseEntity<List<UnavailableDateRangeDto>>
    getUnavailableDates(@PathVariable("id") Long id) {
        List<UnavailableDateRangeDto> unavailableDates
        = reservationService.getUnavailableDateRanges(id);
        return ResponseEntity.ok(unavailableDates);
    }

    @PutMapping(path =("/{id}")
    public ResponseEntity<VehicleDto> updateVehicle(
        @PathVariable("id") Long id,
        @Valid @RequestBody
    UpdateVehicleRequestDto updateVehicleRequestDto
    ) {
        UpdateVehicleRequest updateVehicleRequest =
        vehicleMapper.toUpdateVehicleRequest(updateVehicle
        RequestDto);
        Vehicle updatedVehicle =
        vehicleService.updateVehicleById(id,
        updateVehicleRequest);
        VehicleDto vehicleDto =
        vehicleMapper.toDto(updatedVehicle);
        return ResponseEntity.ok(vehicleDto);
    }

    @DeleteMapping(path =("/{id}")
    public ResponseEntity<Void>
    deleteVehicle(@PathVariable("id") Long id) {
        vehicleService.deleteVehicle(id);
        return ResponseEntity.noContent().build();
    }

    @PostMapping(path =("/{id}/image")
    public ResponseEntity<VehicleDto>
    uploadVehicleImage(
        @PathVariable("id") Long id,
        @RequestParam("file") MultipartFile file
    ) {
        String objectKey =
        s3StorageService.storePublicFile(file);

        String fileDownloadUri =
        ServletUriComponentsBuilder.fromCurrentContextPath(
        )
        .path("/api/files/public/" + objectKey)
        .toUriString();

        Vehicle updatedVehicle =
        vehicleService.updateVehicleImage(id,
        fileDownloadUri);
        return
        ResponseEntity.ok(vehicleMapper.toDto(updatedVehicle
        ));
    }
}

package com.example.rental.controllers;

import
com.example.rental.domain.CreateReservationRequest;

```

```

import
com.example.rental.domain.dtos.CreateReservationRequ
estDto;
import com.example.rental.domain.dtos.ReservationDto;
import com.example.rental.domain.entities.Reservation;
import com.example.rental.mappers.ReservationMapper;
import com.example.rental.services.ReservationService;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.data.web.PageableDefault;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import
org.springframework.security.core.Authentication;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping(path = "/api/reservations")
@RequiredArgsConstructor
public class ReservationController {

    private final ReservationService reservationService;
    private final ReservationMapper reservationMapper;

    @PostMapping
    public ResponseEntity<ReservationDto>
    createReservation(
        @Valid @RequestBody
    CreateReservationRequestDto requestDto,
    Authentication authentication) {
        String userEmail = authentication.getName();
        CreateReservationRequest request =
        reservationMapper.toCreateReservationRequest(request
        Dto);
        Reservation reservation =
        reservationService.createReservation(request,
        userEmail);
        return new
        ResponseEntity<>(reservationMapper.toDto(reservation)
        , HttpStatus.CREATED);
    }

    @GetMapping
    public ResponseEntity<Page<ReservationDto>>
    listReservations(
        @PageableDefault(size = 5, sort = "id", direction
        = Sort.Direction.DESC) Pageable pageable) {
        Page<ReservationDto> reservationDtos =
        reservationService
        .listReservations(pageable)
        .map(reservationMapper::toDto);
        return ResponseEntity.ok(reservationDtos);
    }

    @GetMapping(path = "/my")
    public ResponseEntity<Page<ReservationDto>>
    listMyReservations(

```

```

        Authentication authentication,
        @PageableDefault(size = 5, sort = "id", direction
= Sort.Direction.DESC) Pageable pageable) {
    String userEmail = authentication.getName();
    Page<ReservationDto> reservationDtos =
reservationService
        .listReservationsByUserEmail(userEmail,
pageable)
        .map(reservationMapper::toDto);
    return ResponseEntity.ok(reservationDtos);
}

@GetMapping(path =("/{id}")
public ResponseEntity<ReservationDto>
getReservation(@PathVariable("id") Long id) {
    Reservation reservation =
reservationService.findReservationById(id);
    return
ResponseEntity.ok(reservationMapper.toDto(reservation
));
}

@PatchMapping(path =("/{id}/confirm")
public ResponseEntity<ReservationDto>
confirmReservation(@PathVariable("id") Long id) {
    Reservation confirmedReservation =
reservationService.confirmReservationById(id);
    return
ResponseEntity.ok(reservationMapper.toDto(confirmed
Reservation));
}

@PatchMapping(path =("/{id}/cancel")
public ResponseEntity<ReservationDto>
cancelReservation(@PathVariable("id") Long id) {
    Reservation cancelledReservation =
reservationService.cancelReservationById(id);
    return
ResponseEntity.ok(reservationMapper.toDto(cancelledR
eservation));
}

@DeleteMapping(path =("/{id}")
public ResponseEntity<Void>
deleteReservation(@PathVariable("id") Long id) {
    reservationService.deleteReservationById(id);
    return ResponseEntity.noContent().build();
}
}

package com.example.rental.services;

import
com.example.rental.domain.CreateReservationRequest;
import com.example.rental.domain.entities.Reservation;
import
com.example.rental.domain.enums.ReservationStatus;
import
com.example.rental.repositories.ReservationRepository;
import jakarta.persistence.EntityNotFoundException;
import jakarta.transaction.Transactional;
import lombok.RequiredArgsConstructor;

```

```

import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.stereotype.Service;

import java.time.LocalDate;
import java.time.temporal.ChronoUnit;
import java.util.List;

@Service
@RequiredArgsConstructor
public class ReservationService {

    private final ReservationRepository
reservationRepository;
    private final UserService userService;
    private final VehicleService vehicleService;

    @Transactional
    public Reservation
createReservation(CreateReservationRequest request,
String email) {
        LocalDate startDate = request.getStartDate();
        LocalDate endDate = request.getEndDate();

        if (endDate.isBefore(startDate)) {
            throw new
IllegalArgumentException("Reservation end date must
not be before start date");
        }
        if (!isVehicleAvailable(request.getVehicleId(),
startDate, endDate)) {
            throw new IllegalArgumentException("Vehicle is
already reserved for the selected dates.");
        }

        var user = userService.findUserByEmail(email);
        if (!user.isVerified()) {
            throw new IllegalArgumentException("User is
not verified. Please complete your profile.");
        }

        var vehicle =
vehicleService.findVehicleById(request.getVehicleId());
        if (vehicle.isDeleted()) {
            throw new IllegalArgumentException("Cannot
reserve a deleted vehicle.");
        }
        Reservation newReservation = new Reservation();
        newReservation.setUser(user);
        newReservation.setVehicle(vehicle);
        newReservation.setStartDate(request.getStartDate()
);
        newReservation.setEndDate(request.getEndDate());
        newReservation.setStatus(ReservationStatus.PROC
ESSING);

        newReservation.setDailyPriceAtBooking(vehicle.ge
tDailyRate());
        long days =
ChronoUnit.DAYS.between(request.getStartDate(),
request.getEndDate()) + 1;

```

```

        newReservation.setTotalAmount(vehicle.getDailyRate() * days);

        return reservationRepository.save(newReservation);
    }

    public Page<Reservation> listReservations(Pageable pageable) {
        return reservationRepository.findAll(pageable);
    }

    public Page<Reservation>
listReservationsByUserEmail(String email, Pageable
pageable) {
        return
reservationRepository.findAllByUserEmail(email,
pageable);
    }

    public Reservation findReservationById(Long id) {
        return reservationRepository.findById(id)
            .orElseThrow(() -> new
EntityNotFoundException("Reservation not found with
id " + id));
    }

    public void deleteReservationById(Long id) {
        reservationRepository.deleteById(id);
    }

    public Reservation confirmReservationById(Long id)
    {
        Reservation reservation =
reservationRepository.findById(id)
            .orElseThrow(() -> new
EntityNotFoundException("Reservation not found with
id " + id));
        reservation.setStatus(ReservationStatus.CONFIRMED);
        return reservationRepository.save(reservation);
    }

    public Reservation cancelReservationById(Long id) {
        Reservation reservation =
reservationRepository.findById(id)
            .orElseThrow(() -> new
EntityNotFoundException("Reservation not found with
id " + id));

        if (reservation.getStatus() ==
ReservationStatus.CONFIRMED) {
            throw new IllegalArgumentException("Cannot
cancel a confirmed reservation.");
        }
        if (reservation.getStatus() ==
ReservationStatus.CANCELLED) {
            throw new
IllegalArgumentException("Reservation is already
cancelled.");
        }
        if
(LocalDate.now().isAfter(reservation.getEndDate())) {

```

```

            throw new IllegalArgumentException("Cannot
cancel a reservation that has already ended.");
        }

        reservation.setStatus(ReservationStatus.CANCELLED);
        return reservationRepository.save(reservation);
    }

    public boolean isVehicleAvailable(Long vehicleId,
LocalDate startDate, LocalDate endDate) {
        List<Reservation> conflicts =
reservationRepository.findConflictingReservations(vehicleId,
startDate, endDate);
        return conflicts.isEmpty();
    }

    public
List<com.example.rental.domain.dtos.UnavailableDateRangeDto>
getUnavailableDateRanges(Long vehicleId) {
        List<ReservationStatus> activeStatuses =
List.of(ReservationStatus.PROCESSING,
ReservationStatus.CONFIRMED);
        List<Reservation> reservations =
reservationRepository.findAllByVehicleIdAndStatusIn(vehicleId,
activeStatuses);
        return reservations.stream()
            .map(r -> new
com.example.rental.domain.dtos.UnavailableDateRangeDto(r.getStartDate(), r.getEndDate()))
            .toList();
    }

    public
List<com.example.rental.domain.entities.Vehicle>
getRecentlyReservedVehicles() {
        return
reservationRepository.findTop10ByOrderByIdDesc()
            .stream()
            .map(Reservation::getVehicle)
            .filter(v -> !v.isDeleted())
            .distinct()
            .limit(3)
            .toList();
    }
}

'use client';

import React, { useState } from 'react';
import { ProtectedRoute } from
'@/components/layout/ProtectedRoute';
import { useMyReservations } from
'@/lib/hooks/useApi';
import { ReservationList } from
'@/components/reservations/ReservationList';
import { Pagination } from
'@/components/common/Pagination';

export default function ReservationsPage() {
    const [page, setPage] = useState<number>(0);

```

```

const { data: reservationsPage, isLoading, error } =
useMyReservations({ page });

return (
  <ProtectedRoute>
    <div className="max-w-7xl mx-auto px-4 sm:px-6
lg:px-8 py-8">
      <h1 className="text-3xl font-bold text-gray-900
mb-6">Мої бронювання</h1>
      <ReservationList
        reservations={reservationsPage?.content}
        isLoading={isLoading}
        error={error}
      />
      {reservationsPage && (
        <Pagination
          currentPage={reservationsPage.number}
          totalPages={reservationsPage.totalPages}
          onPageChange={setPage}
        />
      )}
    </div>
  </ProtectedRoute>
);
}

```

```

import React from 'react';
import Link from 'next/link';
import { ReservationDto } from
'@/lib/types/reservation';
import { Card } from '../common/Card';
import { formatDate, calculateTotalCost } from
'@/lib/utills/date';
import { Button } from '../common/Button';
import { useAuth } from '@/lib/context/AuthContext';

```

```

interface ReservationCardProps {
  reservation: ReservationDto;
  showActions?: boolean;
  onConfirm?: () => void;
  onCancel?: () => void;
  onDelete?: () => void;
}

```

```

export const ReservationCard:
React.FC<ReservationCardProps> = ({
  reservation,
  showActions = false,
  onConfirm,
  onCancel,
  onDelete,
}) => {
  const { user } = useAuth();
  const totalCost = reservation.totalAmount ??
calculateTotalCost(
  reservation.vehicle.dailyRate,
  reservation.startDate,
  reservation.endDate
);
  const days = Math.ceil(
    (new Date(reservation.endDate).getTime() - new
Date(reservation.startDate).getTime()) /

```

```

    (1000 * 60 * 60 * 24)
  ) + 1;
  const isPastEndDate = new Date() > new
Date(reservation.endDate);

return (
  <Card>
    <div className="space-y-4">
      <div className="flex justify-between items-start">
        <div>
          <Link
            href={` /vehicles/${reservation.vehicle.id}`}>
            <h3 className="text-xl font-semibold text-
gray-900 hover:text-blue-600">
              {reservation.vehicle.make}
              {reservation.vehicle.model}
            </h3>
          </Link>
          <p className="text-sm text-gray-500 mt-1">
            {reservation.vehicle.year} •
            {reservation.vehicle.category.name}
          </p>
          {user?.role === 'MANAGER' ? (
            <Link
              href={` /admin/users/${reservation.user.id}`}>
              <p className="text-sm text-blue-600
hover:underline mt-1 font-medium">
                {reservation.user.email}
              </p>
            </Link>
          ) : (
            <p className="text-sm text-gray-700 mt-1">
              {reservation.user.email}
            </p>
          )}
        </div>
        <span
          className={`px-3 py-1 rounded-full text-sm
font-medium ${reservation.status === 'CONFIRMED'
? 'bg-green-100 text-green-800'
: reservation.status === 'PROCESSING'
? 'bg-yellow-100 text-yellow-800'
: 'bg-gray-100 text-gray-800'
}`}>
          >
            {reservation.status === 'CONFIRMED' ?
'ПІДТВЕРДЖЕНО' : reservation.status ===
'PROCESSING' ? 'В ОБРОБЦІ' : 'СКАСОВАНО'}
          </span>
        </div>

        <div className="grid grid-cols-2 gap-4 text-sm">
          <div>
            <p className="text-gray-600">Дата
початку</p>
            <p className="font-
medium">{formatDate(reservation.startDate)}</p>
          </div>
          <div>
            <p className="text-gray-600">Дата
закінчення</p>

```

```

        <p className="font-
medium">{formatDate(reservation.endDate)}</p>
    </div>
    <div>
        <p className="text-gray-600">Тривалість</p>
        <p className="font-medium">{days} дн.</p>
    </div>
    <div>
        <p className="text-gray-600">Загальна
вартість</p>
        <p className="font-medium text-blue-
600">${totalCost.toFixed(2)}</p>
    </div>
</div>

{showActions && (
    <div className="flex space-x-2 pt-4 border-t">
        {reservation.status === 'PROCESSING' &&
!reservation.user.isDeleted && onConfirm && (
            <Button variant="primary" size="sm"
onClick={onConfirm}>
                Підтвердити

```

```

        </Button>
    )}
    {reservation.status === 'PROCESSING' &&
!isPastEndDate && onCancel && (
        <Button variant="outline" size="sm"
onClick={onCancel}>
            Скасувати
        </Button>
    )}
    {onDelete && (
        <Button variant="danger" size="sm"
onClick={onDelete}>
            Видалити
        </Button>
    )}
</div>
)}
</div>
</Card>
);
};

```