НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	**Завідувач кафедри комп'ютерних наук**
_____ **Ігор БОЛБОТ**	_____ **Белла ГОЛУБ**
(підпис)	(підпис)
«___» _____ 2026 р.	«___» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Інформаційна система обліку оренди автомобілів фізичних осіб»

Спеціальність «122 Комп'ютерні науки»

Освітньо-професійна програма «Комп'ютерні науки»

Гарант освітньої програми д.е.н., професор	_____ (підпис)	**Роман РУДЕНСЬКИЙ**
Керівник бакалаврської кваліфікаційної роботи асистент	_____ (підпис)	**Тетяна БАРАНОВА**
Консультант бакалаврської кваліфікаційної роботи к.т.н., доцент	_____ (підпис)	**Олексій ТКАЧЕНКО**
Виконав	_____ (підпис)	**Сергій МУЛЯРЧУК**

Київ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____ Белла ГОЛУБ
(підпис)

«___» _____ 2026 р.

ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ
Мулярчуку Сергію Сергійовичу

Спеціальність «122 Комп'ютерні науки»

Освітньо-професійна програма «Комп'ютерні науки»

Тема бакалаврської кваліфікаційної роботи

«**Інформаційна система обліку оренди автомобілів фізичних осіб**»
затверджена наказом від «06» січня 2026 р. № 46 «С»
Термін подання завершеної роботи на кафедру «22» травня 2026 р.

Вихідні дані до бакалаврської кваліфікаційної роботи (проєкту): Технічне завдання на розробку клієнт-серверної інформаційної системи обліку оренди автомобілів фізичних осіб.
Перелік питань, що підлягають дослідженню:
Аналіз предметної галузі та існуючих рішень. 2) Проєктування архітектури, БД та модулів. 3) Розробка серверної і клієнтської частин. 4) Інтеграція зовнішніх сервісів. 5) Тестування та рекомендації щодо впровадження**.**
Дата видачі завдання «06» січня 2026

Керівник бакалаврської кваліфікаційної роботи _____ **Тетяна БАРАНОВА**
(підпис)
асистент

Консультант бакалаврської кваліфікаційної роботи _____ **Олексій ТКАЧЕНКО**
(підпис)
к.т.н., доцент

Виконав _____ **Сергій МУЛЯРЧУК**
(підпис)

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Позначення	Пояснення
API	Application Programming Interface, програмний інтерфейс взаємодії компонентів
REST	Архітектурний стиль побудови веб-API
JWT	JSON Web Token, токен для автентифікації та авторизації
ORM	Object-Relational Mapping, відображення об'єктів програми на таблиці бази даних
SQL	Structured Query Language, мова запитів до реляційних баз даних
СУБД	Система управління базами даних
UML	Unified Modeling Language, мова графічного моделювання
ER	Entity-Relationship, модель сутностей і зв'язків
UI	User Interface, користувацький інтерфейс
OTP	One-Time Password, одноразовий код підтвердження
S3	Об'єктне сховище для збереження файлів і медіаданих

ВСТУП

Цифрові сервіси поступово змінюють підхід до користування транспортом. Для дедалі більшої кількості людей автомобіль перестає бути обов'язковим об'єктом власності і стає послугою, яку можна отримати на потрібний період. Це створює потребу в інформаційних системах, здатних об'єднати власників транспортних засобів, орендарів та адміністраторів у спільному цифровому середовищі.

Процес оренди між фізичними особами включає не лише передачу автомобіля, а й низку супровідних операцій: реєстрацію та верифікацію сторін, опис транспортного засобу, узгодження дати й місця передачі, формування заявки, розрахунок вартості, оплату, відстеження статусу поїздки, повернення автомобіля і збір відгуків. Якщо ці кроки виконуються вручну або через різні інструменти, виникають помилки в даних, ускладнюється контроль оплати і зростає ризик конфліктів між учасниками.

Актуальність роботи полягає в необхідності створення не просто електронного каталогу автомобілів, а системи, яка забезпечує повний облік процесів оренди: учасників, транспортних засобів, бронювань, платежів, статусів поїздок, результатів верифікації, відгуків і звітних показників. Саме тому в роботі розглядається повноцінна клієнт-серверна інформаційна система, а не окремий вебсайт.

Метою бакалаврської кваліфікаційної роботи є проектування та розробка інформаційної системи обліку оренди автомобілів фізичних осіб, яка автоматизує основні бізнес-процеси взаємодії власників автомобілів, орендарів та адміністраторів.

Для досягнення поставленої мети у роботі виконано: аналіз предметної галузі та огляд існуючих аналогів, визначення ролей і інформаційних потоків, проектування архітектури й схеми бази даних, програмну реалізацію основних модулів, підключення зовнішніх сервісів і тестування ключових сценаріїв.

Об'єктом дослідження є процеси обліку оренди автомобілів фізичних осіб. Предметом дослідження — архітектурні рішення, моделі даних, алгоритми та програмні засоби, що забезпечують автоматизацію цих процесів. Методи дослідження включають системний аналіз предметної галузі, порівняльний аналіз аналогів, UML-моделювання, реляційне проектування бази даних, проектування REST API, модульне та інтеграційне тестування [20, 21]. У реалізації використано клієнт-серверний стек: FastAPI для серверної частини, Next.js і React для веб-інтерфейсу, PostgreSQL для зберігання даних та Docker для відтворюваного розгортання. Платежі, SMS-підтвердження, геолокацію і зберігання медіафайлів реалізовано через зовнішні сервіси; обґрунтування вибору технологій подано у другому розділі.

Практичне значення роботи полягає в тому, що розроблена система може бути використана як основа для сервісу оренди автомобілів або як внутрішня система обліку для організації, що працює з автопарком та заявками користувачів. Обрані технології дають можливість у подальшому розширювати функціональність системи, інтегрувати нові платіжні сервіси та розгортати її у контейнеризованому середовищі. Структуру пояснювальної записки складено відповідно до методичних рекомендацій до бакалаврської кваліфікаційної роботи [1].

Апробація результатів не проводилась. Розроблену систему заплановано як основу для подальшого розвитку сервісу оренди автомобілів між фізичними особами.

Бакалаврська кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг записки — 93 сторінки; кількість використаних джерел — 25, додатків — 3. У першому розділі виконано аналіз предметної галузі, сформульовано постановку задачі та визначено вимоги до системи. У другому розділі спроектовано архітектуру, базу даних і функціональні модулі. Третій розділ описує реалізацію серверної та клієнтської частин. У четвертому розділі наведено результати тестування та рекомендації щодо впровадження.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка завдання

Оренда автомобілів між фізичними особами пов'язана з тим, що власник передає свій автомобіль іншій людині на певний час [2]. На перший погляд така оренда схожа на звичайний автопрокат, але різниця помітна вже в правилах. У прокатної компанії є власний автопарк, типовий договір, фіксовані тарифи і більш-менш однаковий порядок видачі машин. З приватними власниками все менш однаково. Один може здавати автомобіль тільки на вихідні, інший погоджується привезти його в потрібне місце, ще хтось просить більшу заставу або ставить обмеження щодо стажу водія.

Тут важливо враховувати інтереси обох сторін. Власник, звісно, хоче отримувати дохід від автомобіля, який не використовується постійно. Але разом з цим йому потрібно розуміти, кому він передає машину, чи підтверджені документи орендаря, чи проведена оплата і на яких умовах автомобіль має бути повернений. Для орендаря ситуація трохи інша. Йому важливо без зайвих дзвінків знайти підходящий автомобіль, побачити його фото, зрозуміти кінцеву ціну і не отримати відмову вже після домовленості.

Через це система не може бути просто дошкою оголошень. Якщо на сайті є тільки список автомобілів і контакти власника, більша частина процесу все одно залишається “вручну”. Тоді оплата, підтвердження, перевірка документів і статус поїздки можуть зберігатися в різних місцях. Це незручно і для власника, і для орендаря.

У процесі оренди є багато умов, які треба врахувати заздалегідь. Наприклад, орендар повинен мати водійське посвідчення, відповідний вік і стаж. Також мають значення місце знаходження автомобіля, мінімальний строк оренди, можливість доставки, застава, правила скасування, комісія сервісу та

статус оплати. Якщо такі речі не внести в логіку системи, швидко з'являться помилки. Наприклад, користувач може подати заявку на автомобіль, який у ці дати вже зайнятий, або перейти до оплати до завершення перевірки.

Сам процес можна подати не одним записом у базі, а кількома станами. Спочатку орендар знаходить автомобіль і переглядає інформацію про нього. Потім створює заявку на потрібні дати. Після цього перевіряється доступність, рахується вартість, а власник підтверджує або відхиляє бронювання. Якщо все погоджено, виконується оплата. Далі починається фактичне користування автомобілем, а після повернення поїздка закривається. Уже після цього сторони можуть залишити відгук.

Для цієї роботи важливо розділити кілька понять. Заявка — це перше звернення орендаря щодо конкретного автомобіля. Бронювання — це заявка, яку підтверджено і для якої вже визначено суму та статус. Поїздка стосується фактичного періоду користування автомобілем. А оренда в цілому охоплює весь шлях: від пошуку машини до завершення розрахунків і закриття поїздки.

Без окремої інформаційної системи такі процеси часто ведуться через месенджери, телефонні дзвінки, таблиці або оголошення. Для одного-двох випадків це ще може працювати. Але коли заявок стає більше, починаються типові проблеми: хтось не оновив доступність автомобіля, хтось не зафіксував оплату, десь загубилась домовленість про час повернення. У результаті немає одного місця, де можна швидко перевірити всі дані [3].

Тому основна проблема цієї предметної області полягає не тільки в пошуку автомобіля. Потрібен нормальний облік усього процесу. Власник має бачити свої автомобілі, заявки, платежі, майбутні поїздки та відгуки. Орендарю потрібен зрозумілий пошук, перевірена ціна, оплата і статус замовлення. Адміністратор повинен мати доступ до перевірки користувачів, документів, спірних ситуацій і звітів. Саме під ці задачі і проектується інформаційна система обліку оренди автомобілів фізичних осіб.

Виявлені проблеми предметної області, їхні наслідки та відповідні вимоги до інформаційної системи систематизовано у таблиці 1.1.

Таблиця 1.1

Проблеми предметної області та вимоги до інформаційної системи

Проблема предметної області	**Наслідок**	**Вимога до системи**
Дані про авто та заявки ведуться у різних каналах	Втрата або дублювання інформації	Єдина база автомобілів, заявок і користувачів
Доступність авто перевіряється вручну	Перетини бронювань і конфлікти дат	Автоматична перевірка зайнятості автомобіля
Розрахунок вартості виконується неформально	Непрозорість ціни для орендаря	Алгоритм розрахунку вартості, комісій і застави
Оплата не пов'язана зі статусом поїздки	Складність фінансового контролю	Інтеграція платіжного сервісу та облік платежів
Перевірка водія не стандартизована	Підвищені ризики для власника	Верифікація профілю та водійських даних
Немає єдиної історії взаємодій	Складно розглядати претензії	Зберігання статусів, відгуків і службових подій

Об'єктом дослідження є процеси обліку оренди автомобілів фізичних осіб. Предметом — методи, моделі та програмні засоби для побудови клієнт-серверної системи, що автоматизує ці процеси.

Мета бакалаврської кваліфікаційної роботи — спроєктувати та розробити інформаційну систему «Інформаційна система обліку оренди автомобілів фізичних осіб», яка впорядковує основні операції між власниками авто, орендарями та адміністратором.

Для цього у роботі спочатку розглянуто саму предметну область — хто бере участь, які дані виникають у процесі, де частіше з'являються проблеми. Паралельно проаналізовано кілька аналогів. Після цього сформовано вимоги та підготовлено основу для проєктування архітектури, бази даних і програмної частини.

Система розрахована на три категорії користувачів. Власник авто публікує оголошення, завантажує фото, задає умови і підтверджує або відхиляє заявки. Орендар веде пошук, обирає дати, проходить перевірку і платить. Адміністратор перевіряє акаунти і розбирає проблемні ситуації.

Функціональні вимоги охоплюють роботу з акаунтами (реєстрація, автентифікація, ролі, верифікація), управління автомобілями (оголошення, фото, пошук, доступність) і власне процес оренди — заявку, розрахунок ціни, зміну статусів, оплату і відгуки.

З технічного боку система має працювати за клієнт-серверною архітектурою з реляційною базою даних, захищеною автентифікацією та адаптивним інтерфейсом. Важливо також передбачити обробку помилок і підтримку контейнеризованого розгортання [4]. Критичні сценарії краще покрити тестами — перевірку доступності авто, розрахунок вартості, бронювання і платіж.

Нестандартні ситуації у подібних сервісах виникають постійно. Бронювання вже недоступного авто, неправильні дати, проблема з платежем, відсутній документ при верифікації. До цього додаються речі, які взагалі не залежать від користувача: SMS може не дійти, зовнішній платіжний сервіс може повернути помилку. Важливо, щоб система не просто "падала" в таких випадках, а показувала зрозуміле повідомлення і водночас зберігала інформацію про збій для адміністратора.

Безпека тут теж важлива, особливо з урахуванням оплат і персональних даних. Паролі хешуються, доступ обмежується токенами і ролями, адміністративні функції закриті. Секретні ключі не мають потрапляти в клієнтський код. Платіжні операції краще не обробляти самостійно — для цього є спеціалізовані провайдери.

Щодо зручності: орендарю потрібен нормальний пошук з фільтрами, зрозуміла ціна і підказки щодо верифікації. Власнику достатньо бачити заявки і

швидко приймати рішення. Адміністраторський інтерфейс — не для краси, а для перевірки даних і пошуку проблем.

Загалом завдання роботи — система, у якій дії користувачів залишаються в даних. Не в чатах і не в таблицях, а в структурованих записах, за якими можна відстежити заявку, оплату і статус поїздки.

1.2 Огляд інформаційних джерел та існуючих рішень

Під час підготовки роботи джерела розділилися на дві групи — не формально, а за тим, навіщо вони використовувалися. Частина допомагала розібратися в самій предметній галузі: як працює peer-to-peer оренда, чим вона відрізняється від car-sharing, що потрібно від платформи для верифікації та оплат [2, 5, 15]. Інша частина — суто технічна, для роботи: як організувати API, яку базу даних використовувати, як запустити все в контейнері [3, 6, 8, 14, 20-24].

Під час аналізу важливіше за перелік функцій було зрозуміти, як ці функції пов'язані між собою. Пошук формує параметри запиту, бронювання створює запис із статусом, оплата додає фінансовий документ, підтвердження змінює стан. Все це — один ланцюжок, і аналоги оцінювалися саме з цього погляду.

При порівнянні дивився на конкретні речі: чи можна здати особистий автомобіль, чи є кабінет власника, як реалізований пошук, чи є онлайн-оплата, верифікація, карта і відгуки. Також важило, чи реалістично відтворити подібний підхід у рамках навчального проєкту [15-17].

Частину інформації вдалося отримати безпосередньо з матеріалів самих сервісів — правила, описи тарифів, окремі огляди ринку. Саме через них стало зрозуміліше, як у подібних платформах організовано процес від пошуку автомобіля до його повернення.

Технічна документація FastAPI, SQLAlchemy, PostgreSQL, Next.js, React, Stripe, Twilio, Mapbox і Docker [6–14] — це вже робочий інструментарій. До неї зверталися переважно під час реалізації окремих функцій, коли потрібно було уточнити деталі: структуру запитів, параметри розгортання або поведінку конкретного методу.

Для аналізу існуючих рішень обрано Turo, PROTO та Getmancar [15–17]. Усі три представляють різні підходи. У Turo власник самостійно додає автомобіль і задає умови, а сервіс бере на себе бронювання, оплату і відгуки. PROTO і Getmancar влаштовані інакше: автомобілі належать компанії, і користувач отримує доступ до них через застосунок без прямої взаємодії з власником.

Turo серед цих трьох — найближчий до потреб цієї роботи [15]. Сервіс дає власнику змогу вказати ціну, умови, доступність і місце видачі. У ньому досить детальні картки автомобілів і продумана логіка бронювання. Проте переносити цю модель напряму не вийде: Turo орієнтований на американський ринок зі своєю правовою базою, яка до українських реалій не адаптована.

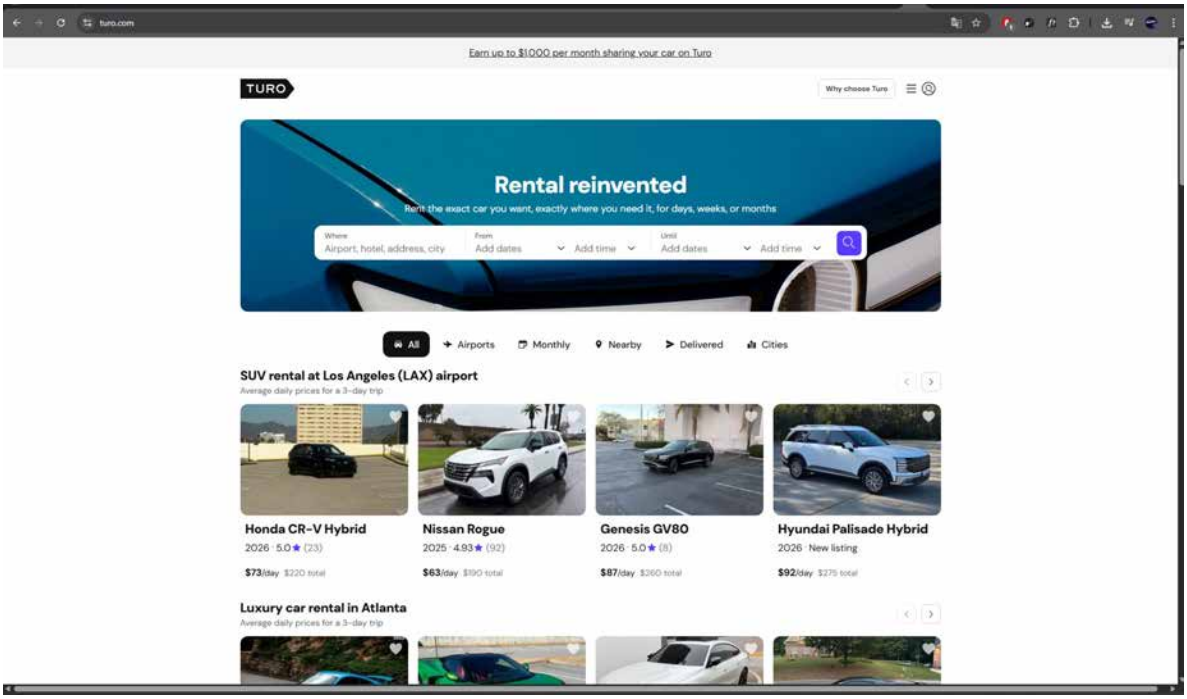

Рис. 1.1 Головна сторінка та картка автомобіля платформи Turo

PORTO демонструє інший підхід: користувач орендує автомобіль не в окремого власника, а з автопарку компанії [16]. Основна перевага такого сервісу полягає у швидкості доступу: автомобіль можна знайти на карті, забронювати, відкрити через мобільний застосунок і оплатити за фактичний час користування. Водночас ця модель не вирішує задачу обліку автомобілів фізичних осіб, оскільки в ній відсутній кабінет власника й механізм публікації приватного автомобіля.

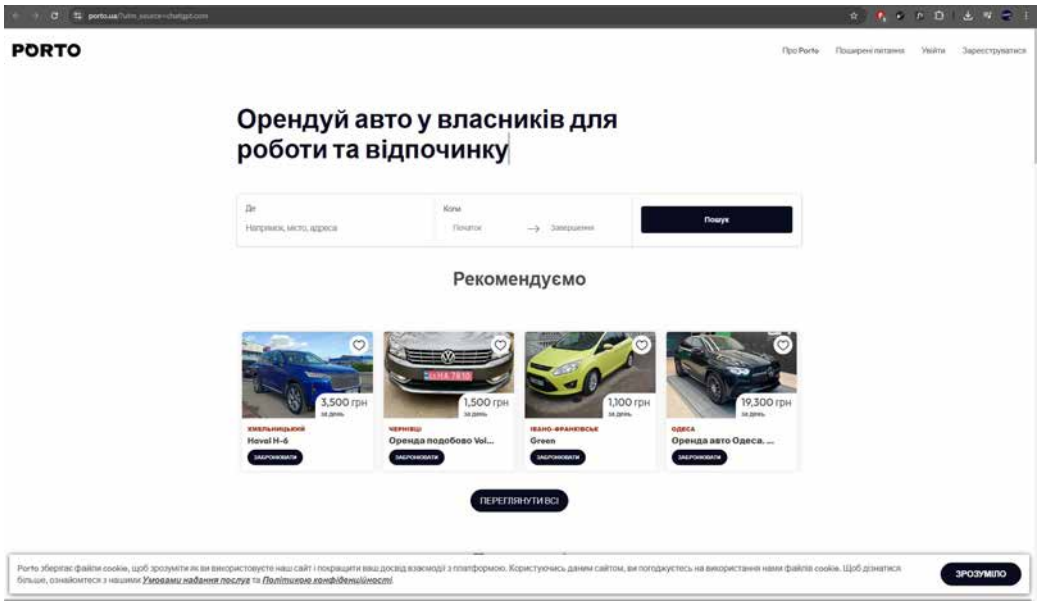

Рис. 1.2 Інтерфейс пошуку автомобіля у мобільному додатку PORTO

Getmancar є українським прикладом сервісу короткострокової оренди автомобілів [17]. Його перевагами є локалізація, робота з українськими користувачами, оплата у гривні та звичний формат карти доступних автомобілів. Для поточної дипломної роботи Getmancar корисний як джерело інтерфейсних і бізнес-ідей. Однак сервіс також не є платформою оренди автомобілів фізичних осіб, оскільки не підтримує повну роль приватного власника.

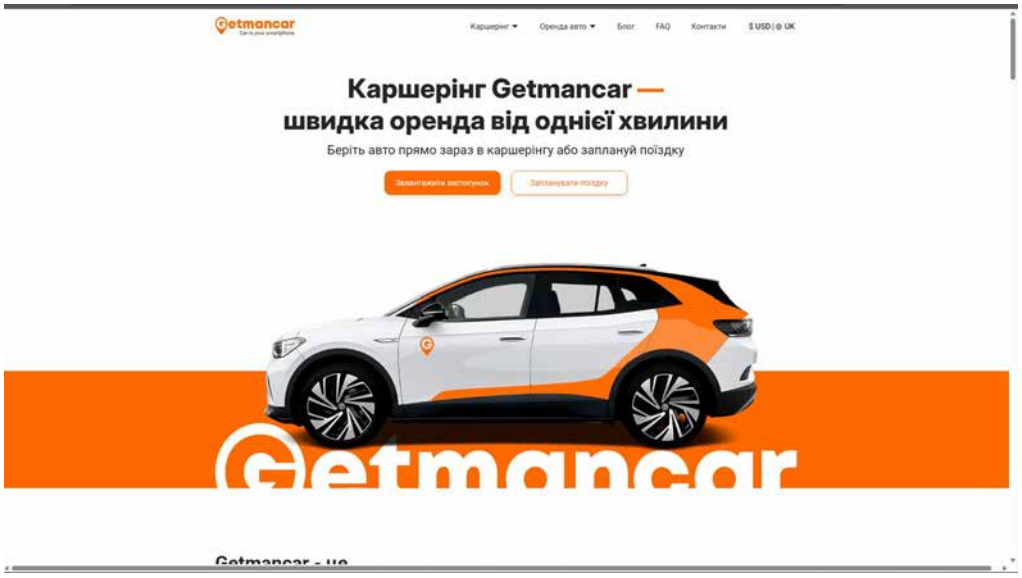

Рис. 1.3 Головна сторінка та карта автомобілів сервісу Getmancar

Результати порівняльного аналізу функціональних можливостей розглянутих рішень наведено у таблиці 1.2.

Таблиця 1.2

Порівняння існуючих рішень із розроблюваною інформаційною системою

Критерій	**Turo**	**PORTO**	**Getmancar**	**Розроблювана**
Модель оренди авто фіз осіб	Так	Ні	Ні	Так
Кабінет власника автомобіля	Так	Ні	Ні	Так
Українська локалізація	Ні	Так	Так	Так
Пошук за датами та параметрами авто	Так	Частково	Частково	Так
Онлайн-оплата	Так	Так	Так	Так
Верифікація користувача	Так	Так	Так	Так
Звітність для власника	Так	Ні	Ні	Так

Порівняльний аналіз показує, що існуючі сервіси мають корисні функціональні можливості, але не покривають повністю задачу дипломного проекту. Turo найближчий до предметної області, однак не адаптований до локальних умов і не може бути використаний як власна система обліку. PROTO і Getmancar добре демонструють зручність короткострокової оренди через карту та мобільний інтерфейс, але їхня бізнес-модель не передбачає участі фізичних осіб як власників автомобілів.

Розроблювана система має поєднати корисні риси розглянутих рішень: каталог автомобілів, пошук, бронювання, оплату, верифікацію, відгуки та

зручний інтерфейс. Водночас для локального сценарію потрібні функції, яких бракує аналогам: окремий кабінет власника, облік заявок і оплат, звітність, українська мова та можливість подальшого розширення відповідно до задач дипломного проекту.

За результатами огляду власну систему доцільно розглядати як більш вузько сфокусоване рішення, ніж великі комерційні платформи. Її мета — реалізувати облік оренди автомобілів фізичних осіб: хто надав автомобіль, хто його орендував, на який період, за якою ціною, у якому статусі перебуває заявка, чи пройдено верифікацію, чи виконано оплату та які результати залишилися після завершення поїздки.

1.3 Моделювання предметної області

Моделювання предметної області необхідне для переходу від текстового опису проблеми до формалізованого подання системи. На цьому етапі визначаються межі інформаційної системи, основні учасники, зовнішні сервіси, бізнес-процеси та інформаційні потоки [18]. У першому розділі використано узагальнений бізнес-процес, контекстну модель та діаграму прецедентів.

На концептуальному рівні предметну область можна описати через набір основних сутностей: користувач, орендар, власник автомобіля, автомобіль, оголошення, бронювання, платіж, відгук, документ верифікації, локація та звіт. Користувач є базовою сутністю для автентифікації. Орендар і власник відображають різні ролі у процесі. Автомобіль є об'єктом оренди, а бронювання — центральною подією, яка пов'язує користувача, автомобіль, період, ціну, оплату та статус виконання.

З точки зору даних бронювання не існує ізольовано. Для нього потрібні зв'язки з автомобілем, власником, орендарем, тарифом, розрахованою вартістю, обраною локацією, платіжним записом і статусом. Така структура дає змогу відповідати на практичні питання: які автомобілі зайняті у певний період, які заявки очікують підтвердження, які платежі не завершено, які поїздки вже закрито, які користувачі потребують перевірки.

Першою моделлю є узагальнений бізнес-процес оренди автомобіля. Він показує послідовність ключових етапів: роботу з каталогом, створення бронювання, оплату, виконання поїздки, повернення автомобіля, формування відгуків і звітності. У процесі беруть участь орендар, власник автомобіля та адміністратор. Орендар ініціює пошук і бронювання, власник підтверджує або відхиляє заявку, адміністратор контролює коректність даних і вирішує спірні ситуації.

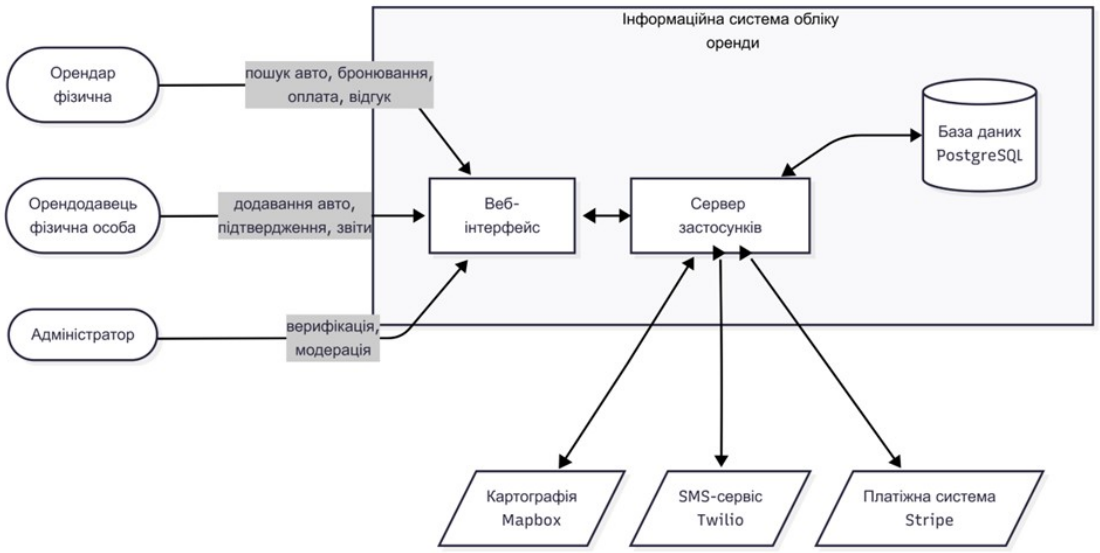

Рис. 1.4 Узагальнений бізнес-процес оренди автомобілів фізичних осіб

На рисунку 1.4 система розглядається як єдиний функціональний блок, який забезпечує послідовне проходження заявки через основні стани. Така модель корисна для першого розділу, оскільки вона не перевантажує опис технічними деталями, але показує, які процеси потребують автоматизації. На її основі надалі можуть бути побудовані діаграми діяльності для окремих алгоритмів: пошуку автомобіля, створення бронювання, розрахунку ціни, формування фінансового звіту та PDF-квитанції.

Другою моделлю є контекстна модель інформаційної системи. Вона визначає межі системи та показує зовнішні сутності, з якими система взаємодіє. До зовнішніх користувачів належать орендар, власник автомобіля та адміністратор. До зовнішніх сервісів належать платіжна система Stripe, SMS-

сервіс Twilio, картографічний сервіс Mapbox, об'єктне сховище AWS S3 та база даних PostgreSQL. У межах клієнт-серверної архітектури frontend відповідає за взаємодію з користувачем, backend — за бізнес-логіку, а база даних — за збереження структурованої інформації.

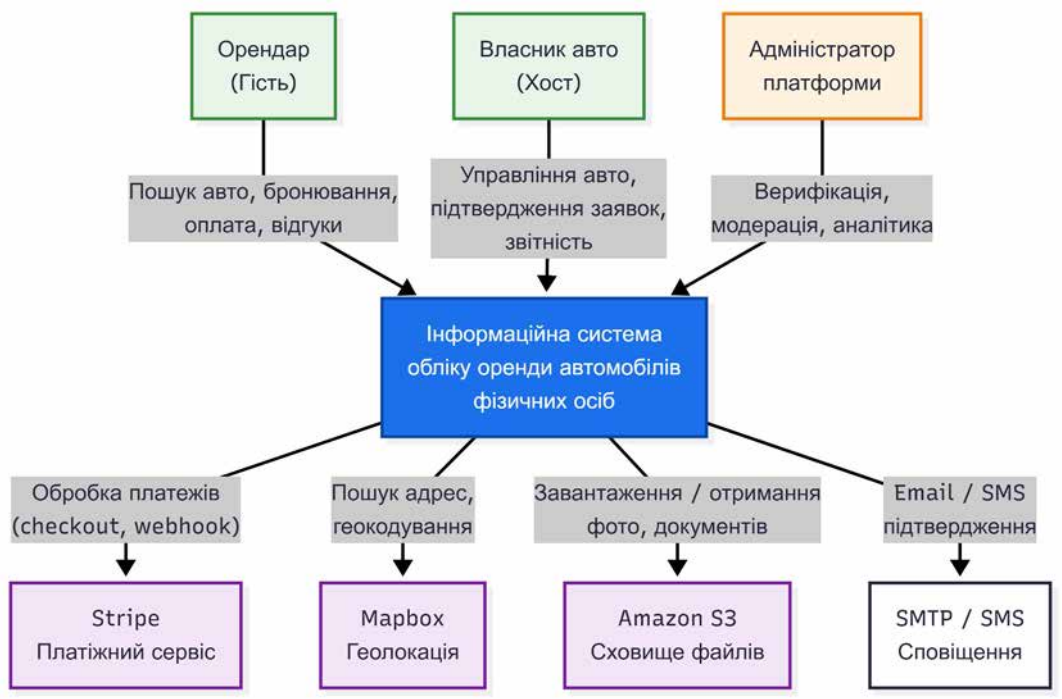

Рис. 1.5 Контекстна модель інформаційної системи обліку оренди автомобілів фіз. осіб

Контекстна модель дозволяє визначити основні інформаційні потоки. Орендар передає системі дані пошуку, персональні дані, інформацію для верифікації, заявку на бронювання та дані оплати. Власник передає дані про автомобіль, фотографії, умови доступності, рішення щодо заявки та відомості про видачу або повернення. Адміністратор отримує доступ до службових даних, статусів, претензій і звітів. Зовнішні сервіси забезпечують платіжні операції, SMS-підтвердження, геокодування адрес і зберігання медіафайлів.

Для уточнення взаємодії користувачів із системою побудовано діаграму прецедентів. Вона показує, які функції доступні орендарю, власнику автомобіля та адміністратору. Така діаграма є доречною у першому розділі, оскільки вона

не описує внутрішню реалізацію, а фіксує очікувану функціональність системи з погляду користувачів.

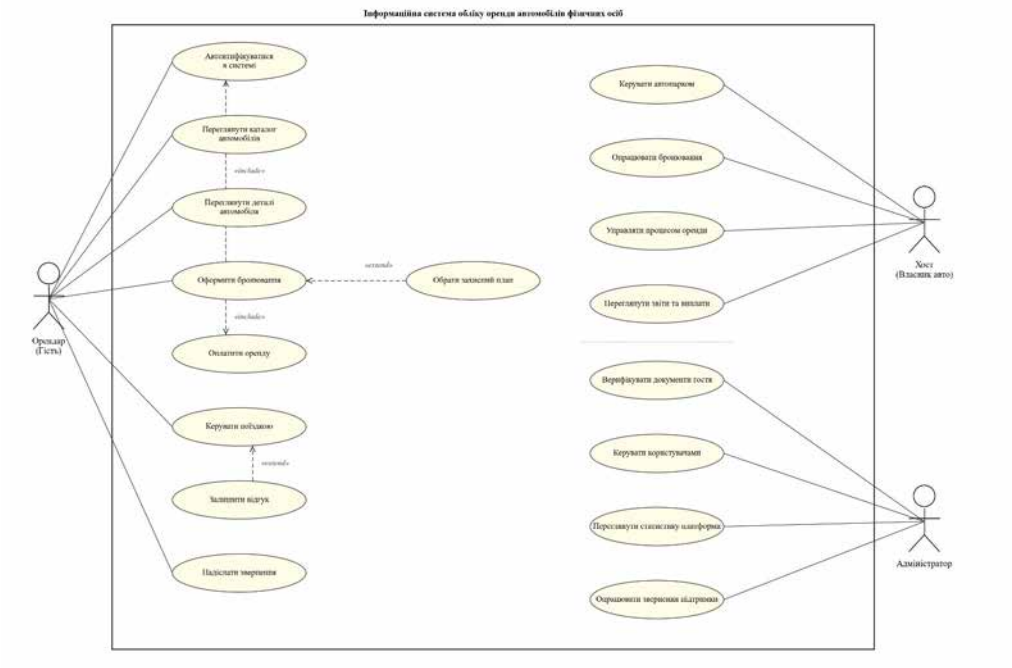

Рис. 1.6 Діаграма прецедентів інформаційної системи

Орендар на діаграмі взаємодіє з прецедентами реєстрації, перегляду каталогу, фільтрації автомобілів, перегляду картки, створення бронювання, оплати та залишення відгуку. Власник автомобіля додає транспортний засіб, керує оголошенням, підтверджує або відхиляє заявки та переглядає звіти. Адміністратор перевіряє користувачів, модерує дані, розглядає спірні ситуації та має доступ до звітної інформації.

Основні інформаційні потоки між учасниками системи та зовнішніми сервісами наведено у таблиці 1.3.

Таблиця 1.3

Основні інформаційні потоки предметної області

Джерело даних	Передані дані	Призначення у системі
Орендар	Параметри пошуку, дати, профіль, документи, оплата	Пошук, бронювання, верифікація, оплата
Власник автомобіля	Характеристики авто, фото, тариф, доступність, рішення щодо заявки	Формування каталогу та керування орендою
Адміністратор	Рішення модерації, перевірки, службові статуси	Контроль якості даних і вирішення спірних ситуацій
Stripe	Статуси платежів, ідентифікатори транзакцій	Облік оплат і застави
Twilio	OTP-коди та статуси доставки	Підтвердження контактних даних
Mapbox	Адреси, координати, результати пошуку місця	Пошук і відображення локацій
PostgreSQL	Структуровані записи системи	Збереження користувачів, авто, заявок, платежів

Склад концептуальних сутностей предметної області наведено у таблиці 1.4.

Таблиця 1.4

Концептуальні сутності предметної області

Сутність	Зміст	Роль у предметній області
Користувач	Обліковий запис, контакти	Базова ідентифікація
Власник автомобіля	Профіль орендодавця, налаштування, статистика	Керує автомобілями та заявками
Орендар	Профіль водія, статус верифікації	Створює бронювання та виконує оплату
Автомобіль	Характеристики, фото, тариф, статус	Основний об'єкт оренди
Бронювання	Період, ціна, статус, зв'язки з авто й користувачами	Центральна бізнес-операція
Платіж	Сума, статус, ідентифікатор	Фінансовий облік оренди

	провайдера	
Відгук	Оцінка та текст після завершення	Формування довіри між учасниками

Концептуальна модель сутностей є підготовчим етапом до побудови ER-діаграми у другому розділі. У першому розділі достатньо визначити склад сутностей та їхнє призначення, оскільки детальна структура полів, ключів, зв'язків і обмежень належить до етапу проектування бази даних. Такий розподіл матеріалу дозволяє уникнути дублювання: системний аналіз пояснює, які дані потрібні, а проектування описує, як саме вони зберігатимуться.

Побудовані моделі відповідають задачам першого розділу: вони описують предметну область на рівні користувачів, процесів і зовнішніх взаємодій. При цьому вони не дублюють матеріал наступних розділів. Діаграми послідовності, діяльності, ER-діаграма, діаграма класів, компонентна діаграма, діаграма пакетів і діаграма розгортання належать до етапу детального проектування. У поточному дипломному проекті частина таких матеріалів уже підготовлена в межах лабораторних робіт: package/deployment-діаграми, алгоритмічні блок-схеми пошуку, бронювання, розрахунку ціни та звітності.

Побудовані моделі підтверджують доцільність обраної клієнт-серверної архітектури. Система має чітко визначені межі, три групи користувачів, набір зовнішніх сервісів та послідовний бізнес-процес. Це дає змогу перейти до проектування бази даних, структури серверних модулів, клієнтських інтерфейсів та алгоритмів обробки заявок у наступних розділах бакалаврської роботи.

Висновки до розділу 1

Перший розділ дав змогу перейти від загальної ідеї сервісу оренди до формалізованої предметної області. Основна проблема полягає не лише у відсутності зручного каталогу автомобілів, а й у відсутності єдиного обліку заявок, оплат, статусів, документів, відгуків і звітності.

Порівняння Turo, PROTO і Getmancar показало, що кожен із сервісів має корисні рішення, однак жоден не закриває повністю локальний сценарій оренди автомобілів фізичних осіб. Для розроблюваної системи важливо поєднати каталог, пошук, бронювання, оплату, верифікацію, кабінет власника, звітність і українську локалізацію.

Побудовані моделі — бізнес-процес, контекстна модель і діаграма прецедентів — фіксують межі системи, ролі учасників та основні інформаційні потоки. Цього достатньо для переходу до другого розділу, де ці висновки уточнюються вже на рівні архітектури, бази даних і програмних компонентів.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Другий розділ присвячено проєктним рішенням для інформаційної системи обліку оренди автомобілів фізичних осіб. Після визначення процесів і вимог у першому розділі далі розглядаються архітектура, модель даних, функціональні модулі, API та розгортання.

Опис побудовано від загального до конкретного. Спочатку подано загальну схему взаємодії frontend, backend, бази даних і зовнішніх сервісів. Далі розглянуто логічну структуру бази даних, модулі backend, сценарій створення бронювання та набір технологій, який реально використовується у програмній частині проекту.

2.1 Проєктування архітектури системи

Система проєктується як клієнт-серверний веб-застосунок. У поточній реалізації клієнтська частина винесена в Next.js-застосунок, а серверна частина реалізована як FastAPI API. Такий поділ відповідає принципам побудови інформаційних систем, у яких інтерфейс, бізнес-логіка та рівень даних мають різні зони відповідальності [3, 6, 9, 20, 23].

Архітектура системи включає три основні рівні: клієнтський рівень, серверний рівень та рівень даних. Окремо виділено зовнішні сервіси, які не належать до внутрішнього коду системи, але забезпечують важливі функції: прийом платежів, SMS-підтвердження, геолокацію, зберігання медіафайлів та надсилання електронних повідомлень [11-14]. Загальну архітектуру інформаційної системи наведено на рисунку 2.1.

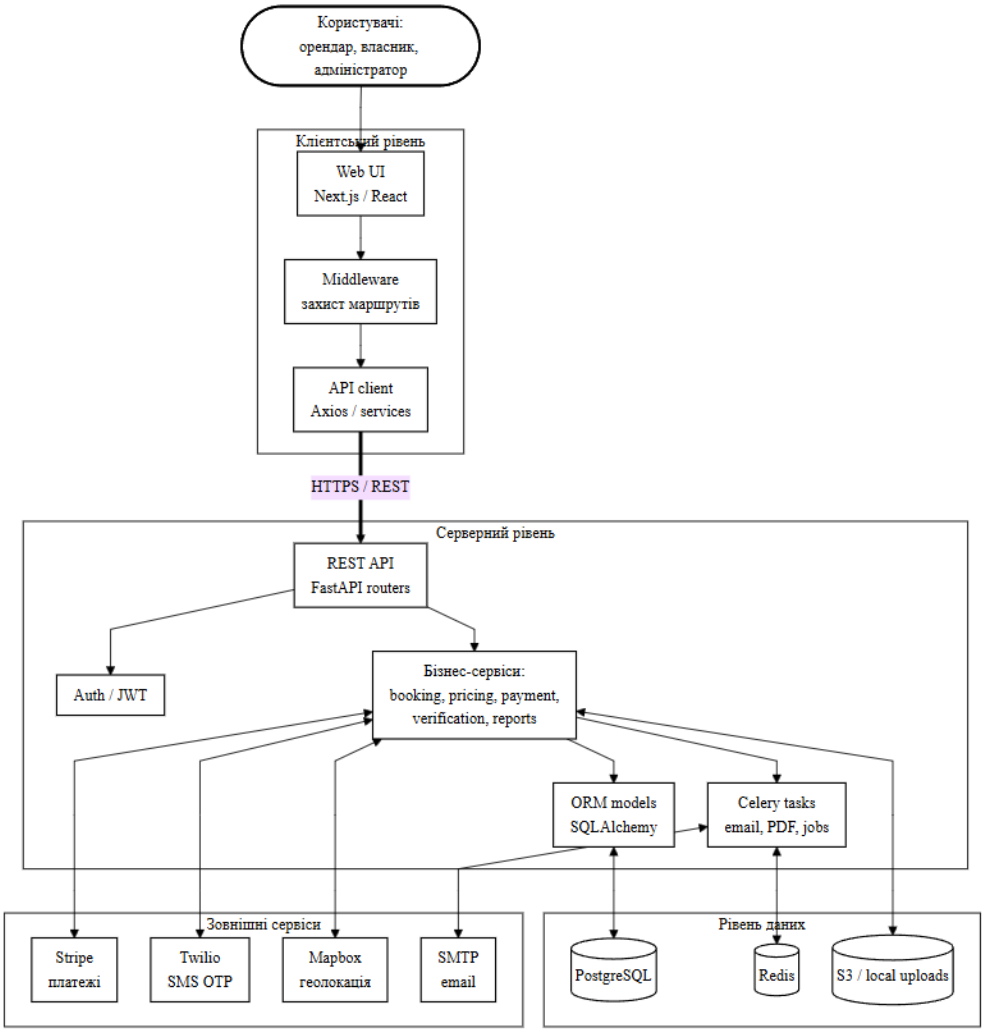

Рис. 2.1 Архітектура інформаційної системи

На рисунку 2.1 клієнтська частина представлена веб-інтерфейсом Next.js/React, проміжним шаром захисту маршрутів та API-клієнтом. Для орендаря передбачено сторінки профілю, бронювання й оплати, а для власника автомобіля — кабінет, керування автомобілями та заявками. Веб-інтерфейс не працює з базою напряму: усі дії проходять через REST API [6, 9, 10].

Серверний рівень містить API-маршрути FastAPI, Pydantic-схеми, сервісні модулі та SQLAlchemy-моделі. У проекті API розділено на публічні маршрути каталогу, профіль орендаря, платежі, бізнес-кабінет власника, документи та адміністративні функції [6, 7]. Такий поділ зручний для контролю доступу: орендар бачить власні дані й заявки, власник працює зі своїми автомобілями, а службові операції залишаються в адміністративній частині.

Рівень даних складається з PostgreSQL, Redis та сховища медіафайлів. У PostgreSQL зберігаються користувачі, автомобілі, поїздки, платежі, відгуки та звітні значення [8]. Redis використовується для черг і фонових задач. Фотографії автомобілів і файли, пов'язані з поїздками, можуть зберігатися в S3 або локальному каталозі uploads, що вже враховано у backend.

Таблиця 2.1

Рівні архітектури інформаційної системи

Рівень архітектури	**Компоненти**	**Призначення**
Клієнтський	Next.js, React, middleware, API services	Взаємодія з користувачем, маршрути, форми, каталог, кабінети
Серверний	FastAPI, routers, schemas, services	Бізнес-логіка, валідація, авторизація, API
Рівень даних	PostgreSQL, Redis, S3/uploads	Зберігання структурованих даних, черги, медіафайли
Зовнішні сервіси	Stripe, Twilio, Mapbox, SMTP	Платежі, OTP, геолокація, повідомлення

У цій роботі сценарії різних ролей не змішуються. Публічний каталог і створення заявки залишаються доступними для орендаря, операції підтвердження, видачі та повернення винесені у бізнес-частину власника, а перевірки й службові дії залишаються за адміністратором. Через це код легше розділити на модулі й пояснити у розділі програмної реалізації.

2.2 Проєктування бази даних

Під час проєктування бази даних основною задачею було не просто зберегти список автомобілів, а пов'язати між собою власника, орендаря, транспортний засіб, період оренди, оплату, верифікацію та результат поїздки. Для такої структури обрано PostgreSQL, оскільки дані мають багато зв'язків і потребують узгодженого виконання пов'язаних операцій при створенні заявки та оновленні платежів [8, 24].

У реалізації найбільш навантаженою сутністю є Trip. Для неї передбачено індекси за автомобілем, орендарем, статусом і діапазоном дат. Це потрібно для

практичних операцій: швидко перевірити, чи не перетинається нове бронювання з існуючими поїздками, показати історію орендаря, сформувати список заявок власника або відібрати поїздки для звіту.

Логічну ER-модель бази даних наведено на рисунку 2.2. У діаграмі показано не всі службові поля, а лише основні сутності та зв'язки, потрібні для розуміння інформаційної структури системи.

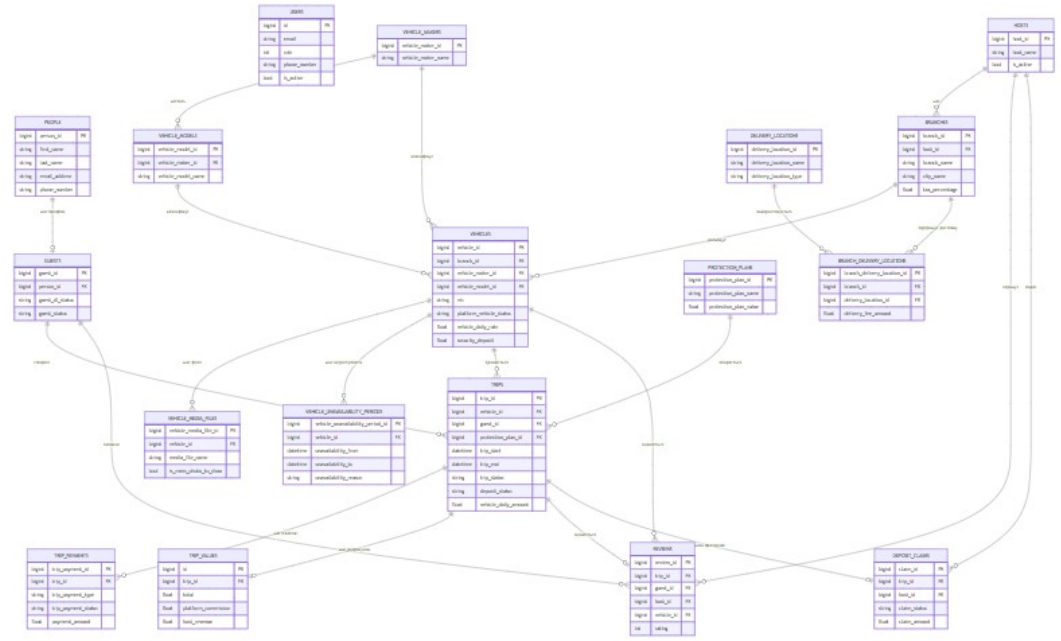

Рис. 2.2 ER-діаграма бази даних інформаційної системи

У моделі Trip зберігається не тільки період оренди, а й фінансові значення на момент створення заявки: добова ставка, сума оренди, застава, комісії, доставка, податки та статус депозиту. Це зроблено свідомо: якщо власник пізніше змінить тариф автомобіля, уже створена поїздка не втрачає свій фінансовий розрахунок.

Сутність Vehicle зберігає VIN, номер, марку, модель, тип кузова, паливо, добову ставку, заставу, статус публікації та параметри доступності. Фото автомобіля винесені в VehicleMediaFile, а періоди ремонту або особистого використання — в VehicleUnavailabilityPeriod. Через це пошук може окремо враховувати технічні характеристики авто і його календарну доступність.

Таблиця 2.2

Основні сутності бази даних

Сутність	Призначення	Ключові зв'язки
User	Облікові записи та ролі доступу	Пов'язується з роллю власника, адміністратора або іншого користувача
Person / Guest	Персональні дані орендаря та статус верифікації	Один орендар може мати багато бронювань
Host	Власник автомобілів або орендодавець	Має філії, автомобілі, відгуки та звіти
Branch	Точка видачі або підрозділ власника	Містить автомобілі та параметри доставки
Vehicle	Автомобіль, який може бути орендований	Належить філії, має бронювання, фото, недоступність
Trip	Заявка або поїздка	Пов'язує орендаря, автомобіль, період, статус і платежі
TripPayment	Облік оплат, застави, повернень і доплат	Належить конкретній поїздці
TripValue	Фінансова деталізація поїздки	Містить підсумкові суми та комісії
Review	Оцінка після завершення оренди	Пов'язана з поїздкою, орендарем, авто та власником
DepositClaim	Претензія щодо застави	Подається власником і розглядається адміністратором

Зв'язки в ER-моделі відповідають основним правилам сервісу. Один власник може мати кілька філій, філія містить багато автомобілів, автомобіль може мати багато поїздок, а поїздка може мати кілька платежів: окремо заставу, основну оплату, продовження або повернення коштів. Відгук і претензія щодо застави прив'язуються до конкретної поїздки, щоб при розгляді спору було зрозуміло, про яку оренду йдеться.

SQL-структура таблиць не наводиться у тексті розділу, оскільки для бакалаврської роботи важливішим є логічне проєктування даних. Фізична

реалізація схеми, міграції Alembic та фрагменти моделей SQLAlchemy доцільно виносити у розділ програмної реалізації або додатки.

2.3 Проєктування функціональних модулів

Функціональну структуру системи сформовано за модулями, які збігаються з реальними сценаріями користувачів: автентифікація, каталог автомобілів, бронювання, розрахунок вартості, платежі, верифікація, відгуки, звіти, документи та повідомлення. Такий поділ допомагає розглядати окремі частини системи без зайвого змішування різних правил предметної області [23].

На серверному рівні цей поділ реалізовано через API-роутери, Pydantic-схеми, сервісні модулі та ORM-моделі. Роутер приймає HTTP-запит, схема перевіряє дані, сервісний модуль виконує бізнес-операцію, а модель працює з базою. На клієнтському рівні поділ видно у сторінках, компонентах, hooks, контекстах та API-services.

Діаграму компонентів системи наведено на рисунку 2.3.

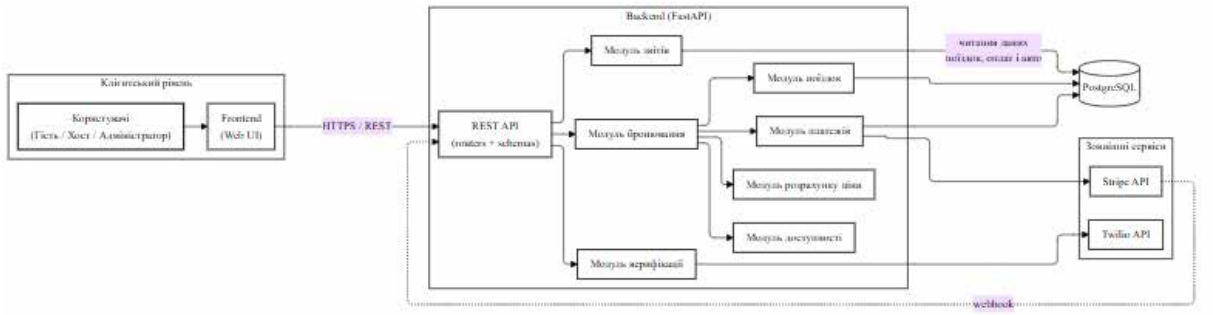

Рис. 2.3 Діаграма компонентів інформаційної системи

Під час побудови діаграми використано правила UML для component diagram: система поділена на самостійні компоненти за відповідальністю, зв'язки показані як залежності між компонентами, а точки інтеграції позначені інтерфейсами provided/required [18]. Це дозволяє чітко бачити межі модулів і не змішувати внутрішню логіку з зовнішніми сервісами.

На рисунку 2.3 показано поділ системи на клієнтську частину, серверну частину, базу даних, фонову обробку задач і зовнішні сервіси. Клієнтська частина взаємодіє із серверною через REST-запити [6, 22]. Основна логіка

згрупована у модулі бронювання, оплати, розрахунку вартості, верифікації, звітів та документів.

Таблиця 2.3

Функціональні модулі backend

Модуль	Призначення
Auth	Автентифікація користувачів, токени доступу, перевірка ролей
Vehicle catalog	Пошук автомобілів, фільтри, картка авто, медіафайли
Booking	Створення заявки, перевірка доступності, контроль статусів
Pricing	Розрахунок вартості, застави, комісій, доставки та коефіцієнтів
Payments	Створення платежів, інтеграція зі Stripe, асинхронне підтвердження оплати
Verification	Перевірка водійських даних та OTP-підтвердження
Reviews	Створення та відображення відгуків після завершення поїздки
Reports	Фінансові звіти та показники використання автомобілів
Documents	Формування PDF-квитанцій, рахунків та документів поїздки
Notifications	Email/SMS-повідомлення та фонові задачі

Модуль бронювання відповідає за перевірку доступності автомобіля, створення попередньої заявки, контроль конфліктів дат і перехід поїздки між статусами. Для цього у проекті окремо виділено модель допустимих переходів: перехід від попередньої заявки до фактичної оренди виконується через підтвердження та оплату.

Модуль звітності має інший характер: він переважно читає накопичені дані про поїздки, платежі та автомобілі. Тому його можна розглядати як аналітичне доповнення до основного процесу бронювання й оплати.

2.4 Проєктування API та взаємодії компонентів

Взаємодія клієнтської і серверної частин виконується через REST API. У проекті маршрути згруповані за доменами: публічний каталог, створення заявки, профіль орендаря, платежі, бізнес-кабінет власника, звіти, документи та адміністративні функції [6].

Для передачі даних використовуються JSON-запити та відповіді. Вхідні дані перевіряються Pydantic-схемами, а доступ до захищених маршрутів контролюється токенами доступу. У клієнтській частині запити винесені в сервісні модулі; тому компонент сторінки бронювання не містить у собі всю мережеву логіку, а викликає підготовлену функцію клієнтського API [6, 9].

Таблиця 2.4

Основні групи API інформаційної системи

Група API	**Приклади операцій**	**Призначення**
Public catalog	GET /vehicles, POST /vehicles/search	Пошук і перегляд автомобілів
Trips	POST /trips/quote	Створення заявки на бронювання
Payments	POST /payments/checkout, підтвердження Stripe	Оплата оренди та оновлення статусу платежу
Guest profile	GET /profile/trips, POST /driver_license	Особистий кабінет орендаря
Business/host	GET /business/trips, POST /approve	Кабінет власника та керування поїздками
Reports	GET /reports/summary	Фінансова та операційна звітність
Documents	GET /documents/trips/{id}/receipt	PDF-квитанції та документи поїздки

Одним із ключових сценаріїв є створення бронювання з подальшою оплатою. Послідовність взаємодії компонентів у цьому сценарії наведено на рисунку 2.4.

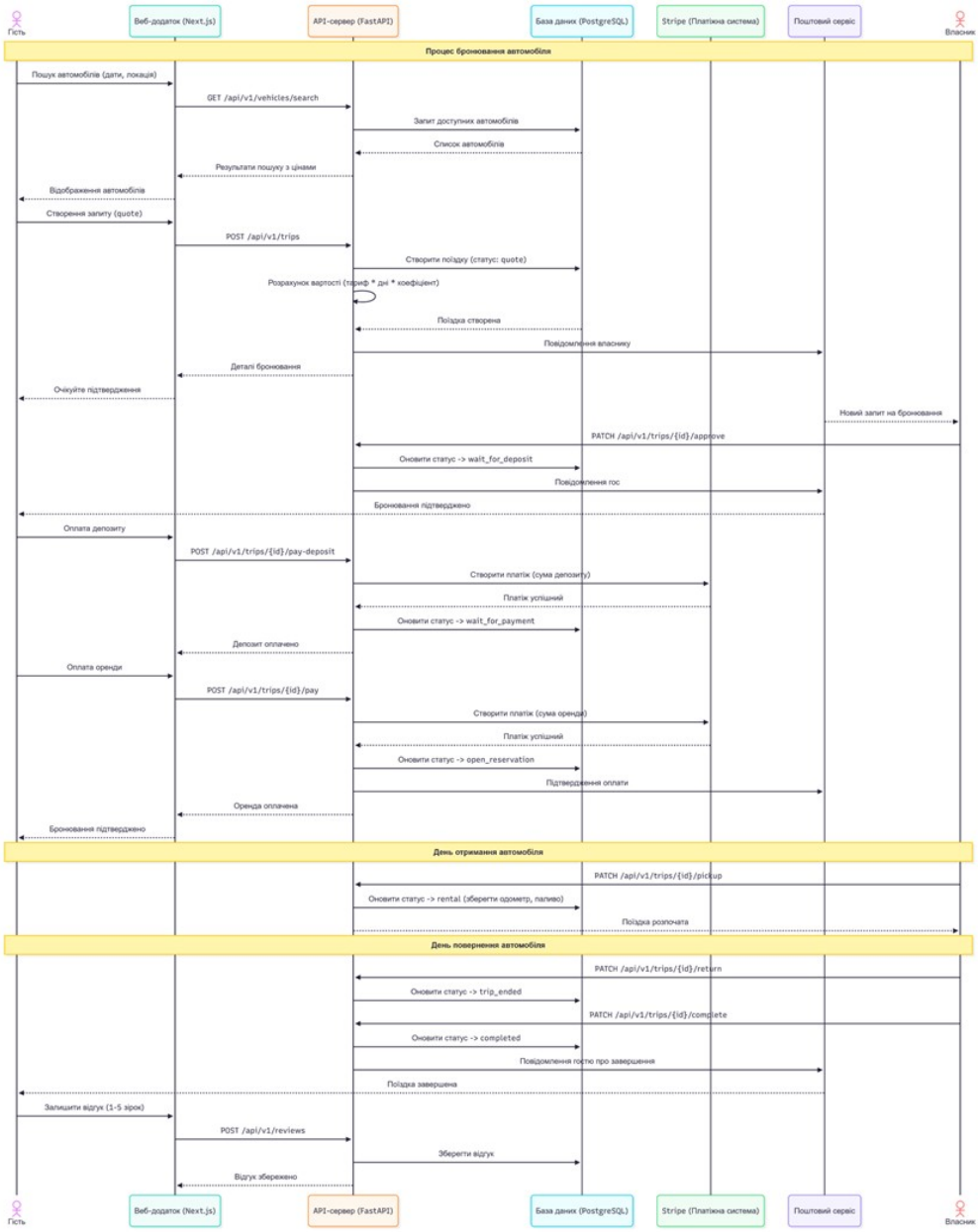

Рис. 2.4 Діаграма послідовності створення та оплати бронювання

Сценарій починається з вибору автомобіля, дат і місця отримання. Клієнтська частина надсилає запит на створення попередньої заявки. Серверна частина спочатку перевіряє доступність автомобіля: враховуються вже створені поїздки та записи про періоди недоступності. Якщо є перетин дат, користувач отримує відповідь про недоступність автомобіля.

Якщо автомобіль доступний, серверна частина розраховує вартість: добову ставку, тривалість, план захисту, доставку, податки, комісії та заставу. Після цього створюються записи Trip і TripValue. Для оплати використовується

Stripe checkout-сесія; після підтвердження платіжним сервісом оновлюються TripPayment і статус поїздки.

У цьому сценарії статус поїздки змінюється поступово: від quote до підтвердженого бронювання, оренди та завершення.

2.5 Вибір засобів реалізації

Вибір засобів реалізації здійснено з урахуванням вимог до клієнт-серверної архітектури, підтримки типізації, інтеграції зовнішніх сервісів та можливості розгортання [20, 25]. Для клієнтської частини обрано Next.js і React [9, 10], для серверної — FastAPI [6], для бази даних — PostgreSQL [8], для ORM — SQLAlchemy [7], для міграцій — Alembic. Додатково використовуються Docker [14], механізм фонових задач на базі Redis/Celery, Stripe [11], Twilio [12], Mapbox [13], S3 та SMTP.

Таблиця 2.5

Засоби реалізації інформаційної системи

Засіб	**Роль у системі**	**Обґрунтування вибору**
Next.js / React	Клієнтський веб-інтерфейс	Маршрути, компоненти, адаптивний UI, інтеграція з API
FastAPI	Серверне API	Розробка REST API, типізація, OpenAPI-документація
PostgreSQL	Реляційна база даних	Транзакції, зв'язки, індекси, надійність для облікових даних
SQLAlchemy	ORM та міграції	Опис моделей у коді та контроль змін схеми БД
Docker	Розгортання	Відтворюване середовище для клієнтської частини, серверної частини і сервісів
Stripe	Платежі	Платіжні сесії, підтвердження оплати, зменшення обсягу власної фінансової логіки
Twilio	SMS/OTP	Підтвердження телефону та додатковий фактор перевірки
Mapbox	Геолокація	Пошук адрес, координати, карта автомобілів

Висновки до розділу 2

У другому розділі загальні вимоги з першого розділу перетворено на проєктні рішення. Для системи обрано клієнт-серверну архітектуру: клієнтська частина відповідає за взаємодію з користувачем, серверна — за правила предметної області та API, а PostgreSQL — за збереження структурованих даних.

Окремо спроєктовано логічну структуру бази даних. Центральними сутностями стали користувачі, власники, орендарі, автомобілі, бронювання, платежі, фінансові значення, відгуки та претензії щодо застави. Така модель відображає реальний процес оренди і готує основу для програмної реалізації.

Також визначено функціональні модулі, групи API, послідовність створення й оплати бронювання та схему розгортання. Розділ не зводиться до переліку технологій: він пояснює, чому саме така структура потрібна для подальшого опису програмної частини.

3 ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

У третьому розділі описано програмну реалізацію інформаційної системи. Розроблена система є повноцінним веб-застосунком із серверною частиною на FastAPI [6], клієнтською частиною на Next.js і React [9, 10], базою даних PostgreSQL [8], інтеграціями з платіжним сервісом, SMS-підтвердженням, картографічним сервісом, файловим сховищем та фоновими задачами [11-14].

Мета розділу полягає у демонстрації практичного втілення проєктних рішень. У тексті наведено структуру проєкту, основні модулі, приклади коду та пояснення ключових сценаріїв: пошуку автомобіля, створення бронювання, перевірки доступності, оплати, авторизації, роботи кабінету власника і формування звітної інформації.

Фрагменти коду подано вибірково, оскільки повні файли серверної частини або React-компоненти доцільно виносити у додатки. У розділі наведено лише ті частини, які пояснюють основну логіку: маршрутизацію API, перевірку доступності автомобіля, зміну статусів поїздки, оплату, захист маршрутів і взаємодію клієнтської частини із серверною.

3.1 Реалізація серверної частини

Серверна частина реалізована у каталозі backend-carrental. Вона побудована на FastAPI і відповідає за REST API, перевірку даних, авторизацію, роботу з PostgreSQL, платежі, верифікацію орендаря, звіти і службові операції власника автомобіля [6, 8]. Саме тут зосереджена основна бізнес-логіка системи.

У проекті використано поділ на файли обробників API, Pydantic-схеми, сервісні модулі та SQLAlchemy-моделі [6, 7]. Така організація зменшує дублювання логіки між різними маршрутами API. Наприклад, перевірка доступності автомобіля винесена в окремий сервіс і може використовуватися як при створенні бронювання, так і при продовженні поїздки.

Спрощену структуру backend наведено нижче. Вона показує, де знаходяться основні частини серверної реалізації.

```
backend-carrental/app/
  api/v1/endpoints/      # HTTP-маршрути FastAPI
  models/                # SQLAlchemy-моделі таблиць
  schemas/               # Pydantic-схеми запитів і відповідей
  services/              # бізнес-логіка та інтеграції
    tasks/                 # фонові задачі Celery
    auth.py                # токени доступу, хешування паролів, ролі
  database.py            # підключення до PostgreSQL
  main.py                # створення FastAPI-застосунку
```

Лістинг 3.1 – Загальна структура серверної частини

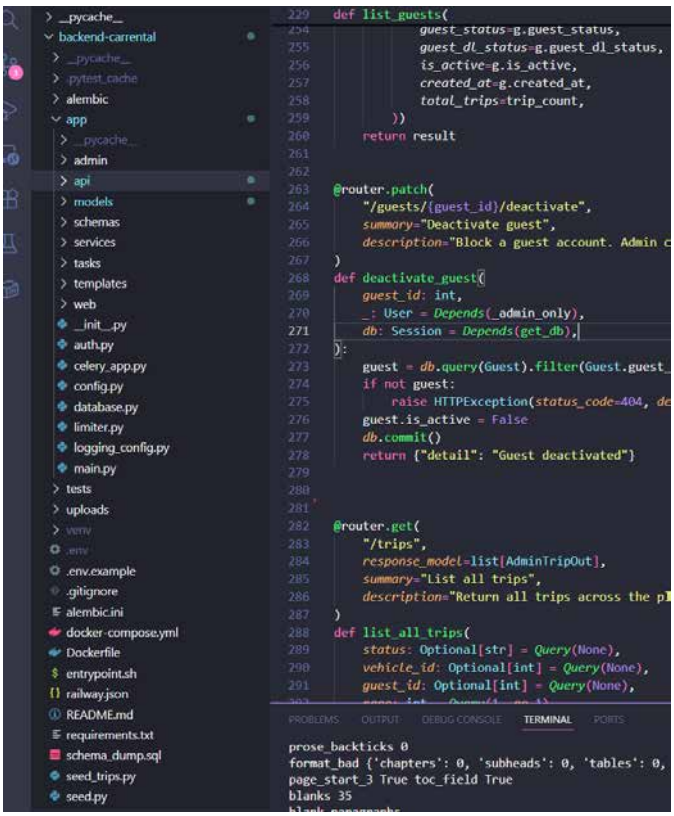

Рис. 3.1 Структура backend-проєкту у середовищі VS Code

Початкова точка серверної частини знаходиться у файлі main.py. Саме там створюється FastAPI-застосунок, підключається CORS, обмеження частоти запитів, OpenAPI/Swagger і центральний router. Окремо описані OpenAPI-теги, завдяки чому документація API у Swagger виглядає не як один великий список маршрутів, а як набір зрозумілих груп: публічний каталог, поїздки, платежі, кабінет власника, профіль орендаря, документи та адміністрування.

Центральний файл маршрутизації app/api/v1/router.py підключає маршрути за предметними доменами. Це простий, але важливий момент реалізації: у системі багато функцій, тому один великий файл з усіма маршрутами швидко став би незручним. Поділ на окремі router-и полегшує розробку і пошук помилок.

```
api_router = APIRouter()


api_router.include_router(vehicles_public_router)
api_router.include_router(trips_public_router)
api_router.include_router(payments_router)
api_router.include_router(host_dashboard_router)
api_router.include_router(business_trips_router)
api_router.include_router(guest_profile_router)
api_router.include_router(admin_router)
```

Лістинг 3.2 – Підключення основних router-ів FastAPI

У цьому фрагменті видно, що API поділено не випадково, а відповідно до ролей і сценаріїв. Публічні маршрути потрібні для каталогу і створення заявки. Business-маршрути використовуються власником автомобіля. Guest profile відповідає за особистий кабінет орендаря. Admin-маршрути призначені для службового контролю

Таблиця 3.1

Основні модулі серверної частини

Група backend-модулів	**Приклади файлів**	**Що реалізує**
Public API	vehicles_public.py, trips_public.py	Каталог, пошук, створення quote-заявки
Business API	business_trips.py, business_vehicles.py	Кабінет власника, авто, заявки, статуси
Guest API	guest_profile.py	Профіль орендаря, поїздки, водійські дані
Payments	payments.py, stripe_service.py	Оплата, платіжна сесія, підтвердження Stripe, облік платежів
Services	availability.py, price_calculator.py	Перевірки, розрахунки, повторно використана логіка
Models	trip.py, vehicle.py, user.py	Відображення таблиць PostgreSQL у коді
Tasks	trip_tasks.py	Фонові задачі: нагадування, скасування старих заявок

Один із ключових серверних маршрутів відповідає за створення заявки на бронювання. Він розміщений у файлі trips_public.py і використовується для початкового оформлення оренди. Цей маршрут не просто створює рядок у таблиці: спочатку він перевіряє автомобіль, мінімальний термін оренди, доступність на вибрані дати, створює або знаходить орендаря, розраховує ціну і тільки після цього зберігає поїздку у статусі попередньої заявки.

```
@router.post("/quote", response_model=TripDetailOut)
def create_quote(body: TripQuoteRequest, db: Session = Depends(get_db)):
    vehicle = db.query(Vehicle).filter(
        Vehicle.vehicle_id == body.vehicle_id,
        Vehicle.is_active == True,
    ).first()
    if not vehicle:
        raise HTTPException(status_code=404, detail="Vehicle not found")


    if not check_vehicle_available(db, body.vehicle_id, body.trip_start,
body.trip_end):
        raise HTTPException(status_code=409, detail="Vehicle is not
available")


    price = calculate_trip_price(db=db, vehicle=vehicle,
        pickup_date=body.trip_start, return_date=body.trip_end,
        protection_plan_id=body.protection_plan_id)
```

Лістинг 3.3 – Скорочений фрагмент створення заявки на бронювання

У повному коді після цього створюються записи Trip, TripValue і, за потреби, профіль орендаря. Для дипломної роботи важливо показати саме порядок дій: спочатку перевірка, потім розрахунок, потім запис у базу. Такий порядок зменшує ризик появи некоректних бронювань.

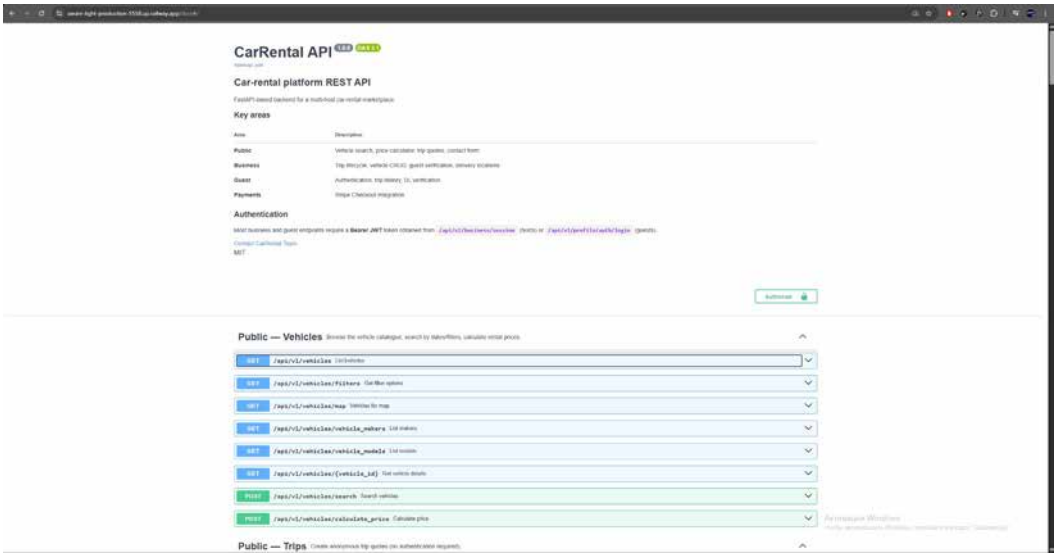

Рис. 3.2 Swagger/OpenAPI документація backend API

Swagger-документація є корисною частиною реалізації, тому що вона показує реальні маршрути, схеми запитів і відповіді сервера. Для комісії це швидкий спосіб побачити, що API дійсно існує і його можна перевіряти не тільки через frontend.

3.2 Реалізація клієнтської частини

Клієнтська частина розташована у каталозі frontend і реалізована на Next.js, React та TypeScript [9, 10]. Вона відповідає за сторінки каталогу, картку автомобіля, бронювання, оплату, профіль орендаря і кабінет власника автомобіля. Клієнтська частина не працює напряму з базою даних. Усі дані вона отримує через REST API серверної частини [6].

У проекті використовується маршрутизація Next.js App Router. Сторінки організовані за папками у src/app: каталог, сторінка конкретного автомобіля, бронювання, оплата, профіль орендаря та сторінки власника. Така структура відповідає логіці користувацьких сценаріїв і спрощує навігацію в коді.

```
frontend/src/app/
  catalog/page.tsx          # каталог автомобілів
  car/[id]/page.tsx         # детальна сторінка автомобіля
  booking/[id]/page.tsx     # оформлення бронювання
  payment/page.tsx          # сторінка оплати
  profile/trips/            # поїздки орендаря
  host/dashboard/           # кабінет власника
  host/vehicles/            # керування автомобілями
```

Лістинг 3.4 – Основні маршрути клієнтської частини

Таблиця 3.2

Основні сторінки frontend-застосунку

Сторінка	**Файл**	**Призначення**
Каталог	catalog/page.tsx	Завантаження фільтрів і списку автомобілів
Картка авто	car/[id]/page.tsx	Детальна інформація, фото, характеристики
Бронювання	booking/[id]/page.tsx	Дати, місце, план захисту, дані орендаря
Оплата	payment/page.tsx	Перехід до оплати і результат платежу
Профіль	profile/trips	Активні та історичні поїздки орендаря
Кабінет власника	host/dashboard	Статистика, заявки, автопарк

Сторінка каталогу завантажує визначення фільтрів і початковий список автомобілів. Це зроблено на серверному боці Next.js, а вже далі інтерактивна частина передається клієнтському компоненту. Такий підхід дає нормальний перший рендер сторінки і водночас залишає користувачу можливість змінювати фільтри без повного перезавантаження.

Сторінка автомобіля використовує динамічний маршрут. Вона отримує дані про конкретний автомобіль через функцію getCarDetails, нормалізує відповідь серверної частини і передає її у компонент CarDetailsClient. На цій сторінці користувач бачить фото, характеристики, ціну, рейтинг, доступні локації та може перейти до бронювання.

Процес бронювання реалізований окремо, тому що він має більше станів, ніж звичайна сторінка перегляду. Компонент BookingClient працює з датами, місцем отримання і повернення, планом захисту, додатковими послугами, водійськими даними і помилками створення заявки. Частина даних може передаватися через URL-параметри, щоб користувач не втрачав вибір після переходу з каталогу або картки автомобіля.

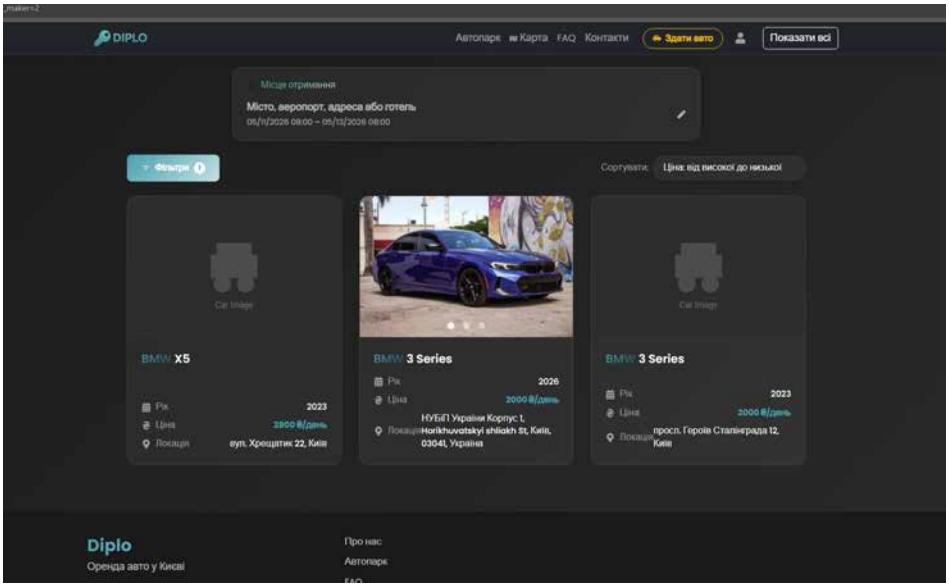

Рис. 3.3 Сторінка каталогу автомобілів

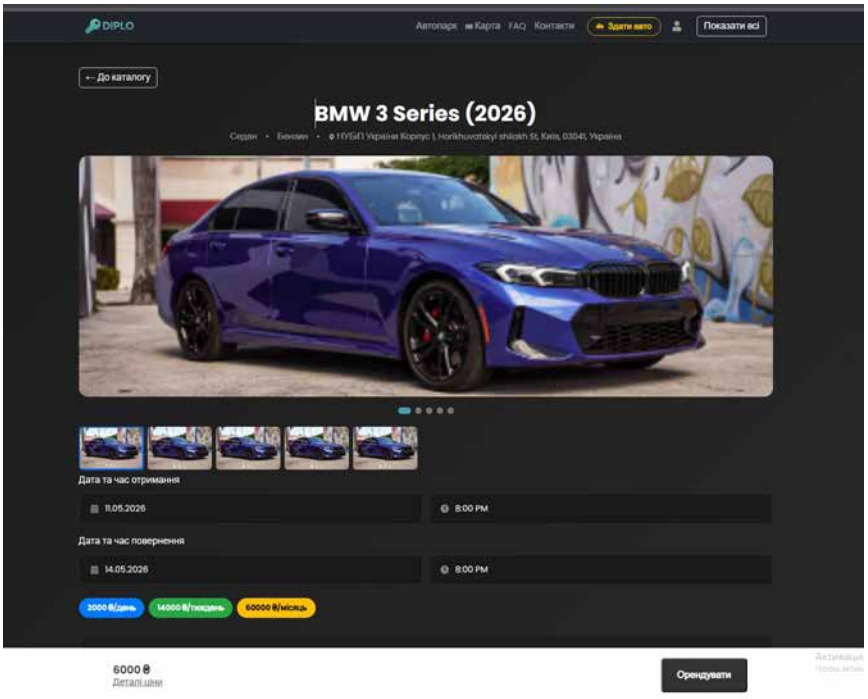

Рис. 3.4 Детальна сторінка автомобіля

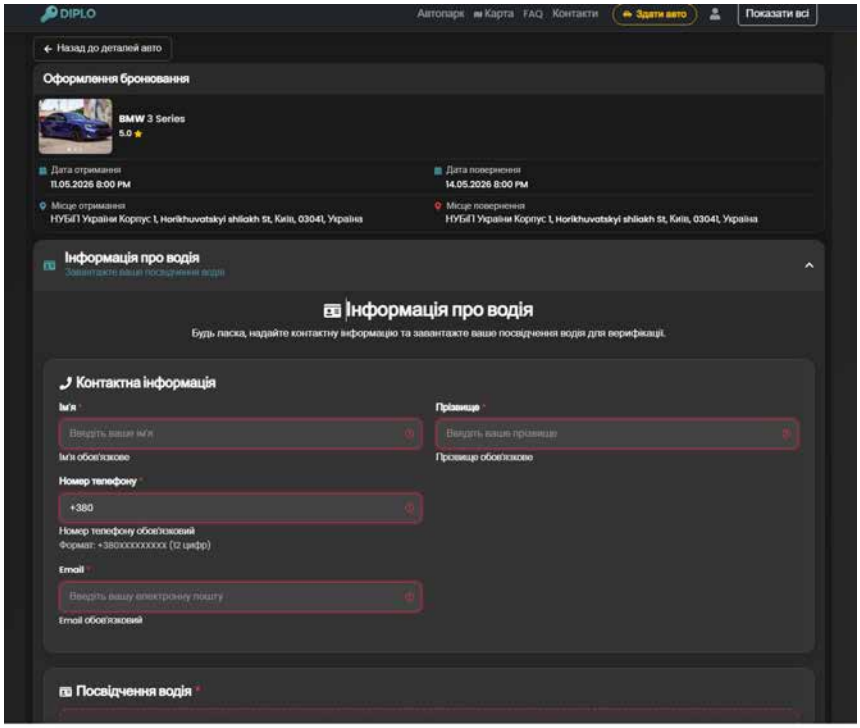

Рис. 3.5 Сторінка бронювання автомобіля

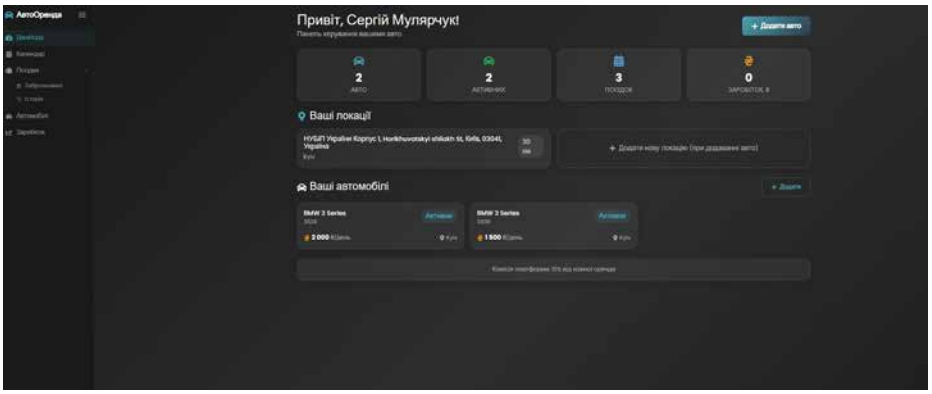

Рис. 3.6 Кабінет власника автомобіля

Для запитів клієнтської частини до серверної використовується спільний API-клієнт. Він створений на основі Axios і автоматично додає токен доступу до заголовка Authorization, якщо користувач авторизований. Це зручно, тому що компонентам сторінок не потрібно кожного разу вручну додавати токен.

```
const apiClient = axios.create({
  baseURL: API_BASE_URL,
  headers: { 'Content-Type': 'application/json' },
});


apiClient.interceptors.request.use(async (config) => {
  let token = authService.getAccessToken();
  if (!token) token = await authService.getAccessTokenAsync();
  if (token) config.headers.Authorization = `Bearer ${token}`;
  return config;
});
```

Лістинг 3.5 – Додавання токена доступу до API-запитів клієнтської частини

Окремо реалізований вибір локацій. Коли користувач вводить адресу або назву місця, frontend звертається до Mapbox API через власні Next.js API routes. Після вибору адреси зберігаються назва, координати і тип місця. Ці дані використовуються під час пошуку автомобілів і розрахунку доставки.

3.3 Реалізація бази даних

База даних реалізована в PostgreSQL [8], а в серверній частині вона описана за допомогою SQLAlchemy ORM [7]. У другому розділі вже була показана ER-діаграма, тому тут доцільно зосередитися на тому, як ця модель відображена в коді. Основні моделі знаходяться у папці app/models.

Ключовою сутністю для предметної області є Trip. Вона не просто зберігає факт бронювання. У ній є дати, автомобіль, орендар, локації, статус, фінансові параметри, застава, дані водійського посвідчення і службові поля. Саме через цю модель можна відстежити шлях заявки від створення до завершення поїздки.

```
class Trip(AuditMixin, Base):
    __tablename__ = "trips"
    __table_args__ = (
        Index("idx_trips_vehicle_id", "vehicle_id"),
        Index("idx_trips_guest_id", "guest_id"),
        Index("idx_trips_status", "trip_status"),
        Index("idx_trips_start_end", "trip_start", "trip_end"),
    )


    trip_id = mapped_column(BigInteger, primary_key=True)
    trip_start = mapped_column(DateTime, nullable=False)
    trip_end = mapped_column(DateTime, nullable=False)
    guest_id = mapped_column(BigInteger, ForeignKey("guests.guest_id"))
    vehicle_id = mapped_column(BigInteger, ForeignKey("vehicles.vehicle_id"))
```

Лістинг 3.6 – Скорочений фрагмент SQLAlchemy-моделі Trip

Індекси для автомобіля, орендаря, статусу і діапазону дат потрібні для забезпечення прийнятної швидкодії основних запитів. Система часто шукає поїздки конкретного автомобіля, перевіряє зайнятість на період або відбирає заявки за статусом. Без індексів такі запити з часом працювали б повільніше.

Таблиця 3.3

Основні ORM-моделі системи

Модель	**Призначення**	**Чому винесена окремо**
User	Обліковий запис і роль	Вхід, права доступу, токени
Guest	Профіль орендаря	Водійські дані і статус перевірки
Host	Власник автомобілів	Кабінет, статистика, автопарк
Vehicle	Автомобіль	Характеристики, тариф, статус публікації
Trip	Бронювання/поїздка	Центральна бізнес-операція
TripPayment	Платіж	Окремий облік депозиту, оренди, refund
TripValue	Фінансовий розрахунок	Збереження суми, комісій, доходу
Review	Відгук	Рейтинг після завершення поїздки

Платежі не зберігаються прямо в таблиці trips, а винесені в TripPayment. Це зроблено тому, що одна поїздка може мати кілька фінансових операцій: депозит, основну оплату, продовження, повернення застави або додатковий

платіж. Якби все зберігалося в одному полі, було б складно пояснити історію оплат і працювати з підтвердженнями Stripe.

Розраховані фінансові значення винесені в TripValue. Там зберігаються орендна сума, податки, комісії, загальна сума, баланс поїздки, дохід платформи і дохід власника. Це важливо для звітності: власник може бачити підсумки, а адміністратор може перевірити, як була сформована сума.

3.4 Реалізація інтеграцій із зовнішніми сервісами

У проекті є кілька інтеграцій, які роблять систему ближчою до реального сервісу. Для платежів використовується Stripe [11], для SMS/OTP - Twilio [12], для адрес і карти - Mapbox [13], для файлів - S3 або локальне uploads-сховище, для email - SMTP. Окремі довші операції можуть виконуватися через механізм фонових задач на базі Celery і Redis.

Таблиця 3.4

Зовнішні сервіси, використані у реалізації

Інтеграція	**Де використовується**	**Роль у системі**
Stripe	stripe_service.py, payments.py	Оплата оренди, депозит, підтвердження платежу, повернення коштів
Twilio	guest/profile OTP	Підтвердження телефону орендаря
Mapbox	клієнтські API routes, bookingService	Пошук адрес, координати, карта
S3/uploads	vehicle media, trip media	Фотографії автомобілів і файли поїздки
SMTP	email service	Підтвердження бронювання, квитанції
Celery/Redis	tasks/trip_tasks.py	Фонові задачі та нагадування

Платіжний сценарій реалізовано як окрему послідовність дій. Спочатку серверна частина перевіряє, чи поїздка знаходиться у статусі, який дозволяє оплату. Потім створюється платіжна сесія Stripe. Після успішної оплати Stripe надсилає підтвердження, серверна частина створює запис TripPayment, оновлює баланс у TripValue і змінює статус поїздки.

```
@router.post("/webhooks/stripe", summary="Stripe webhook")
async def stripe_webhook(request: Request, db: Session = Depends(get_db)):
    payload = await request.body()
    sig_header = request.headers.get("stripe-signature")
    event = stripe.Webhook.construct_event(
        payload, sig_header, settings.STRIPE_WEBHOOK_SECRET
    )
    if event["type"] == "checkout.session.completed":
        handle_checkout_completed(db, event["data"])


    return {"status": "ok"}
```

Лістинг 3.7 – Обробка підтвердження Stripe після успішної оплати

Підтвердження від платіжного сервісу важливе тому, що клієнтська частина не має самостійно вирішувати, чи платіж пройшов успішно. Користувач може закрити сторінку або втратити з'єднання, але подія від Stripe все одно надходить на сервер. Через це фінансовий стан поїздки оновлюється на основі повідомлення платіжного сервісу, а не тільки на основі дій браузера.

Mapbox використовується для пошуку адрес і точок отримання автомобіля. Користувач вводить текст, frontend отримує список підказок, після вибору конкретного варіанту забирає повні дані про адресу і координати. Далі ці координати можуть використовуватися для пошуку автомобілів, розрахунку доставки або відображення на карті.

```
static async getSearchLocations(query: string): Promise<TSuggestionsList[]> {
  if (!query || query.trim().length < 2) return [];


  const params = {
    query: query.trim(),
    types: 'address,poi,city',
    country: 'ua',
    sessionToken: this.getSessionToken(),
  };
  return await LocationsApi.fetchSuggestionList(params);
}
```

Лістинг 3.8 – Пошук адрес через Mapbox у клієнтській частині

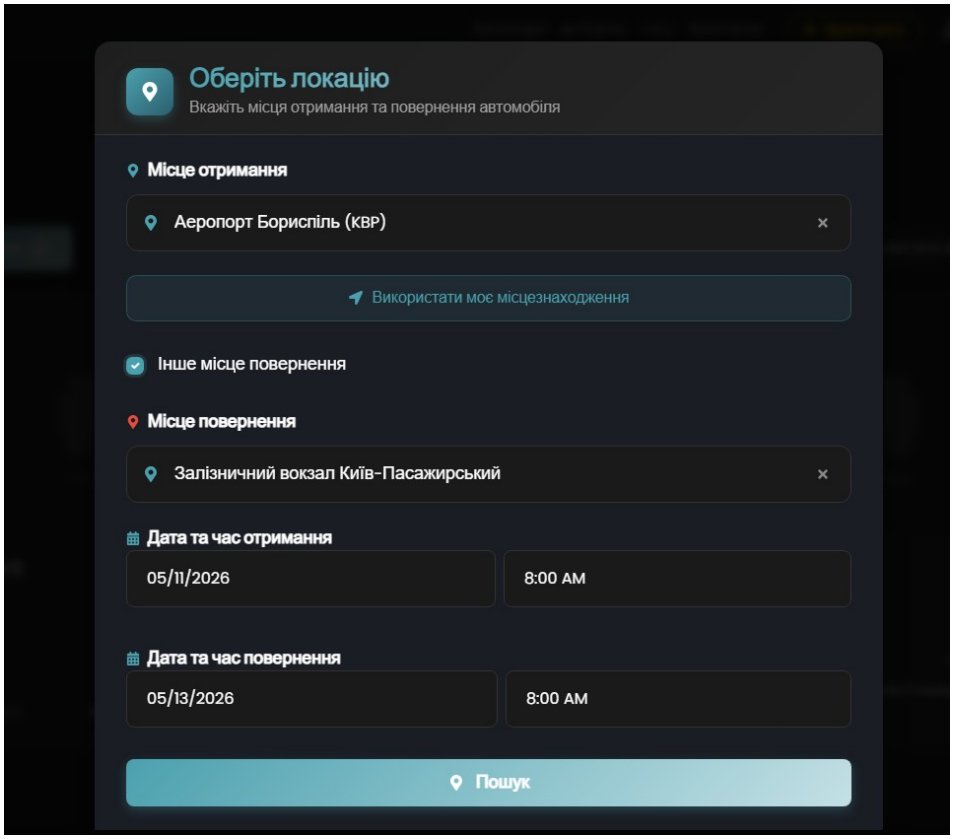

Рис. 3.7 Вибір адреси отримання або повернення автомобіля

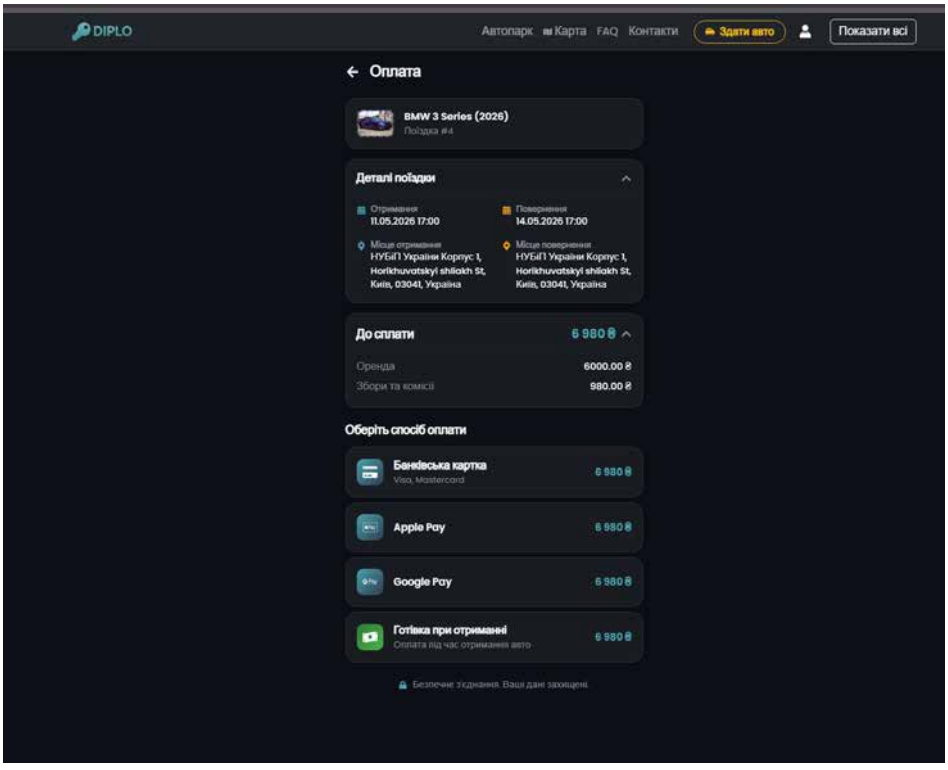

Рис. 3.8 Сторінка оплати після створення бронювання

Файли автомобілів і поїздок зберігаються окремо від основних таблиць. У базі зберігаються посилання, назви файлів, групи медіа і службові ознаки. Це нормальний підхід для веб-системи: база даних відповідає за структуру, а файлове сховище - за самі зображення.

3.5 Реалізація авторизації та безпеки

У системі є кілька ролей користувачів, тому авторизація не обмежується просто входом за email і паролем. Після входу користувач отримує токен доступу, а серверна частина перевіряє його у захищених маршрутах. Для паролів використовується хешування через bcrypt/passlib, тобто пароль не зберігається у відкритому вигляді.

У серверній частині є окремі залежності FastAPI для поточного користувача, поточного орендаря і перевірки ролей. Це дає змогу на рівні маршрутів API обмежувати доступ до службових функцій. Власник автомобіля працює зі своїми заявками й автомобілями, орендар бачить власні поїздки, а адміністративні дії залишаються в окремій службовій частині.

```
def create_access_token(data: dict, expires_delta: timedelta | None = None)
-> str:
    to_encode = data.copy()
    expire = datetime.now(timezone.utc) + (
        expires_delta or
timedelta(minutes=settings.ACCESS_TOKEN_EXPIRE_MINUTES)
    )
    to_encode.update({"exp": expire, "token_type": "access"})
    return jwt.encode(to_encode, settings.SECRET_KEY,
algorithm=settings.ALGORITHM)


def require_roles(*roles: UserRole):
    def _check(user: User = Depends(get_current_user)) -> User:
        if UserRole(user.role) not in roles:
            raise HTTPException(status_code=403, detail="Insufficient
permissions")
        return user
    return _check
```

Лістинг 3.9 – Токен доступу і перевірка ролей у серверній частині

Таблиця 3.5

Ролі користувачів та обмеження доступу

Роль	Доступ у системі	Приклад функцій
Орендар	Особистий профіль і свої поїздки	Бронювання, оплата, відгук
Власник	Свої автомобілі, заявки, звіти	Підтвердження quote, керування авто
Адміністратор	Службові дані платформи	Користувачі, перевірки, спірні ситуації
Співробітник	Обмежені операції власника	Перегляд або фіксація стану поїздки

У клієнтській частині також є захист маршрутів. Для цього використовується middleware.ts. Якщо користувач відкриває сторінку профілю без токена орендаря, його перенаправляє на сторінку входу. Якщо власник намагається відкрити кабінет без токена власника, його перенаправляє на сторінку входу для власника.

```
function isProtectedRoute(pathname: string): boolean {
  return PROTECTED_ROUTES.some(route =>
    pathname === route || pathname.startsWith(`${route}/`)
  );
}


if (isHostProtectedRoute(pathname)) {
  if (!hasValidHostToken(request)) {
    const redirectUrl = new URL(HOST_LOGIN_URL, request.url);
    redirectUrl.searchParams.set('redirect', pathname);
    return NextResponse.redirect(redirectUrl);
  }
}
```

Лістинг 3.10 – Захист маршрутів у Next.js middleware

Цей frontend-захист не замінює backend-перевірки, але робить роботу користувача зрозумілішою. Якщо токена немає або він прострочений, користувач одразу потрапляє на потрібну сторінку входу, а не бачить набір помилок від API.

3.6 Особливості реалізації бізнес-логіки

Найважливіша частина реалізації - це бізнес-логіка бронювання. Система не є простим CRUD-застосунком, де користувач тільки створює, редагує і

видаляє записи. У процесі оренди є життєвий цикл заявки, перевірка доступності, розрахунок ціни, оплата депозиту, підтвердження власником, старт і завершення поїздки, повернення застави, претензії та відгуки.

Для керування статусами використовується окрема модель допустимих переходів у файлі trip_state_machine.py. Вона задає, з якого статусу в який можна перейти. Це зменшує ризик випадкових або некоректних змін. Наприклад, перехід від «quote» до «completed» передбачає проміжні етапи підтвердження, оплати, бронювання і фактичної оренди.

```
TRANSITIONS = {
    "quote": ["quote_approved_by_platform", "quote_cancelled_by_platform"],
    "quote_approved_by_platform": ["wait_for_deposit", "wait_for_payment"],
    "wait_for_deposit": ["wait_for_payment"],
    "wait_for_payment": ["open_reservation"],
    "open_reservation": ["booked"],
    "booked": ["rental"],
    "rental": ["trip_ended"],
    "trip_ended": ["completed"],
}
```

Лістинг 3.11 – Скорочений перелік переходів статусів поїздки

Таблиця 3.6

Життєвий цикл бронювання у системі

Етап	**Статус**	**Що відбувається**
Створення заявки	quote	Орендар вибрав авто і дати
Підтвердження	quote_approved_by_platform	Власник погодив заявку
Депозит	wait_for_deposit	Очікується оплата застави
Основна оплата	wait_for_payment	Очікується оплата оренди
Бронювання	booked	Поїздка підтверджена
Оренда	rental	Виданий автомобіль
Повернення	trip_ended	Автомобіль повернено
Завершення	completed	Розрахунок закрито

Окрема важлива частина - перевірка доступності автомобіля. Вона враховує не тільки вже підтверджені поїздки, а й ручні періоди недоступності. Наприклад, власник може закрити автомобіль на ремонт або для особистого

користування. У такому випадку автомобіль не відображається як доступний у вибраному діапазоні дат.

```
def check_vehicle_available(db, vehicle_id, pickup, return_,
exclude_trip_id=None):
    q = db.query(Trip).filter(
        Trip.vehicle_id == vehicle_id,
        Trip.trip_status.in_(BLOCKING_STATUSES),
        Trip.trip_start < return_,
        Trip.trip_end > pickup,
    )
    if exclude_trip_id:
        q = q.filter(Trip.trip_id != exclude_trip_id)
    if q.first():
        return False


    conflict = db.query(VehicleUnavailabilityPeriod).filter(
        VehicleUnavailabilityPeriod.vehicle_id == vehicle_id,
        VehicleUnavailabilityPeriod.is_active == True,
        VehicleUnavailabilityPeriod.vehicle_unavailability_date_from <
return_.date(),
        VehicleUnavailabilityPeriod.vehicle_unavailability_date_to >
pickup.date(),
    ).first()
    return conflict is None
```

Лістинг 3.12 – Перевірка перетину бронювань і періодів недоступності

Розрахунок ціни також не зводиться до множення добової ставки на кількість днів. У системі враховуються добова ставка, коефіцієнти тривалості, план захисту, доставка, податки, платіжні комісії, адміністративний збір і застава. Результат розрахунку зберігається у TripValue, щоб потім його можна було показати на сторінці оплати, у квитанції або звіті.

Для власника автомобіля реалізовані операції підтвердження, відхилення, старту, завершення і закриття поїздки. Під час старту можна фіксувати одометр і рівень пального, а під час повернення - фактичні дані після поїздки. Це робить систему ближчою до реального процесу оренди, де важливо мати не тільки факт платежу, а й стан автомобіля до та після використання.

Фонова логіка винесена в задачі Celery. Наприклад, система може скасовувати застарілі попередні заявки або нагадувати про початок поїздки. Такі дії не потрібно виконувати під час звичайного HTTP-запиту, тому їх логічно передати окремому процесу обробки фонових задач.

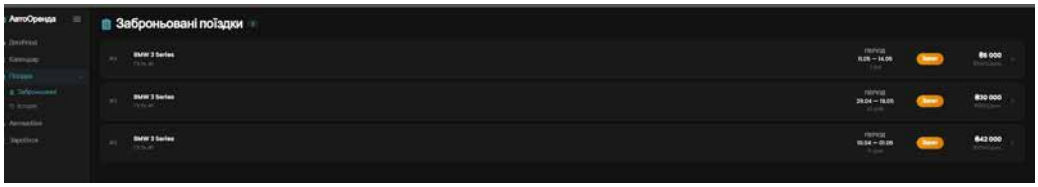

Рис. 3.9 Список поїздок або заявок у кабінеті власника

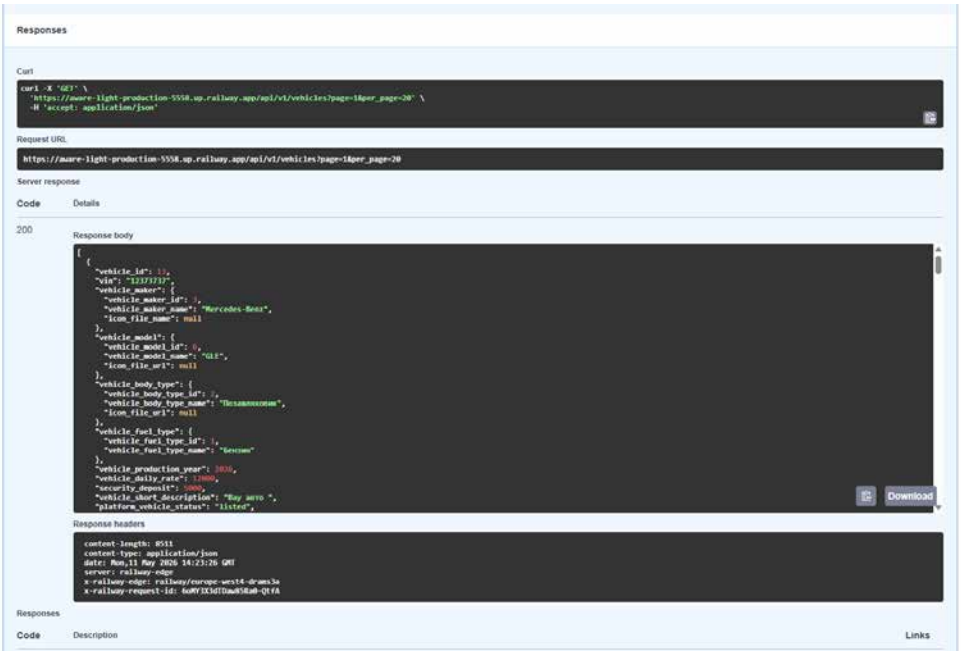

Рис. 3.10 Приклад відповіді API або тестування маршруту у Swagger

Завдяки такій реалізації система має кілька рівнів перевірки. Frontend не дає користувачу виконати очевидно неправильні дії. Backend повторно перевіряє дані і права доступу. База даних зберігає зв'язки та історію. Зовнішні сервіси виконують спеціалізовані задачі, такі як оплата або пошук адрес. Саме поєднання цих частин робить проект не просто набором сторінок, а інформаційною системою обліку оренди.

Висновки до розділу 3

У третьому розділі проєктні рішення з попереднього розділу показано на рівні програмної реалізації. Описано структуру серверної частини на FastAPI, поділ на маршрути, сервіси, схеми і моделі, а також наведено фрагменти коду для ключових сценаріїв.

Клієнтська частина на Next.js/React охоплює каталог, сторінку автомобіля, бронювання, оплату, профіль орендаря і кабінет власника. Окремо розглянуто роботу з PostgreSQL через SQLAlchemy та інтеграції зі Stripe, Mapbox, Twilio, файловим сховищем, SMTP і фоновими задачами.

Найбільше значення для предметної області мають авторизація, перевірка доступності автомобіля, життєвий цикл бронювання, облік платежів і фінансових значень. Саме ці фрагменти показують, що система реалізує не лише сторінки інтерфейсу, а й пов'язану бізнес-логіку обліку оренди.

4 ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Четвертий розділ охоплює три взаємопов'язані аспекти завершального етапу розробки: тестування системи, вимоги до апаратного та програмного забезпечення для розгортання, а також процедуру розгортання й запуску всіх компонентів інформаційної системи обліку оренди автомобілів фізичних осіб.

Тестування є обов'язковою умовою виявлення помилок до передачі системи в експлуатацію. Визначення апаратних і програмних вимог необхідне для коректного вибору середовища розгортання. Docker Compose використовується як практичний спосіб відтворити робоче середовище з потрібними залежностями.

4.1 Тестування інформаційної системи

Тестування виконувалося на рівні серверних модулів, оскільки саме серверна частина містить критичну бізнес-логіку: перевірку доступності автомобіля, розрахунок ціни, модель переходів статусів поїздки, авторизацію та обробку платежів. Для клієнтської частини проводилося ручне тестування ключових користувацьких сценаріїв.

Автоматизоване тестування реалізоване за допомогою бібліотеки pytest [19]. Усі тести зосереджені у каталозі tests і запускаються однією командою. Загальна кількість тестів — 119, розподілених у 13 тестових файлах. Кожен файл покриває окремий функціональний модуль системи.

Таблиця 4.1

Склад автоматизованих тестів

Файл тестів	Модуль	Кількість тестів
test_availability.py	Перевірка доступності автомобіля	12
test_state_machine.py	Переходи статусів поїздки	18
test_price_calculator.py	Розрахунок вартості оренди	15
test_authorization.py	Авторизація і ролі доступу	10
test_api.py	HTTP-маршрути (інтеграційні)	21
test_pickup_return.py	Видача та повернення авто	9
test_trip_extension.py	Продовження поїздки	8
test_guest_verification.py	Верифікація орендаря	7
test_delivery_locations.py	Локації доставки	6
test_email_service.py	Email-сповіщення	5
test_contact.py	Контактна форма	4
test_tasks.py	Фонові задачі	4
Всього		119

4.1.1 Юніт-тести логіки доступності

Перевірка доступності автомобіля є критичним компонентом: якщо два орендарі одночасно намагаються забронювати один автомобіль, має бути дозволений лише один коректний запит. Тести перевіряють функції check_vehicle_available та get_unavailable_vehicle_ids при різних комбінаціях статусів і дат поїздок.

Тести покривають такі сценарії: автомобіль без жодної поїздки є доступним; поїздка зі статусом booked або rental блокує перетин дат; скасована поїздка не блокує доступність; поїздка, що вже завершилася до початку запитуваного інтервалу, не впливає на результат.

```
def test_overlapping_active_trip_blocks_vehicle(self, db, seed_vehicle,
seed_guest):
    trip = Trip(
        vehicle_id=seed_vehicle.vehicle_id,
        guest_id=seed_guest.guest_id,
        trip_start=dt("2026-05-02"),
        trip_end=dt("2026-05-06"),
        trip_status=TripStatus.booked.value,
        ...
    )
    db.add(trip); db.flush()
    available = check_vehicle_available(
        db, seed_vehicle.vehicle_id, dt("2026-05-03"), dt("2026-05-07")
    )
    assert available is False  # перетин дат → недоступно
```

Лістинг 4.1 – Тест блокування автомобіля при перетині дат

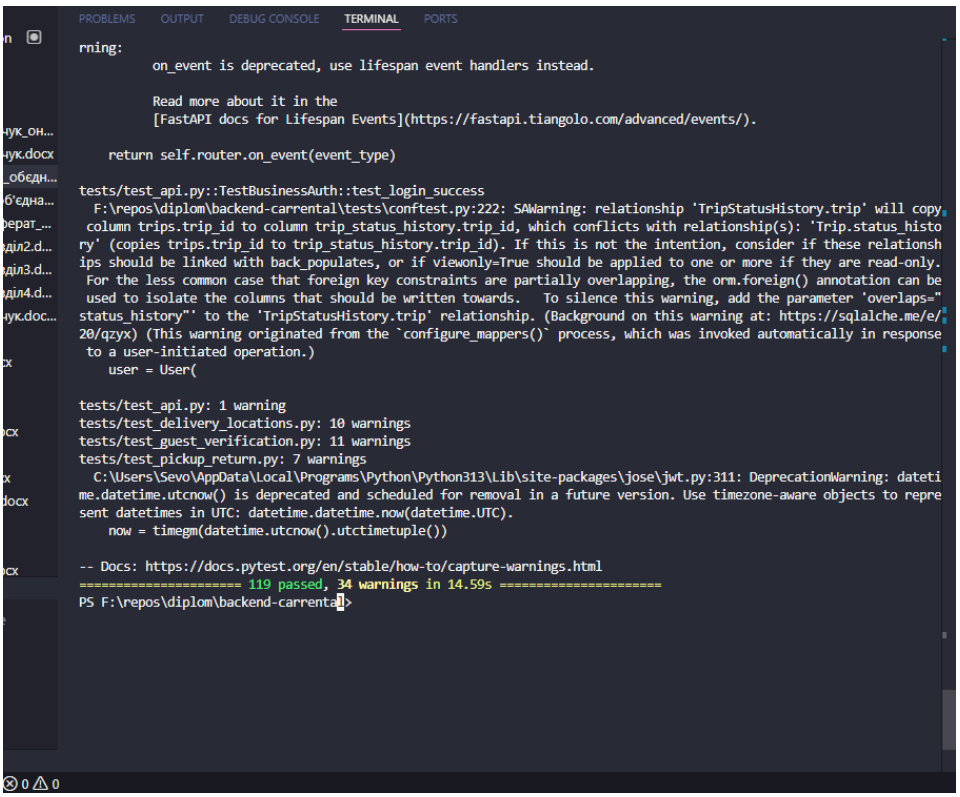

Рис. 4.1 Результат запуску тестів доступності (pytest -v test_availability.py)

4.1.2 Тести переходів статусів поїздки

Кожна поїздка проходить через фіксовану послідовність статусів. Неправильний перехід — наприклад, зі статусу «quote» одразу у «rental» — є логічною помилкою, яку система має відхиляти. Модель переходів реалізована у модулі app/services/trip_state_machine.py і тестується на 18 сценаріях.

Допустима послідовність статусів: quote, quote_approved_by_platform, wait_for_deposit, wait_for_payment, open_reservation, booked, rental, trip_ended, completed. Будь-який перехід поза дозволеним графом повертає HTTP 422 з описом помилки.

Таблиця 4.2

Сценарії тестування стан-машини поїздки

Сценарій переходу	Очікуваний результат
quote → quote_approved_by_platform	Успішно, статус змінено
quote → rental (пропуск стадій)	HTTP 422 — недозволений перехід
rental → quote (назад)	HTTP 422 — зворотний перехід заборонено
approve_trip() → встановлення timestamp	quote_approved_at заповнено
decline_trip() → причина відмови	decline_reason збережено
cancel_trip() з completed статусом	HTTP 422 — скасування після завершення

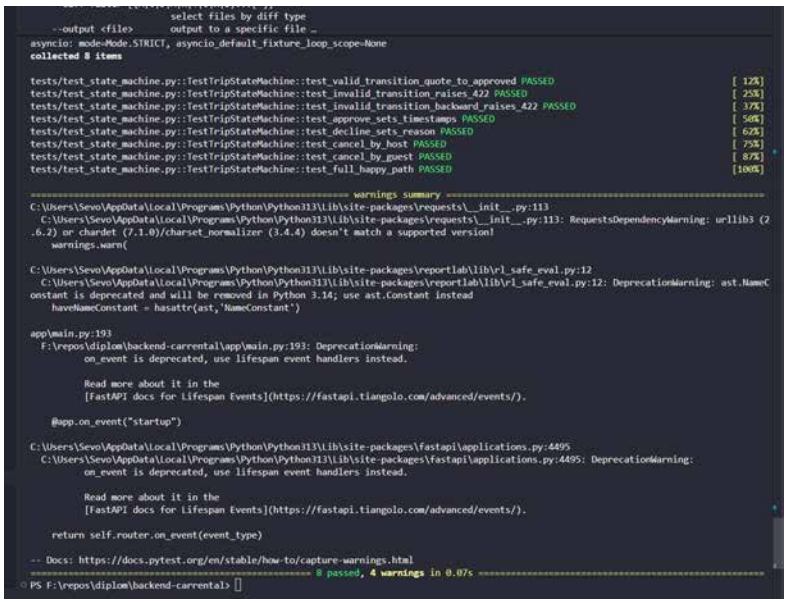

Рис. 4.2 Результат тестів переходів статусів (pytest -v test_state_machine.py)

4.1.3 Тести розрахунку ціни

Розрахунок вартості оренди є одним із найскладніших алгоритмів системи. Фінальна ціна формується з базової денної ставки, кількості днів, коефіцієнта доставки, вікового нарахування, відсотка комісії платформи, збору та податку. Тести перевіряють 15 комбінацій вхідних параметрів і порівнюють отримані значення з еталонними.

Тест базового розрахунку: 3 дні, 100 грн/день, 10% податок, 5 грн збір — дає очікувану загальну суму. Тест з коефіцієнтом: доставка за 10 км збільшує базову ціну. Тест із вірним порядком нарахувань гарантує, що комісія та податок не накладаються один на одного некоректно.

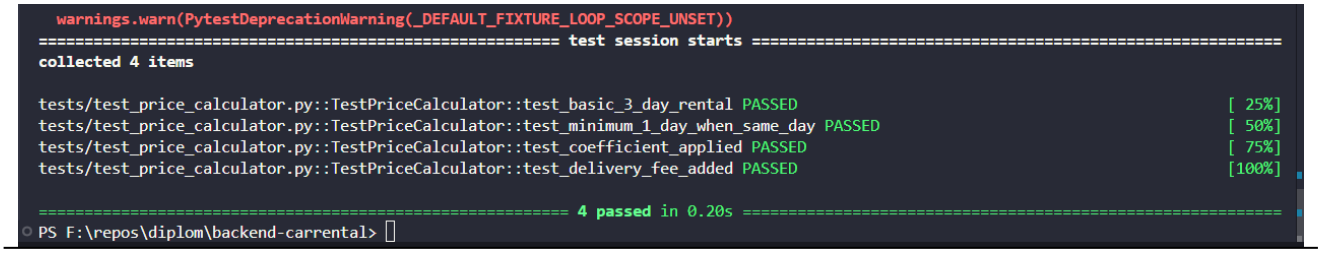

Рис. 4.3 Результат тестів калькулятора ціни (pytest -v test_price_calculator.py)

4.1.4 Інтеграційне тестування API

Файл test_api.py містить 21 інтеграційний тест, що перевіряє HTTP-маршрути через TestClient FastAPI. Запити виконуються до реального застосунку з тестовою базою даних SQLite in-memory, що дозволяє уникнути залежності від зовнішнього PostgreSQL.

Перевіряються реєстрація і вхід користувача, захист маршрутів токеном доступу, повернення 403 для неавторизованих запитів, коректні коди відповідей при валідних і невалідних даних, структура JSON-відповідей для публічного каталогу автомобілів.

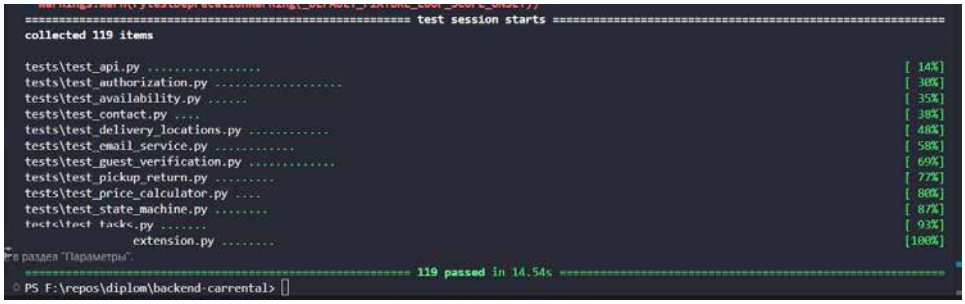

Рис. 4.4 Запуск усіх 119 тестів (pytest --tb=short)

4.1.5 Ручне тестування інтерфейсу

Ручне тестування frontend виконувалося за основними сценаріями кожної ролі. Для орендаря перевірялися: реєстрація, пошук автомобіля за датами та фільтрами, перегляд картки, створення бронювання, оплата через Stripe, перегляд статусу поїздки та залишення відгуку. Для власника: додавання автомобіля, встановлення доступності, підтвердження заявки, видача та повернення. Для адміністратора: верифікація документів, перегляд статистики, робота з претензіями.

Таблиця 4.3

Результати ручного тестування ключових сценаріїв

Сценарій	Роль	Результат
Реєстрація + OTP-підтвердження через Twilio	Орендар	✓ SMS отримано, акаунт активовано
Пошук авто за датами і містом	Орендар	✓ Фільтр працює, карта оновлюється
Перегляд картки автомобіля з фото	Орендар	✓ Галерея, характеристики, ціна
Бронювання + оплата (Stripe test mode)	Орендар	✓ Checkout відкрито, платіж пройшов
Підтвердження/відхилення заявки	Власник	✓ Статус змінено, орендар отримав email
Додавання автомобіля з фотографіями	Власник	✓ Фото завантажено на S3, авто активне
Видача авто (одометр + підпис)	Власник	✓ Статус rental, trip_start зафіксовано

Перегляд фінансового звіту за місяць	Власник	✓ Таблиця поїздок, PDF завантажено
Верифікація водійських прав	Адмін	✓ Документ переглянуто, статус змінено
Перегляд загальної статистики платформи	Адмін	✓ Дашборд з метриками відображено

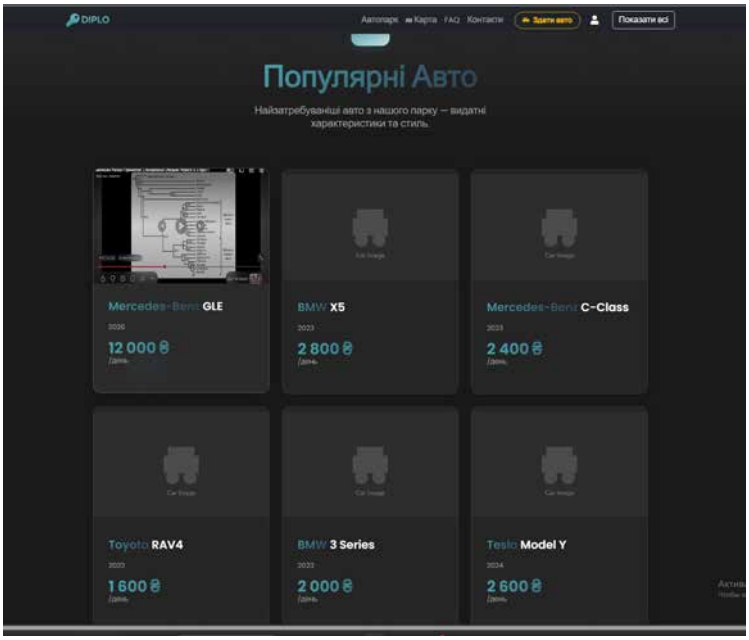

Рис. 4.5 Головна сторінка каталогу з фільтрами пошуку

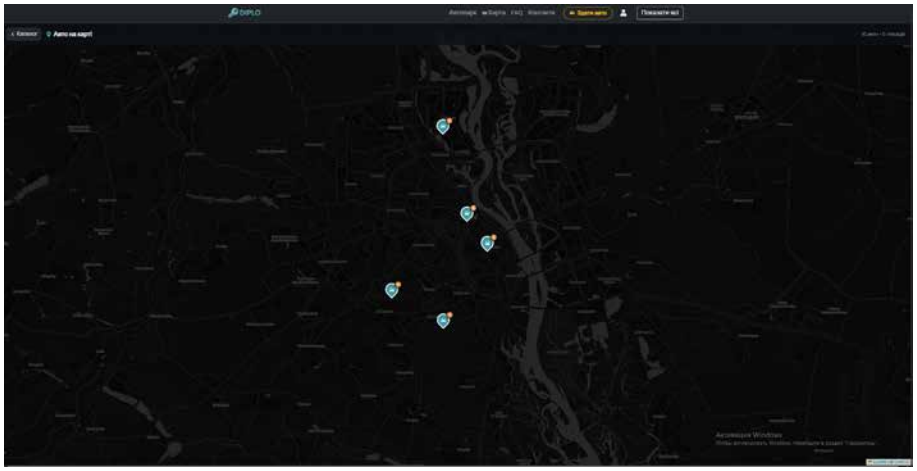

Рис. 4.6 Інтерактивна карта доступних автомобілів

Рис. 4.7 Кабінет власника: календар з датами активності автомобілів

4.2 Вимоги до апаратного та програмного забезпечення

Інформаційна система розгортається у контейнеризованому середовищі Docker [14]. Тому апаратні вимоги визначаються ресурсами, необхідними для одночасного запуску основних сервісів: PostgreSQL, FastAPI-сервера.

4.2.1 Вимоги до серверного середовища

Таблиця 4.4

Вимоги до серверного середовища

Компонент	Мінімальні вимоги	Рекомендовані вимоги
Процесор	2 ядра, x86_64 / ARM64	4 ядра, 2.5 ГГц+
Оперативна пам'ять	2 ГБ	4 ГБ+
Сховище	10 ГБ SSD	20 ГБ SSD (+ S3 для медіа)
ОС	Linux (Ubuntu 22.04 / Debian 12)	Ubuntu 22.04 LTS
Docker Engine	24.x+	26.x+
Docker Compose	v2.x	v2.27+
Python	3.11+	3.12 / 3.13
PostgreSQL	15+ (у Docker-контейнері)	16 Alpine (у Docker-контейнері)
Мережа	Відкритий порт 80/443 (HTTPS)	Nginx reverse proxy + SSL

4.2.2 Вимоги до клієнтського середовища

Frontend є веб-застосунком Next.js і доступний через браузер. Жодного встановлення не потрібно з боку орендаря або власника автомобіля. Вимоги до клієнта зводяться до наявності сучасного браузера та з'єднання з інтернетом.

Таблиця 4.5

Вимоги до клієнтського середовища

Компонент	Вимога
Браузер	Chrome 110+, Firefox 115+, Safari 16+, Edge 110+
Інтернет	Стабільне з'єднання від 5 Мбіт/с
Роздільна здатність	Від 375px (мобільна адаптивність підтримується)
JavaScript	Увімкнений (обов'язково)

4.2.3 Зовнішні сервіси та облікові записи

Для повноцінної роботи системи необхідно мати активні облікові записи у зовнішніх сервісах, ключі яких передаються через файл змінних середовища. Кожен сервіс відповідає за окрему функцію і підключається через API-ключ або секретний токен.

Таблиця 4.6

Зовнішні сервіси, необхідні для роботи системи

Сервіс	Призначення	Тип доступу
Stripe	Приймання онлайн-платежів, асинхронні підтвердження	Secret Key + Webhook Secret
Twilio	SMS-підтвердження OTP при реєстрації	Account SID + Auth Token
Mapbox	Геокодування адрес, карта локацій	Public Access Token
AWS S3	Зберігання фотографій автомобілів і документів	Access Key + Secret Key + Bucket
SMTP (Email)	Сповіщення орендарю та власнику про статус	Host + Port + Credentials
Redis	Черга фонових задач Celery	Redis URL (локально або хмара)

4.3 Розгортання та запуск системи

Система складається з двох основних компонентів: серверної частини FastAPI і клієнтської частини Next.js, кожен з яких упакований у Docker-образ [6, 9, 14]. Для локальної розробки та виробничого розгортання

використовується Docker Compose, який запускає залежні сервіси PostgreSQL і Redis разом із застосунком.

Схема розгортання наведена на рисунку 4.8. Зовнішній трафік проходить через Nginx reverse proxy і розподіляється між клієнтською частиною (порт 3000) та серверною частиною (порт 8000). Клієнтська частина звертається до серверної через REST API, а серверна взаємодіє з PostgreSQL, Redis та зовнішніми сервісами.

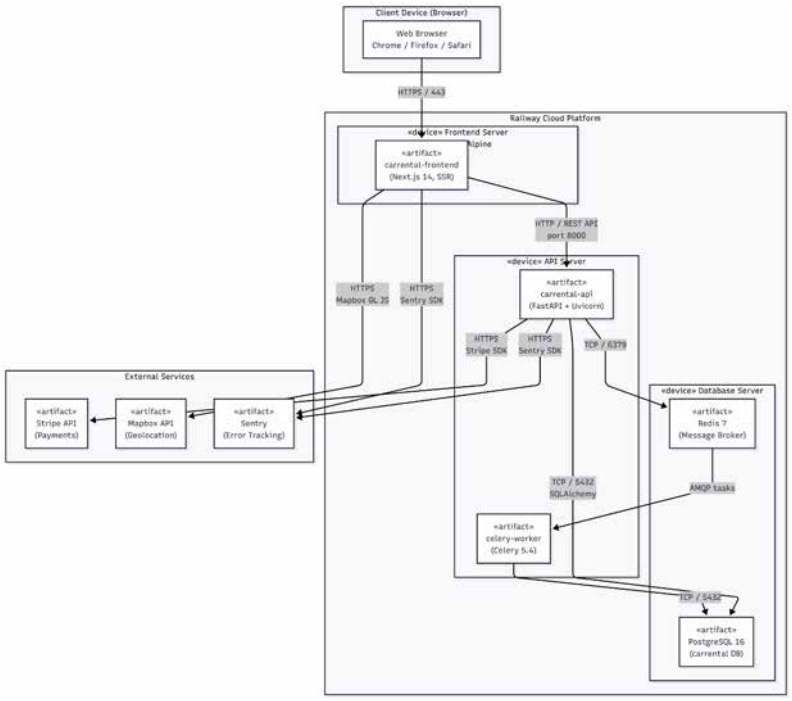

Рис. 4.8 Діаграма розгортання системи (Deployment diagram)

4.3.1 Кроки розгортання серверної частини

Розгортання серверної частини виконується у п'ять кроків: клонування репозиторію, налаштування змінних середовища, збірка Docker-образу, застосування міграцій бази даних і запуск сервера.

```
# 1. Клонувати репозиторій
git clone <repo-url>
cd backend-carrental

# 2. Створити файл .env (на основі .env.example)
cp .env.example .env
# Заповнити: DATABASE_URL, SECRET_KEY, STRIPE_SECRET_KEY,
#             TWILIO_*, MAPBOX_TOKEN, AWS_*, SMTP_*, REDIS_URL

# 3. Запустити всі сервіси (PostgreSQL + API)
docker compose up --build -d

# 4. Застосувати міграції Alembic
docker compose exec api alembic upgrade head
# 5. (Опціонально) Завантажити початкові дані
docker compose exec api python seed.py
```

Лістинг 4.2 – Послідовність розгортання серверної частини через Docker Compose

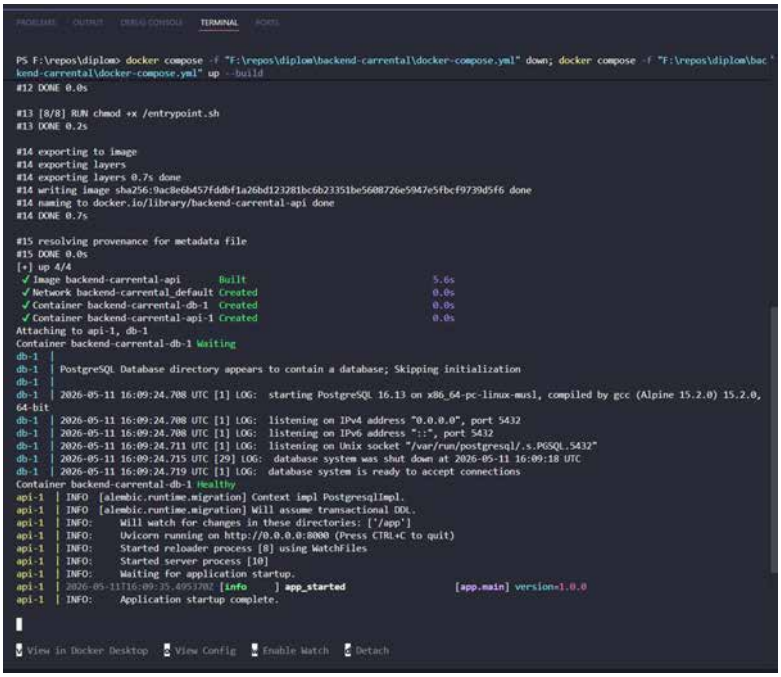

Рис. 4.9 Запуск контейнерів (docker compose up --build)

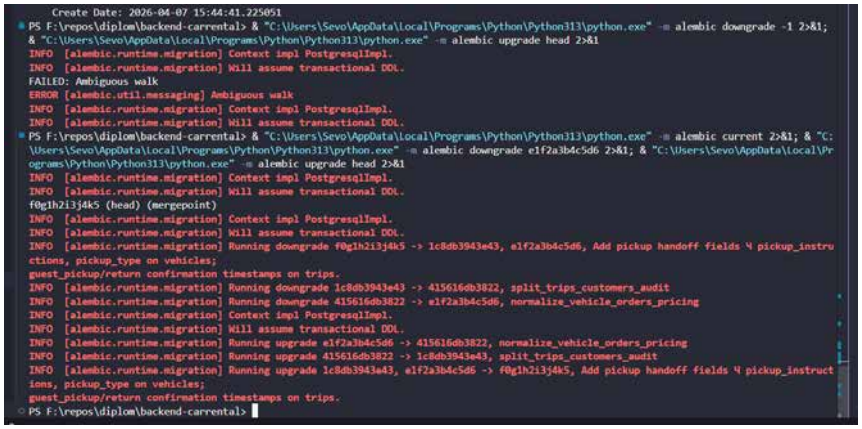

Рис. 4.10 Виконання міграцій бази даних (alembic upgrade head)

4.3.2 Кроки розгортання клієнтської частини

Клієнтська частина розгортається незалежно від серверної. Вона потребує змінної NEXT_PUBLIC_API_URL, що вказує на адресу API. У production-режимі Next.js збирає статичні файли і запускається через власний сервер або Nginx.

```
cd frontend


# 1. Встановити залежності
npm install


# 2. Налаштувати змінні середовища
# .env.local:
# NEXT_PUBLIC_API_URL=https://api.your-domain.com
# NEXT_PUBLIC_MAPBOX_TOKEN=pk.eyJ1...


# 3. Зібрати застосунок
npm run build


# 4. Запустити production-сервер
npm run start
# або через Docker:
docker build -t carrental-frontend .
docker run -p 3000:3000 carrental-frontend
```

Лістинг 4.3 – Збірка і запуск клієнтської частини

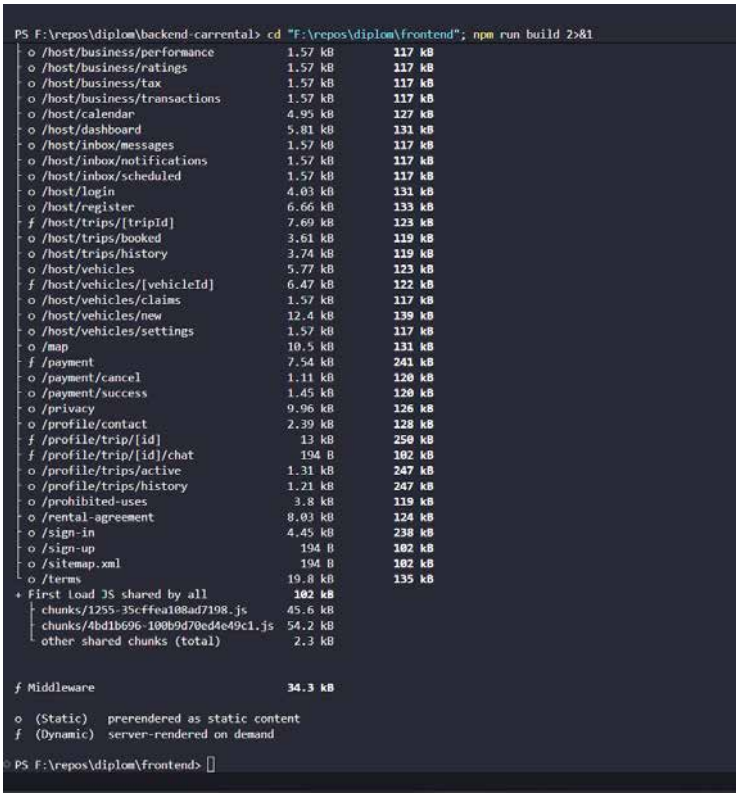

Рис. 4.11 Успішна збірка Next.js (npm run build)

4.3.3 Перевірка працездатності після розгортання

Після розгортання рекомендується перевірити декілька ключових точок системи, які підтверджують, що всі компоненти з'єднані і працюють коректно.

Таблиця 4.7

Контрольні перевірки після розгортання

Перевірка	**Очікуваний результат**
GET /health (backend)	HTTP 200: {"status": "ok"}
GET /docs (API)	Відображається Swagger UI зі списком маршрутів
Відкриття клієнтської частини у браузері	Завантажується головна сторінка каталогу
Реєстрація нового користувача	OTP надходить через Twilio SMS
Вхід і перехід у профіль	Токен доступу видано, дані профілю отримано
Пошук автомобіля	Результати з бази даних відображаються на карті Mapbox
docker compose ps	Всі контейнери у статусі «healthy»

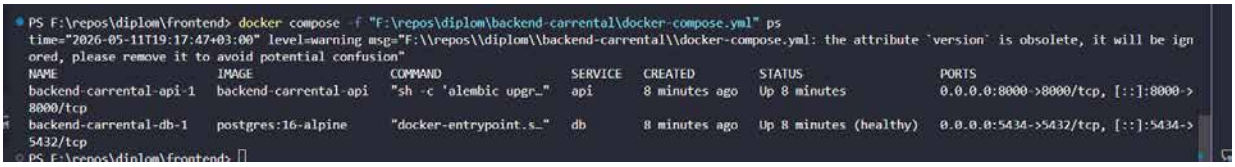

Рис. 4.12 Статус контейнерів (docker compose ps)

Висновки до розділу 4

У четвертому розділі показано, як перевіряється запускається розроблена система. Автоматизоване тестування охоплює 119 тестів у 13 модулях. Ручне тестування доповнює ці перевірки сценаріями орендаря, власника й адміністратора.

Також визначено мінімальні та рекомендовані вимоги до серверного і клієнтського середовища. Для запуску використовується Docker Compose, міграції бази даних і контрольні перевірки після розгортання. Такий опис достатній для демонстрації системи і для подальшого перенесення її в інше середовище.

ВИСНОВКИ

У бакалаврській кваліфікаційній роботі виконано проєктування та розробку інформаційної системи обліку оренди автомобілів фізичних осіб — від аналізу предметної галузі до реалізації і підготовки до розгортання.

Аналіз показав, що оренда між фізичними особами потребує єдиного середовища для обліку користувачів, автомобілів, заявок, оплат і відгуків. Порівняння з Turo, PROTO та Getmancar підтвердило: готові рішення не покривають локальний сценарій із повноцінною роллю приватного власника.

На етапі проєктування визначено клієнт-серверну архітектуру, структуру бази даних і функціональні модулі. PostgreSQL використано для збереження пов'язаних облікових даних, FastAPI — для серверної логіки, Next.js і React — для клієнтського інтерфейсу.

Реалізовано модулі каталогу автомобілів, бронювання, платежів, профілів користувачів, кабінету власника, звітності та адміністрування. Зовнішні сервіси підключено там, де це практично виправдано: Stripe для платежів, Twilio для SMS, Mapbox для геолокації, Docker для розгортання.

Автоматизовані тести охоплюють доступність автомобіля, розрахунок ціни, переходи статусів поїздки, авторизацію та API. Порядок запуску системи описано через Docker Compose.

Отримане рішення може слугувати основою для подальшого розвитку сервісу або як навчальний приклад клієнт-серверної системи. Ключовий результат — узгоджена модель обліку, в якій заявка, бронювання, оплата, статуси та відгуки пов'язані в єдиному процесі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Методичні рекомендації до виконання бакалаврської кваліфікаційної роботи для здобувачів ОС «Бакалавр» спеціальності 122 «Комп'ютерні науки». Київ : НУБіП України, 2026.

2. Shaheen S., Cohen A. Innovative Mobility Carsharing Outlook. Transportation Sustainability Research Center, University of California, Berkeley, 2020. URL: https://escholarship.org/uc/item/61q03282

3. Laudon K. C., Laudon J. P. Management Information Systems: Managing the Digital Firm. 16th ed. Pearson, 2020.

4. ISO/IEC/IEEE 29148:2018. Systems and software engineering — Life cycle processes — Requirements engineering. Geneva : ISO, 2018.

5. Statista. Car Rentals — Worldwide. URL: https://www.statista.com/outlook/mmo/shared-mobility/worldwide?currency=USD#revenue

6. FastAPI Documentation. URL: https://fastapi.tiangolo.com/

7. SQLAlchemy 2.0 Documentation. URL: https://docs.sqlalchemy.org/

8. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/

9. Next.js Documentation. URL: https://nextjs.org/docs

10. React Documentation. URL: https://react.dev/

11. Stripe Documentation. URL: https://docs.stripe.com/

12. Twilio Docs. URL: https://www.twilio.com/docs

13. Mapbox Documentation. URL: https://docs.mapbox.com/

14. Docker Documentation. URL: https://docs.docker.com/

15. Turo. Car sharing marketplace. URL: https://turo.com/

16. Bolt Drive / PROTO. Car rental and car-sharing service. URL: https://bolt.eu/

17. Getmancar. Український сервіс каршерингу. URL: https://getmancar.com.ua/

18. Booch G., Rumbaugh J., Jacobson I. The Unified Modeling Language User Guide. 2nd ed. Addison-Wesley, 2005.

19. pytest documentation. URL: https://docs.pytest.org/

20. Sommerville I. Software Engineering. 10th ed. Pearson, 2016.

21. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner's Approach. 9th ed. McGraw-Hill, 2020.

22. Пасічник В. В., Резниченко В. А. Організація баз даних та знань. — Київ : BHV, 2006. — 496 с.

23. Гайна Г. А. Проєктування баз даних : навч. посіб. — Київ : НТУУ «КПІ», 2010. — 152 с.

24. Лук'яненко І. Г., Краснікова Л. І. Економетрика та основи інформаційних систем. — Київ : Знання, 2008. — 580 с.

ДОДАТОК А

Повні лістинги програмного коду

В основному тексті пояснювальної записки наведено скорочені фрагменти програмного коду (Лістинги 3.1–3.12). У цьому додатку містяться повні реалізації двох ключових модулів: автомата станів поїздки та калькулятора вартості оренди.

А.1 Повний лістинг модуля автомата станів (app/services/trip_state_machine.py)

Лістинг 3.11 в основному тексті містив скорочений перелік переходів. Нижче наведено повний файл модуля: таблицю TRANSITIONS з усіма дозволеними переходами та всі функції-помічники (approve_trip, decline_trip, cancel_trip, mark_paid, start_rental, end_rental, complete_trip).

```
"""Trip state machine — mirrors Rails Trip status transitions."""


from datetime import datetime, timezone
from fastapi import HTTPException
from sqlalchemy.orm import Session
from app.models.trip import Trip
from app.models.enums import TripStatus


TRANSITIONS: dict[str, list[str]] = {
    TripStatus.quote.value: [
        TripStatus.quote_approved_by_platform.value,
        TripStatus.quote_cancelled_by_platform.value,
        TripStatus.quote_cancelled_by_host.value,
    ],
    TripStatus.quote_approved_by_platform.value: [
        TripStatus.wait_for_deposit.value,
        TripStatus.wait_for_payment.value,
        TripStatus.quote_cancelled_by_platform.value,
        TripStatus.cancelled_by_host.value,
```

```
    ],
    TripStatus.wait_for_deposit.value: [
        TripStatus.wait_for_payment.value,
        TripStatus.quote_cancelled_by_platform.value,
        TripStatus.cancelled_by_guest.value,
    ],
    TripStatus.wait_for_payment.value: [
        TripStatus.open_reservation.value,
        TripStatus.quote_cancelled_by_platform.value,
        TripStatus.cancelled_by_guest.value,
    ],
    TripStatus.open_reservation.value: [
        TripStatus.booked.value,
        TripStatus.cancelled_by_host.value,
        TripStatus.cancelled_by_guest.value,
    ],
    TripStatus.booked.value: [
        TripStatus.rental.value,
        TripStatus.cancelled_by_host.value,
        TripStatus.cancelled_by_guest.value,
    ],
    TripStatus.rental.value:     [TripStatus.trip_ended.value],
    TripStatus.trip_ended.value: [TripStatus.completed.value],
}


def transition_trip(db, trip, new_status, changed_by=None):
    current = trip.trip_status
    allowed = TRANSITIONS.get(current, [])
    if new_status not in allowed:
        raise HTTPException(
            status_code=422,
            detail=f"Cannot transition from {current!r} to {new_status!r}. "
                   f"Allowed: {allowed}",
        )
    trip.trip_status = new_status
    trip.trip_status_changed_at = datetime.now(timezone.utc)
    trip.trip_status_changed_by = changed_by
    db.commit(); db.refresh(trip)
    return trip


def approve_trip(db, trip, approved_by):
```

```
    trip = transition_trip(db, trip,
        TripStatus.quote_approved_by_platform.value, approved_by)
    trip.quote_approved_by = approved_by
    trip.quote_approved_at = datetime.now(timezone.utc)
    if trip.security_deposit and trip.security_deposit > 0:
        return transition_trip(db, trip,
            TripStatus.wait_for_deposit.value, approved_by)
    return transition_trip(db, trip,
        TripStatus.wait_for_payment.value, approved_by)


def decline_trip(db, trip, declined_by, reason):
    trip.quote_decline_reason = reason
    trip.quote_declined_by = declined_by
    trip.quote_declined_at = datetime.now(timezone.utc)
    return transition_trip(db, trip,
        TripStatus.quote_cancelled_by_platform.value, declined_by)


def cancel_trip(db, trip, cancelled_by, by_host=True):
    new_status = (TripStatus.cancelled_by_host.value if by_host
                  else TripStatus.cancelled_by_guest.value)
    return transition_trip(db, trip, new_status, cancelled_by)


def mark_paid(db, trip):
    trip = transition_trip(db, trip, TripStatus.open_reservation.value)
    return transition_trip(db, trip, TripStatus.booked.value)


def start_rental(db, trip, started_by=None):
    trip.trip_actual_start = datetime.now(timezone.utc)
    return transition_trip(db, trip, TripStatus.rental.value, started_by)


def end_rental(db, trip, ended_by=None):
    trip.trip_actual_end = datetime.now(timezone.utc)
    return transition_trip(db, trip, TripStatus.trip_ended.value, ended_by)


def complete_trip(db, trip, completed_by=None):
    return transition_trip(db, trip,
        TripStatus.completed.value, completed_by)
```

А.2 Повний лістинг калькулятора вартості оренди (app/services/price_calculator.py)

Нижче наведено повну реалізацію функції calculate_trip_price: розрахунок кількості днів, застосування коефіцієнта тривалості, страховий план, плата за доставку/повернення, платформні збори, податок, надбавки за вік водія та формування об'єкта PriceBreakdown.

```
"""Trip price calculation service."""


import math
from datetime import datetime
from sqlalchemy.orm import Session
from app.models.vehicle import Vehicle
from app.models.branch import Branch, BranchDailyRateCoefficient
from app.models.protection_plan import ProtectionPlan
from app.models.delivery_location import BranchDeliveryLocation
from app.models.platform import Platform
from app.schemas.vehicle import PriceBreakdown



def calculate_trip_price(
    db: Session,
    vehicle: Vehicle,
    pickup_date: datetime,
    return_date: datetime,
    protection_plan_id: int | None = None,
    pickup_delivery_location_id: int | None = None,
    return_delivery_location_id: int | None = None,
) -> PriceBreakdown:
    """Calculate full price breakdown for a trip."""


    delta = return_date - pickup_date
    number_of_days = max(1, math.ceil(delta.total_seconds() / 86400))


    branch: Branch = vehicle.branch
    daily_rate = vehicle.vehicle_daily_rate
```

```
    coefficient = 1.0
    coefficients = (
        db.query(BranchDailyRateCoefficient)
        .filter(
            BranchDailyRateCoefficient.branch_id == branch.branch_id,
            BranchDailyRateCoefficient.is_active == True,
            BranchDailyRateCoefficient.from_value <= number_of_days,
            BranchDailyRateCoefficient.to_value   >= number_of_days,
        )
        .first()
    )
    if coefficients:
        coefficient = coefficients.branch_daily_rate_coefficient


    daily_amount = daily_rate * coefficient
    total_rent = round(daily_amount * number_of_days, 2)


    protection_amount = 0.0
    if protection_plan_id:
        plan = db.query(ProtectionPlan).filter(
            ProtectionPlan.protection_plan_id == protection_plan_id
        ).first()
        if plan:
            if plan.protection_plan_value_type == "percentage":
                protection_amount = round(
                    total_rent * plan.protection_plan_value / 100, 2)
            else:
                protection_amount = round(
                    plan.protection_plan_value * number_of_days, 2)


    delivery_fee = 0.0
    for loc_id in [pickup_delivery_location_id,
                   return_delivery_location_id]:
        if loc_id:
            bdl = (
                db.query(BranchDeliveryLocation)
                .filter(
                    BranchDeliveryLocation.branch_id  == branch.branch_id,
                    BranchDeliveryLocation.delivery_location_id == loc_id,
                    BranchDeliveryLocation.is_active  == True,
                )
```

```
                .first()
            )
            if bdl:
                delivery_fee += bdl.delivery_fee_amount


    host     = branch.host
    platform = host.platform if host else None
    trip_fee_pct      = platform.trip_fee_percentage        if platform else
0.0
    processing_fee_pct= platform.processing_fees_percentage if platform else
0.0
    admin_charge      = platform.administrative_charge_amount if platform
else 0.0


    subtotal       = total_rent + protection_amount + delivery_fee
    trip_fee       = round(subtotal * trip_fee_pct / 100, 2)
    processing_fee = round(subtotal * processing_fee_pct / 100, 2)


    tax_amount = round(subtotal * branch.tax_percentage / 100, 2)         if
branch.tax_percentage else 0.0
    surcharge  = branch.surcharge_amount or 0.0


    age_charge = 0.0
    if vehicle.guest_age_restriction_type == "surcharge":
        age_charge = (vehicle.guest_age_restriction_charge_amount
                      * number_of_days)


    security_deposit = vehicle.security_deposit or 0.0


    total = round(
        total_rent + protection_amount + delivery_fee
        + tax_amount + surcharge + trip_fee + processing_fee
        + admin_charge + age_charge + security_deposit,
        2,
    )
    return PriceBreakdown(
        daily_rate=daily_rate,
        number_of_days=number_of_days,
        daily_rate_coefficient=coefficient,
```

```
        total_rent=total_rent,
        protection_amount=protection_amount,
        delivery_fee=delivery_fee,
        tax_amount=tax_amount,
        surcharge=surcharge,
        trip_fee=trip_fee,
        processing_fee=processing_fee,
        administrative_charge=admin_charge,
        guest_age_restriction_charge=age_charge,
        security_deposit=security_deposit,
        total=total,
    )
```

ДОДАТОК Б

Конфігурація та розгортання системи

В основному тексті (розділ 4.3) наведено скриншоти терміналу з результатами виконання docker compose up --build та alembic upgrade head. У цьому додатку містяться самі конфігураційні файли розгортання.

Б.1 docker-compose.yml — конфігурація Docker Compose

Файл визначає два сервіси: db (PostgreSQL 16 Alpine) та api (FastAPI). Сервіс db включає healthcheck; api залежить від db та автоматично виконує міграції при старті.

```
version: "3.9"


services:
  db:
    image: postgres:16-alpine
    environment:
      POSTGRES_DB: carrental
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
    ports:
      - "5434:5432"
    volumes:
      - pgdata:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres -d carrental"]
      interval: 5s
      timeout: 5s
      retries: 10


  api:
    build: .
    ports:
      - "8000:8000"
    env_file:
      - .env
    environment:
```

```
      DATABASE_URL: postgresql://postgres:postgres@db:5432/carrental
    depends_on:
      db:
        condition: service_healthy
    volumes:
      - .:/app
    command: >
      sh -c "alembic upgrade head &&
             uvicorn app.main:app --host 0.0.0.0 --port 8000 --reload"


volumes:
  pgdata:
```

Б.2 Dockerfile — образ серверної частини

Образ базується на python:3.12-slim, встановлює системні залежності (libpq-dev, gcc), копіює requirements.txt, встановлює пакети та запускає застосунок через entrypoint.sh.

```
FROM python:3.12-slim

WORKDIR /app

ENV PYTHONPATH=/app

RUN apt-get update && apt-get install -y --no-install-recommends     libpq-
dev gcc &&     rm -rf /var/lib/apt/lists/*

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 8000

COPY entrypoint.sh /entrypoint.sh
RUN chmod +x /entrypoint.sh
```

```
CMD ["/entrypoint.sh"]
```

Б.3 entrypoint.sh — скрипт запуску контейнера

Скрипт послідовно виконує: міграції Alembic, заповнення бази початковими даними (seed.py) та запуск uvicorn на відповідному порті.

```
#!/bin/sh
set -e

echo "Running database migrations..."
alembic upgrade head
echo "Migrations completed successfully."

echo "Seeding database (skipping if already seeded)..."
python seed.py || echo "Seeding skipped (data may already exist)."

echo "Starting application on port ${PORT:-8000}..."
exec uvicorn app.main:app --host 0.0.0.0 --port ${PORT:-8000}
```

Б.4 requirements.txt — залежності серверної частини

Повний перелік Python-пакетів із зафіксованими версіями для відтворюваного розгортання.

```
fastapi==0.115.6
uvicorn[standard]==0.34.0
sqlalchemy==2.0.36
alembic==1.14.1
psycopg2-binary==2.9.10
pydantic==2.10.4
pydantic-settings==2.7.1
python-jose[cryptography]==3.3.0
passlib[bcrypt]==1.7.4
bcrypt==4.0.1
python-multipart==0.0.20
stripe==11.4.1
boto3==1.36.4
```

```
httpx==0.28.1
pytest==8.3.4
pytest-asyncio==0.25.0
python-dotenv==1.0.1
email-validator==2.2.0
celery[redis]==5.4.0
redis==5.2.1
slowapi==0.1.9
structlog==24.4.0
reportlab==4.2.5
Pillow==11.1.0
```

ДОДАТОК В

SQL-визначення ключових таблиць бази даних

В основному тексті (Рисунок 2.2) структуру бази даних наведено у вигляді ER-діаграми. У цьому додатку містяться DDL-визначення (CREATE TABLE) чотирьох ключових таблиць: users, vehicles, trips та trip_payments. Повна схема містить 47 таблиць і знаходиться у файлі schema_dump.sql репозиторію проєкту.

В.1 Таблиця users — облікові записи користувачів

```
CREATE TABLE public.users (
    id                  bigint NOT NULL,
    email               character varying NOT NULL,
    hashed_password     character varying NOT NULL,
    role                integer NOT NULL,
    phone_number        character varying,
    is_active           boolean NOT NULL,
    representable_type  character varying,
    representable_id    bigint,
    created_at          timestamp without time zone DEFAULT now() NOT NULL,
    updated_at          timestamp without time zone DEFAULT now() NOT NULL,
    last_sign_in_at     timestamp without time zone
);
```

В.2 Таблиця vehicles — транспортні засоби

```
CREATE TABLE public.vehicles (
    vehicle_id                          bigint NOT NULL,
    branch_id                           bigint NOT NULL,
    vin                                 character varying(20) NOT NULL,
    license_plate_number                character varying(10) NOT NULL,
    vehicle_maker_id                    bigint NOT NULL,
    vehicle_model_id                    bigint NOT NULL,
    vehicle_production_year             integer NOT NULL,
    vehicle_body_type_id                bigint NOT NULL,
    vehicle_fuel_type_id                bigint NOT NULL,
    vehicle_status                      character varying(50) NOT NULL,
    platform_vehicle_status             character varying(50) NOT NULL,
```

```
    vehicle_daily_rate                    double precision NOT NULL,
    security_deposit                      double precision,
    distance_limit_per_day                bigint,
    msrp                                  bigint NOT NULL,
    engine_horsepower                     bigint NOT NULL,
    odometer                              bigint NOT NULL,
    guest_age_restriction_min_age         bigint NOT NULL,
    guest_age_restriction_max_age         bigint NOT NULL,
    guest_age_restriction_type            character varying(50) NOT NULL,
    guest_age_restriction_charge_amount   double precision NOT NULL,
    rent_min_days                         bigint NOT NULL,
    is_instant_booking                    boolean NOT NULL,
    is_one_way_trip                       boolean NOT NULL,
    is_active                             boolean NOT NULL,
    is_delivery_available                 boolean DEFAULT false NOT NULL,
    delivery_price                        double precision DEFAULT 0 NOT NULL,
    next_trip_preparation_hours           integer NOT NULL,
    created_at  timestamp without time zone DEFAULT now() NOT NULL,
    updated_at  timestamp without time zone DEFAULT now() NOT NULL,
    deleted_at  timestamp without time zone
);
```

B.3 Таблиця trips — поїздки та заявки на оренду

```
CREATE TABLE public.trips (
    trip_id              bigint NOT NULL,
    trip_start           timestamp without time zone NOT NULL,
    trip_end             timestamp without time zone NOT NULL,
    pickup_branch_id     bigint,
    return_branch_id     bigint,
    guest_id             bigint,
    vehicle_id           bigint,
    protection_plan_id   bigint,
    pickup_delivery_location_id   bigint,
    return_delivery_location_id   bigint,
    vehicle_daily_rate   double precision NOT NULL,
    vehicle_daily_amount double precision NOT NULL,
    security_deposit     double precision,
    delivery_fee_amount  double precision,
    protection_plan_value         double precision,
    protection_plan_value_type    character varying,
```

```
    trip_fee_percentage           double precision,
    processing_fees_percentage    double precision,
    administrative_charge_amount  double precision,
    administrative_charge         double precision NOT NULL,
    tax_percentage       double precision,
    surcharge_amount     double precision,
    late_return_fee_amount        double precision NOT NULL,
    guest_age_restriction_charge_amount double precision,
    trip_status          character varying(50) NOT NULL,
    trip_status_changed_by  bigint,
    trip_status_changed_at  timestamp without time zone,
    trip_actual_start    timestamp without time zone,
    trip_actual_end      timestamp without time zone,
    deposit_status       character varying(20) DEFAULT ''pending'' NOT NULL,
    platform_commission_value     double precision NOT NULL,
    legal_entity_commission_value double precision NOT NULL,
    quote_approved_by    bigint,
    quote_approved_at    timestamp without time zone,
    quote_declined_by    bigint,
    quote_declined_at    timestamp without time zone,
    quote_decline_reason character varying,
    created_at  timestamp without time zone DEFAULT now() NOT NULL,
    updated_at  timestamp without time zone DEFAULT now() NOT NULL,
    deleted_at  timestamp without time zone
);
```

ДОДАТОК Г

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

ДЕКЛАРАЦІЯ ЗДОБУВАЧА

Я, Мулярчук Сергій Сергійович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота (проєкт) на тему «Інформаційна система обліку оренди автомобілів фізичних осіб» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):
    - [ ] інструменти генеративного ШІ не використовувалися.
    - [+] інструменти ШІ ChatGPT, Claude були використані для:
        - [+] перекладу іншомовних джерел;
        - [ ] стилістичного редагування та перевірки граматики;
        - [+] допомоги у написанні/відлагодженні програмного коду;
        - [+] інше: пошуку статей по темі бакалаврської роботи

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«22» травня 2026 р. _____ **Сергій МУЛЯРЧУК**
(підпис)