Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп’ютерних наук

Допущено до захисту
Завідувач кафедри
комп’ютерних наук,
доктор технічних наук, професор

Андрій БОНДАРЧУК
«01» червня 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня «бакалавр»
Спеціальність 122 Комп’ютерні науки
Освітня програма 122.00.01 Інформатика

Тема: ПЛАТФОРМА УПРАВЛІННЯ АВТОПАРКОМ ТРАНСПОРТНОЇ КОМПАНІЇ

Виконав
Студент групи ІНб-2-22-4.0д
Рева Віталій Михайлович
_____
(підпис)

Науковий керівник
к.т.н., доцент,
Негоденко О.В.
_____
(підпис)

Київ – 2026

Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп'ютерних наук

«Затверджую»
Завідувач кафедри комп’ютерних наук,
доктор технічних наук, професор
Андрій БОНДАРЧУК
« 31» жовтня 2025 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА
«Платформа управління автопарком транспортної компанії»

Виконавець – студент групи ІНб-2-22-4.0д,
спеціальності 122 Комп’ютерні науки,
освітньої програми 122.00.01 Інформатика
Рева Віталій Михайлович

1. Вихідні дані: Платформа управління автопарком транспортної компанії. Технологічний стек: Python 3.x, Django 5.2, Django REST Framework, PostgreSQL, pandas, python-docx. Автентифікація: JWT-токени.

2. Основні завдання включають: проаналізувати предметну область управління автопарком; порівняти існуючі програмні рішення; визначити вимоги до платформи; розробити архітектуру та структуру бази даних; реалізувати платформу; провести тестування платформи управління автопарком транспортної компанії.

3. Пояснювальна записка: Обсяг близько 52 стор. формату А4 комп’ютерного набору з дотриманням вимог стандарту і методичних рекомендацій кафедри.

4. Графічні матеріали: Комп'ютерна презентація доповіді.

5. Додатки: Фрагменти вихідного програмного коду (Лістинг програмного коду платформи управління автопарком транспортної компанії), таблиця ендпоінтів, таблиця результатів тестування системи.

6. Строк подання роботи на кафедру: «01» червня 2026 р.

Науковий керівник	Виконавець:
к.т.н., доцент	
_____Негоденко О.В.	_____ Рева В.М.
_____ дата	_____ дата

Календарний план

№	Етап виконання роботи	Термін виконання	Примітка
1	Затвердження теми кваліфікаційної роботи	31.10.25	виконано
2	Аналіз проблеми, вивчення аналогів, формування цілей і завдань	01.11.25 – 11.01.26	виконано
3	Аналіз предметної області та існуючих програмних рішень для управління автопарком	26.01.26 – 15.03.26	виконано
4	Проєктування архітектури платформи та структури бази даних	16.03.26 – 31.03.26	виконано
5	Розробка платформи управління автопарком (моделі, сервісний шар, REST API)	01.04.26 – 15.04.26	виконано
6	Тестування платформи та виправлення помилок	16.04.26 – 30.04.26	виконано
7	Впровадження та оцінка результатів платформи	01.05.26 – 15.05.26	виконано
8	Підготовка пояснювальної записки, графічних матеріалів, додатків.	25.05.25 – 12.06.26	виконано
9	Захист кваліфікаційної роботи	18.06.26	

«Затверджую»
Науковий керівник
к.т.н., доцент, Негоденко О.В.

Індивідуальний план склав
Рева В.М.

РЕФЕРАТ

Кваліфікаційна робота бакалавра: с. 52, рис. 9, табл. 9, 36 посилань.

Актуальність. У сучасних умовах розвитку транспортно-логістичної галузі України управління автопарком набуває стратегічного значення. Традиційні методи ведення обліку вже не відповідають вимогам сьогодення, що зумовлює необхідність розробки спеціалізованої платформи для автоматизації процесів обліку та управління.

Об'єкт дослідження – серверна веб-платформа на основі REST API для управління автопарком транспортної компанії.

Предмет дослідження – методи та засоби розробки серверної платформи на основі Django REST Framework для управління транспортними засобами водіями, рейсами та фінансовими розрахунками.

Мета роботи – розробити та реалізувати програмну платформу управління автопарком транспортної компанії з використанням сучасних веб-технологій.

Завдання:

1. Проаналізувати предметну область управління автопарком.
2. Порівняти існуючі програмні рішення для управління автопарком.
3. Визначити вимоги до платформи для управління автопарком.
4. Спроєктувати архітектуру та структуру бази даних платформи для управління автопарком.
5. Реалізувати платформу для управління автопарком.
6. Провести тестування платформи для управління автопарком.

Основні результати. Розроблено платформу управління автопарком транспортної компанії на базі Python, Django 5.2, Django REST Framework та PostgreSQL. Платформа реалізує повний обліковий цикл: управління транспортними засобами з автоматичним відстеженням технічного обслуговування, водіями, клієнтами, замовленнями, рейсами, паливом та платежами. Розроблено 6 сервісних класів з повною бізнес-логікою та систему обробки помилок із 26 кодами виду XX_YY. Успішно пройдено 19 тест-кейсів.

Робота пройшла апробацію на XIII Всеукраїнській науково-практичній конференції молодих учених (Київ, 14 травня 2026 р.).

Практичне значення дослідження. Результати роботи можуть бути використані транспортними компаніями малого та середнього масштабу для автоматизації процесів обліку та управління автопарком, зменшення операційних витрат і підвищення ефективності логістичних операцій.

Ключові слова: управління автопарком, транспортна компанія, Python, Django REST Framework, PostgreSQL, веб-платформа, REST API, автоматизація обліку, бізнес-логіка.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API (Application Programming Interface) – інтерфейс програмування застосунків; набір правил і протоколів для взаємодії між програмними компонентами.

CI/CD (Continuous Integration / Continuous Delivery) – практика безперервної інтеграції та безперервного постачання програмного забезпечення.

CRUD (Create, Read, Update, Delete) – чотири базові операції роботи з даними: створення, читання, оновлення та видалення.

DRF (Django REST Framework) – бібліотека для розробки REST API на базі фреймворку Django.

ЄДРПОУ – Єдиний державний реєстр підприємств і організацій України; унікальний ідентифікатор юридичної особи.

ФОП – фізична особа-підприємець; організаційно-правова форма ведення бізнесу в Україні.

ІПН – індивідуальний податковий номер; унікальний ідентифікатор фізичної особи як платника податків.

JWT (JSON Web Token) – стандарт для створення токенів доступу з цифровим підписом, що використовується для автентифікації та авторизації.

GPS (Global Positioning System) – глобальна система супутникової навігації для визначення місцезнаходження об'єктів.

HTTP (HyperText Transfer Protocol) – протокол передачі гіпертекстових документів, що є основою обміну даними у вебі.

ORM (Object-Relational Mapping) – технологія відображення об'єктів програмного коду на таблиці реляційної бази даних.

PostgreSQL – об'єктно-реляційна система управління базами даних із відкритим вихідним кодом.

REST (Representational State Transfer) – архітектурний стиль проєктування розподілених систем та вебсервісів.

Swagger – інструмент для автоматичної генерації та відображення інтерактивної документації REST API.

TMS (Transportation Management System) – система управління транспортною логістикою.

ТО – технічне обслуговування транспортного засобу.

VIN (Vehicle Identification Number) – унікальний ідентифікаційний номер транспортного засобу.

Cron – планувальник завдань в операційних системах Unix/Linux, що дозволяє автоматично запускати команди або скрипти за розкладом.

Docker – платформа для контейнеризації застосунків, що забезпечує однорідність середовища виконання.

Django – високорівневий вебфреймворк мови Python, що дотримується принципу «все включено».

Endpoint (ендпоінт) – конкретна URL-адреса REST API, що відповідає за обробку певного типу запитів.

Міграція бази даних – механізм контрольованої зміни схеми бази даних із збереженням наявних даних.

Сервісний шар – архітектурний компонент, що містить бізнес-логіку застосунку та забезпечує її незалежність від рівня представлення даних.

Транзакція – неподільна послідовність операцій із базою даних, що виконується повністю або не виконується взагалі.

ЗМІСТ

ВСТУП

Обґрунтування актуальності теми. У сучасних умовах розвитку транспортно-логістичної галузі України управління автопарком набуває стратегічного значення. Зростання обсягів внутрішніх та міжнародних вантажоперевезень, ускладнення логістичних ланцюгів через геополітичні фактори, постійне підвищення вартості пального, запасних частин та послуг технічного обслуговування, а також дефіцит кваліфікованих кадрів змушують транспортні компанії шукати ефективні інструменти оптимізації витрат і підвищення оперативності прийняття рішень. За даними Міністерства інфраструктури України та Державної служби статистики, у 2024-2025 роках автомобільні вантажні перевезення становили понад 65% від загального обсягу вантажообігу в країні, при цьому середній вік вантажних автомобілів перевищує 12-14 років, а витрати на паливо та технічне обслуговування складають близько 40-55 % від загальної собівартості перевезень [1].

Традиційні методи управління автопарком є ручне ведення журналів, таблиць Excel, паперових путівок та окремих програм для бухгалтерського обліку вже не відповідають вимогам сьогодення. Вони призводять до численних помилок в обліку пробігу, витрат пального, статусів транспортних засобів і водіїв, несвоєчасного планування технічного обслуговування, втрати контролю над платежами від клієнтів та накопичення заборгованостей. У результаті компанії стикаються з простоями техніки, перевитратами ресурсів, штрафами за порушення термінів доставки та зниженням рівня задоволеності клієнтів [2].

Актуальність теми дипломної роботи зумовлена кількома ключовими факторами:

- стрімким розвитком цифровізації в логістиці та появою концепцій «розумного автопарку» (Smart Fleet Management), що включають моніторинг у реальному часі, прогнозування витрат та автоматизацію рутинних процесів [3];

- необхідністю адаптації українських транспортних компаній до європейських стандартів обліку (зокрема, вимог до прозорості витрат пального та екологічних показників);
- відсутністю на ринку доступних, гнучких та повністю адаптованих під специфіку українських підприємств рішень, які б поєднували облік транспорту, водіїв, замовлень, поїздок, палива, технічного обслуговування та фінансових розрахунків в одній системі [4];
- можливістю суттєвого зниження операційних витрат (за оцінками експертів галузі приблизно на 15-35 %) завдяки автоматизації та зменшенню людського фактора [5].

Саме тому розробка спеціалізованої платформи управління автопарком є не лише актуальною, а й економічно виправданою задачею для транспортних компаній середнього та малого сегменту.

Об'єкт дослідження – серверна веб-платформа на основі REST API для управління автопарком транспортної компанії.

Предмет дослідження – методи та засоби розробки серверної платформи на основі Django REST Framework для автоматизації процесів управління транспортними засобами водіями, рейсами та фінансовими розрахунками.

Мета дипломної роботи – автоматизація процесів обліку та управління автопарком транспортної компанії завдяки розробці платформи управління автопарком.

Методи дослідження – розробити та реалізувати програмну платформу управління автопарком транспортної компанії з використанням сучасних веб-технологій.

Для досягнення поставленої мети визначено такі завдання:

- проаналізувати предметну область у сфері управління автопарком транспортної компанії;
- порівняти існуючі рішення програмного забезпечення для управління автопарком на комерційній та безоплатній основі;

- визначити вимоги до програмного забезпечення;
- розробити архітектуру платформи для управління автопарком транспортної компанії;
- спроєктувати та розробити платформу для управління автопарком транспортної компанії;
- протестувати платформу для управління автопарком транспортної компанії

Робота складається зі змісту, вступу, трьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг роботи становить близько 52 сторінок, містить 9 рисунків, 9 таблиць, 35 джерел літератури.

РОЗДІЛ 1

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1 Аналіз бізнес-процесів управління автопарком транспортної компанії

Транспортна компанія, що займається вантажними перевезеннями, досить складна організаційна структура, де одночасно відбувається безліч взаємопов'язаних процесів. Щоб усе працювало злагоджено, потрібна чітка система обліку: хто з водіїв вільний, яке авто де знаходиться, яке замовлення вже оплачено, а яке досі не оплачено. Без нормального інструменту для цього менеджери витрачають купу часу на те, щоб просто з'ясувати поточний стан справ.

Розглянемо основні напрями, з яких складається управління автопарком транспортної компанії.

Перш за все, облік самого транспорту. Кожен автомобіль має свій паспорт у системі: марка й модель, держномер, VIN-код, поточний пробіг, норма витрати пального. Окремо відстежується стан технічного обслуговування: для кожного авто визначається інтервал між плановими ТО у кілометрах, і коли машина наближається до порогового значення, система сигналізує про необхідність огляду. Усі проведені ремонти та техроботи фіксуються з датою і вартістю [6].

Управління водіями включає базу даних водіїв, що містить усю необхідну інформацію: ПІБ, номер телефону, серія й номер посвідчення, ІПН, короткий опис досвіду роботи. Кожному водію присвоюється один зі статусів: «вільний», «у рейсі» або «вихідний», який змінюється автоматично при створенні або завершенні поїздки. Окремо відстежуються дати закінчення відпусток та вихідних, після яких статус скидається назад у «вільний» [7].

Управління клієнтами та замовленнями. Компанія може працювати з клієнтами трьох категорій: звичайними фізичними особами, ФОПами та

юридичними особами. Для кожної категорії існує свій набір обов'язкових реквізитів: фізособи й ФОПи ідентифікуються за ІПН, а компанії: за кодом ЄДРПОУ та назвою організації. Замовлення фіксує найменування вантажу, ціну перевезення, адресу доставки та унікальний ідентифікатор для зв'язку з банківським платежем.

Управління рейсами. Рейс, що є центральна операційна одиниця в роботі транспортної компанії. При його створенні до замовлення прив'язується конкретний водій і транспортний засіб, визначаються точки відправлення та призначення. Після підтвердження рейсу статуси водія й авто автоматично змінюються на «у рейсі». Під час виконання фіксуються пройдена відстань і фактичні витрати пального, а після завершення система порівнює планові та фактичні показники [8].

Управління паливом. Для кожного рейсу ведеться журнал заправок: кількість залитих літрів, вартість заправки, дата й час. На основі норми витрати пального авто та відстані рейсу система розраховує плановий показник і зіставляє його з фактичним. Це дозволяє оперативно виявляти надвитрату або можливі зловживання.

Фінансовий облік і обробка платежів. Банківський реєстр платежів надходить у вигляді електронної таблиці. Система автоматично зіставляє ідентифікатори оплати з відповідними замовленнями та проставляє їм статус «оплачено». Якщо для якогось запису в реєстрі відповідне замовлення не знайдено, він зберігається окремо як «неоплачений» для ручного опрацювання менеджером.

1.2 Недоліки традиційних підходів до обліку та управління автопарком транспортної компанії

На практиці більшість невеликих та середніх транспортних компаній досі ведуть облік автопарку «по-старому»: у паперових журналах, таблицях Excel або розрізнених локальних програмах. На перший погляд це здається

зручним і дешевим рішенням, але насправді такий підхід породжує низку серйозних проблем, які з часом лише накопичуються [9].

Традиційні підходи до обліку та управління автопарком характеризуються рядом системних недоліків, які в сукупності суттєво знижують ефективність роботи транспортної компанії. Першою і найбільш фундаментальною проблемою є відсутність єдиного сховища даних. Одна й та ж інформація розпорошена по різних місцях: дані про водія присутні і в таблиці відділу кадрів, і в журналі диспетчера, і в записнику механіка. Оновлення в одному джерелі не відображається в інших, унаслідок чого управлінські рішення приймаються на основі застарілої або суперечливої інформації. Нерозривно пов'язаним із цим є вплив людського фактора: ручне введення даних є постійним джерелом помилок. Переплутані цифри пробігу, пропущений запис заправки, неправильно вписаний номерний знак – кожна подібна неточність може спричинити хибне управлінське рішення, при цьому автоматичної перевірки не існує, тому більшість помилок виявляється вже після того, як їхні наслідки дали про себе знати. Окремої уваги потребує контроль технічного обслуговування. Відстеження строків ТО вручну вимагає від механіка або диспетчера постійної перевірки пробігу кожного транспортного засобу та його порівняння з нормативними інтервалами. Пропущений плановий огляд закономірно призводить до серйозних поломок, вимушених простоїв і непередбачених витрат на ремонт [10]. Аналогічна ситуація складається з контролем витрат пального: без автоматичного порівняння планових і фактичних показників виявити надвитрату або розкрадання практично неможливо, оскільки ручний розрахунок по кожному рейсу є трудомістким, схильним до арифметичних помилок і на практиці просто не виконується регулярно. Не менш гострою є проблема обробки платежів. Ручне зіставлення банківських виписок із замовленнями при значному обсязі транзакцій легко виходить з-під контролю: пропущений платіж або сплутаний ідентифікатор призводять до того, що замовлення залишається зі статусом неоплаченого, а дебіторська заборгованість

накопичується. Зрештою, усі перелічені проблеми зумовлюють відсутність оперативного контролю: керівник компанії не може в будь-який момент швидко отримати зведену картину такого вигляду: скільки машин перебуває в рейсі, хто з водіїв є вільним, якому транспортному засобу необхідне технічне обслуговування. Отримання таких даних вимагає ручного збору інформації з кількох розрізнених джерел.

Усі зазначені проблеми в сукупності дають чітке обґрунтування необхідності розробки спеціалізованої платформи, яка автоматизує ключові процеси та централізує дані в єдиному місці.

Ринок програмного забезпечення для управління автопарком досить широкий: від нішевих телематичних платформ до великих корпоративних TMS-систем. Розглянемо найбільш відомі рішення, що реально використовуються транспортними компаніями:

1. Wialon є хмарною платформою для GPS-моніторингу та управління автопарком від компанії Gurtam. За понад 20 років на ринку система підключила понад 4 млн транспортних одиниць у більш ніж 150 країнах. Сильна сторона Wialon: широка апаратна сумісність та детальна аналітика: відстеження в реальному часі, контроль витрат пального, моніторинг поведінки водіїв, геозонування, оптимізація маршрутів [11]. Разом із тим Wialon , що є насамперед телематична платформа, а не система управління замовленнями чи клієнтами. Крім того, для кожного авто потрібне окреме апаратне забезпечення, що суттєво підвищує вартість впровадження для малого бізнесу.

2. Американська платформа Samsara поєднує GPS-трекінг, відеотелематику з AI-камерами та діагностику транспортних засобів. Система вміє автоматично виявляти небезпечну поведінку водія та формувати звіти для служби безпеки [12]. Для малого та середнього бізнесу Samsara є доволі дорогим варіантом: орієнтовно 27-33 долари за авто на місяць із обов'язковим трирічним контрактом.

3. Fleetio є хмарною системою з акцентом на управлінні технічним обслуговуванням. Централізує планування превентивного ТО, облік запчастин і витрат, має інтеграцію з паливними картками. Порівняно з конкурентами: доступна цінова модель від 4 до 10 доларів за авто на місяць [13]. Основний недолік: відсутній функціонал обробки платежів і управління клієнтами.

4. Trimble є корпоративною TMS-системою,, що охоплює весь ланцюжок поставок від прийому замовлень до аналітики ефективності флоту. Ціноутворення виключно за індивідуальним запитом, що одразу говорить про орієнтацію на великі корпоративні бюджети [14].

5. Verizon Connect є платформою від телекомунікаційного гіганта Verizon поєднує GPS-трекінг, диспетчеризацію та управління безпекою. Найкраще підходить для середніх і великих флотів; вимагає укладення трирічного контракту зі штрафами за дострокове розірвання [15].

1.3 Порівняльний аналіз функціональних можливостей існуючих рішень для управління автопарком транспортної компанії

На основі розглянутих систем проведемо порівняльний аналіз за критеріями, що є найбільш важливими для вантажної транспортної компанії малого та середнього масштабу. Результати зведено в таблицю 1.1. Критерії для порівняння визначено виходячи з ключових операційних потреб транспортної компанії: управління замовленнями, облік пального, контроль технічного стану парку та інтеграційні можливості. Відбір систем для аналізу здійснювався на основі їх поширеності на ринку, наявності публічної документації та відповідності задачам вантажної логістики.

Дані для таблиці отримано з офіційної документації та сайтів відповідних систем [11-15]. Варто зазначити, що більшість розглянутих платформ орієнтовані на великі підприємства з розвиненою IT-інфраструктурою, тоді як потреби малого та середнього бізнесу задовольняються лише частково. Крім того, значна частина функціоналу в

таких системах є платною надбудовою або вимагає окремої інтеграції, що суттєво підвищує загальну вартість впровадження.

Для коректного порівняння систем необхідно уточнити методологію оцінювання. Кожен із критеріїв відображає окремий операційний процес, що є типовим для вантажної транспортної компанії. Зокрема, під «управлінням замовленнями» розуміється здатність системи реєструвати заявки клієнтів, зберігати інформацію про вантаж, адресу доставки та вартість перевезення, а також відстежувати стан виконання. «Контроль витрат пального» передбачає не лише реєстрацію заправок, а й автоматичне зіставлення фактичних показників із нормативними. «Управління технічним обслуговуванням» оцінюється з точки зору можливості автоматичного сповіщення про наближення до порогового пробігу, а не лише ручного ведення журналу ремонтів. Такий підхід дозволяє об'єктивно виявити прогалини в кожному з рішень.

Слід також відзначити, що при формуванні переліку критеріїв враховувалась специфіка українського ринку транспортних послуг. Зокрема, вимога щодо підтримки різних типів контрагентів (фізичні особи, ФОП, юридичні особи з кодом ЄДРПОУ) є характерною саме для вітчизняного правового середовища і не реалізована в жодній із розглянутих іноземних платформ. Аналогічно, автоматична обробка банківських реєстрів платежів у форматі електронних таблиць є поширеною практикою серед українських транспортних компаній, однак не підтримується жодним із аналізованих рішень.

Оцінки у таблиці 1.1 базуються на офіційній документації та публічних тарифних планах відповідних платформ станом на 2024 рік. Знак «+» означає повну підтримку функціоналу, «-» показує його відсутність, «Частково» вказує на реалізацію лише базових можливостей без повноцінної підтримки.

Таблиця 1.1

Порівняльний аналіз програмних рішень для управління автопарком

Критерій	**Wialon**	**Samsara**	**Fleetio**	**Trimble**	**Розроблена платформа**
Облік транспортних засобів	+	+	+	+	+
Облік водіїв	+	+	+	+	+
Облік клієнтів	-	-	Частково	-	+
Управління замовленнями	-	Частково	-	+	+
Управління рейсами	+	+	Частково	+	+
Контроль витрат пального	+	+	+	+	+
Технічне обслуговування	+	+	+	+	+
GPS-трекінг у реальному часі	+	+	-	+	-
Обробка банківських платежів	-	-	-	Частково	+
REST API	+	+	+	+	+
Відкритий вихідний код	-	-	-	-	+
Доступність для малого бізнесу	Середня	Низька	Висока	Низька	Висока
Локалізація (укр. мова)	Частково	-	-	-	+

Також, управління замовленнями в Wialon («-») не передбачає функції прийому або відстеження замовлень на перевезення. Samsara («Частково») реалізує диспетчеризацію маршрутів, однак не підтримує повний lifecycle замовлення від оформлення до виставлення рахунку. Fleetio («-») орієнтовано виключно на технічне обслуговування без функцій управління перевезеннями. Trimble («+») забезпечує повноцінний цикл управління замовленнями в межах корпоративного модуля TMS.

Функції GPS-трекінг у реальному часі в Fleetio («-») не включає власного GPS-моніторингу та вимагає інтеграції зі стороннім провайдером. Розроблена платформа також отримала «-» за цим критерієм, оскільки GPS-трекінг не входив до пріоритетних вимог поточної версії й запланований до реалізації в наступних ітераціях.

Важливо, у обробці банківських платежів жодна із зарубіжних платформ не підтримує формат виписок українських банків. Trimble («Частково») має базову інтеграцію з платіжними системами, але без підтримки реєстрів платежів у форматі CSV/XLS вітчизняних банків. Розроблена платформа («+») реалізує автоматичний імпорт та розбір банківських реєстрів із прив'язкою платежів до замовлень.

Щодо відкритого вихідного коду, то усі розглянуті комерційні системи є пропрієтарними («-»), що унеможливлює адаптацію під специфіку конкретного підприємства без залучення вендора. Розроблена платформа («+») поширюється з відкритим вихідним кодом, що дозволяє самостійно вносити зміни та уникнути залежності від постачальника.

Доступність для малого бізнесу для Wialon («Середня»), бо абонентська плата від 5 до 10 USD на автомобіль на місяць, однак обов'язкове GPS-обладнання додає від 50 USD за одиницю. Samsara («Низька»), від 27 до 33 USD на автомобіль на місяць за умови укладення контракту мінімум на 3 роки. Fleetio («Висока»), від 5 USD на автомобіль на місяць без обов'язкового апаратного забезпечення. Trimble («Низька»): корпоративне ціноутворення без публічних тарифів, орієнтоване виключно на великі підприємства. Розроблена платформа («Висока»): без ліцензійних відрахувань, витрати обмежуються хостингом та підтримкою сервера.

Щодо локалізації, то Wialon («Частково») надає україномовний інтерфейс, однак без локалізованих форм документів та підтримки банківських реєстрів. Samsara, Fleetio та Trimble («-») доступні виключно англійською мовою без підтримки кирилиці й специфіки українського ринку. Розроблена платформа («+») повністю локалізована для українського ринку.

З таблиці 1.1 видно, що всі розглянуті комерційні рішення є потужними платформами з широким набором функцій. Однак жодне з них не задовольняє в повній мірі специфічних потреб малої або середньої вантажної транспортної компанії. Жодна з розглянутих систем не має повноцінної локалізації для українського ринку та не підтримує специфіку роботи з вітчизняними реєстрами банківських платежів.

Таким чином, розробка власної платформи є цілком обґрунтованим рішенням: вона дозволяє врахувати всі специфічні вимоги конкретної компанії та уникнути залежності від дорогих закордонних рішень.

Висновки до розділу 1

У першому розділі детально розглянуто предметну область управління автопарком транспортної компанії та проаналізовано існуючі програмні рішення. Визначено основні бізнес-процеси вантажної транспортної компанії: управління транспортними засобами та їх технічним обслуговуванням, облік водіїв і клієнтів, управління замовленнями та рейсами, контроль витрат пального й обробка платежів. Виявлено ключові проблеми ручного та частково автоматизованого обліку, зокрема відсутність єдиного сховища даних, вплив людського фактора, неможливість оперативного контролю поточного стану парку та труднощі із зіставленням банківських платежів і замовлень. Проведено огляд і порівняльний аналіз п'яти програмних рішень: Wialon, Samsara, Fleetio, Trimble Transportation та Verizon Connect. Встановлено, що жодне з них не покриває повною мірою специфічні потреби малої вантажної транспортної компанії: зокрема, відсутня підтримка типізації контрагентів відповідно до українського законодавства та автоматична обробка банківських реєстрів платежів. Отримані результати підтверджують доцільність розробки власної платформи.

РОЗДІЛ 2

ФОРМУВАННЯ ВИМОГ ТА ПРОЄКТУВАННЯ ПЛАТФОРМИ УПРАВЛІННЯ АВТОПАРКОМ ТРАНСПОРТНОЇ КОМПАНІЇ

2.1 Формування вимог до платформи управління автопарком транспортної компанії

Функціональні вимоги описують, що саме система повинна вміти робити, тобто конкретні функції та операції, доступні користувачам. При їх формуванні за основу бралися бізнес-процеси, описані у першому розділі, та специфіка роботи вантажної транспортної компанії. Паралельно враховувалися обмеження, характерні для підприємств малого та середнього масштабу: відсутність виділеного IT-персоналу, необхідність мінімального часу на впровадження та низький поріг входу для кінцевих користувачів.

Систему вимог структуровано за вісьмома функціональними модулями, що безпосередньо відповідають виявленим бізнес-процесам: транспортні засоби, водії, клієнти, замовлення, рейси, облік палива, обробка платежів та звітність. Таке розбиття забезпечує незалежність компонентів при розробці й тестуванні та спрощує подальше розширення системи без змін у вже реалізованих модулях. Крім того, модульна структура вимог дозволяє пріоритизувати розробку відповідно до критичності кожного процесу для операційної діяльності компанії.

Ряд вимог має наскрізний характер і одночасно зачіпає декілька модулів. Зокрема, вимога щодо автоматичної зміни статусів транспортного засобу та водія при створенні та завершенні рейсу охоплює модулі «Транспорт», «Водії» та «Рейси». Аналогічно, автоматичне зіставлення банківських платежів із замовленнями зачіпає модулі «Платежі» і «Замовлення». Ці міжмодульні залежності реалізовані через окремий сервісний шар застосунку, що забезпечує слабку зв'язність між компонентами системи та спрощує їх незалежне тестування.

Для верифікації повноти покриття кожну функціональну вимогу зіставлено з відповідним бізнес-процесом, описаним у розділі 1. Відповідність між вимогами та бізнес-процесами перевірялася методом трасування: кожна вимога повинна бути обґрунтована конкретною операційною потребою, виявленою в ході аналізу предметної області. Перелік усіх функціональних вимог із зазначенням модуля реалізації наведено у таблиці нижче.

Таблиця 2.1

Функціональні вимоги до платформи управління автопарком

№	Функціональна вимога	Модуль
1	Реєстрація транспортного засобу з фіксацією моделі, номерного знаку, VIN, пробігу та норми витрати пального	Транспорт
2	Відстеження поточного статусу транспортного засобу (вільний / у рейсі)	Транспорт
3	Автоматичне визначення необхідності ТО за пробігом	Транспорт / ТО
4	Ведення бази даних водіїв з обліком статусів та дат вихідних	Водії
5	Автоматичне скидання статусу водія після закінчення вихідного / відпустки	Водії
6	Реєстрація клієнтів трьох категорій: фізособа, ФОП, компанія	Клієнти
7	Валідація обов'язкових реквізитів залежно від типу клієнта (ІПН, ЄДРПОУ)	Клієнти
8	Створення та управління замовленнями з прив'язкою до клієнта	Замовлення
9	Відстеження статусу оплати замовлення (не оплачено / оплачено)	Замовлення
10	Створення рейсу з призначенням водія, транспорту та замовлення	Рейси
11	Автоматична зміна статусів водія, транспорту та замовлення при зміні статусу рейсу	Рейси
12	Облік витрат пального з порівнянням планових та фактичних показників	Паливо
13	Реєстрація записів технічного обслуговування та ремонту	ТО
14	Завантаження банківського реєстру платежів та зіставлення з замовленнями	Платежі
15	Збереження незнайдених платежів для ручного опрацювання	Платежі
16	Генерація звітних документів по рейсах у форматі Word	Звітність

Ключовою особливістю системи є взаємозв'язок між модулями: зміна статусу рейсу автоматично тягне за собою зміну статусів водія, транспортного засобу та замовлення. Це забезпечує цілісність даних без ручного оновлення кожного об'єкта окремо.

Нефункціональні вимоги визначають не що робить система, а яким чином вона це робить, тобто характеристики якості, продуктивності, безпеки та підтримуваності. Нефункціональні вимоги до розробленої системи наведено в таблиці нижче.

Таблиця 2.2

Нефункціональні вимоги до платформи

Категорія	**Вимога**
Продуктивність	Час відповіді API на стандартні запити не більше 300 мс при навантаженні до 100 одночасних запитів
Надійність	Коректна обробка усіх помилкових сценаріїв з поверненням структурованих повідомлень про помилку
Безпека	Захист від SQL-ін'єкцій та XSS-атак засобами Django ORM; валідація всіх вхідних даних
Масштабованість	Модульна архітектура з сервісним шаром, що дозволяє додавати нові модулі без зміни існуючих
Переносимість	Розгортання на будь-якому сервері з підтримкою Python 3.x та PostgreSQL
Документованість	Автоматична генерація OpenAPI-документації через drf-spectacular / Swagger UI
Підтримуваність	Чітке розмежування відповідальності між шарами: моделі, серіалізатори, сервіси, представлення
Локалізація	Повна підтримка кирилиці; повідомлення про помилки: українською мовою

Окремо варто виділити вимогу щодо структурованої обробки помилок. У системі реалізовано власний формат відповідей із унікальними кодами виду XX_YY (де XX – номер модуля, YY – порядковий номер помилки) та україномовними повідомленнями. Такий підхід забезпечує однозначну ідентифікацію будь-якої помилки в системі та суттєво спрощує її локалізацію як під час розробки, так і в процесі експлуатації платформи.

2.2 Проєктування платформи управління автопарком транспортної компанії

На поточному етапі розробки система орієнтована на менеджера або адміністратора транспортної компанії; він має повний доступ до всіх функцій платформи через API. Архітектура системи передбачає можливість подальшого розширення з додаванням автентифікації та розмежування прав доступу.

Основні сценарії взаємодії користувача з системою:

1. Управління автопарком: додавання транспортних засобів, оновлення пробігу, реєстрація ТО або ремонту.
2. Управління персоналом: реєстрація водіїв, встановлення вихідних або відпустки.
3. Робота з клієнтами та замовленнями: реєстрація клієнтів різних категорій, створення замовлень.
4. Планування та виконання рейсів: вибір вільного водія, транспорту та замовлення, створення рейсу.
5. Контроль палива: внесення записів про заправки, перевірка відповідності витрат плановим.
6. Обробка платежів: завантаження банківського реєстру, опрацювання незнайдених записів.
7. Формування звітності: генерація Word-документів по завершених рейсах.

Перейдемо до архітектури платформи, оскільки система побудована за багатошаровою архітектурою з чітким розмежуванням відповідальності між компонентами. Архітектура системи складається з таких шарів:

- шар моделей (Models) описує структуру бази даних та зв'язки між сутностями. Реалізований засобами Django ORM, що абстрагує роботу з PostgreSQL.

- шар серіалізаторів (Serializers) відповідає за перетворення об'єктів моделей у формат JSON та назад, а також за первинну валідацію вхідних даних.
- сервісний шар (Services) містить бізнес-логіку системи: автоматичне управління статусами, розрахунки пального, обробку платіжних реєстрів.
- шар представлень (Views) обробляє HTTP-запити, викликає відповідні сервіси та повертає відформатовані відповіді через функціональні представлення DRF.
- шар маршрутизації (URLs) визначає відповідність між URL-адресами та функціями-обробниками.

Загальна архітектура системи наведена на рисунку нижче.

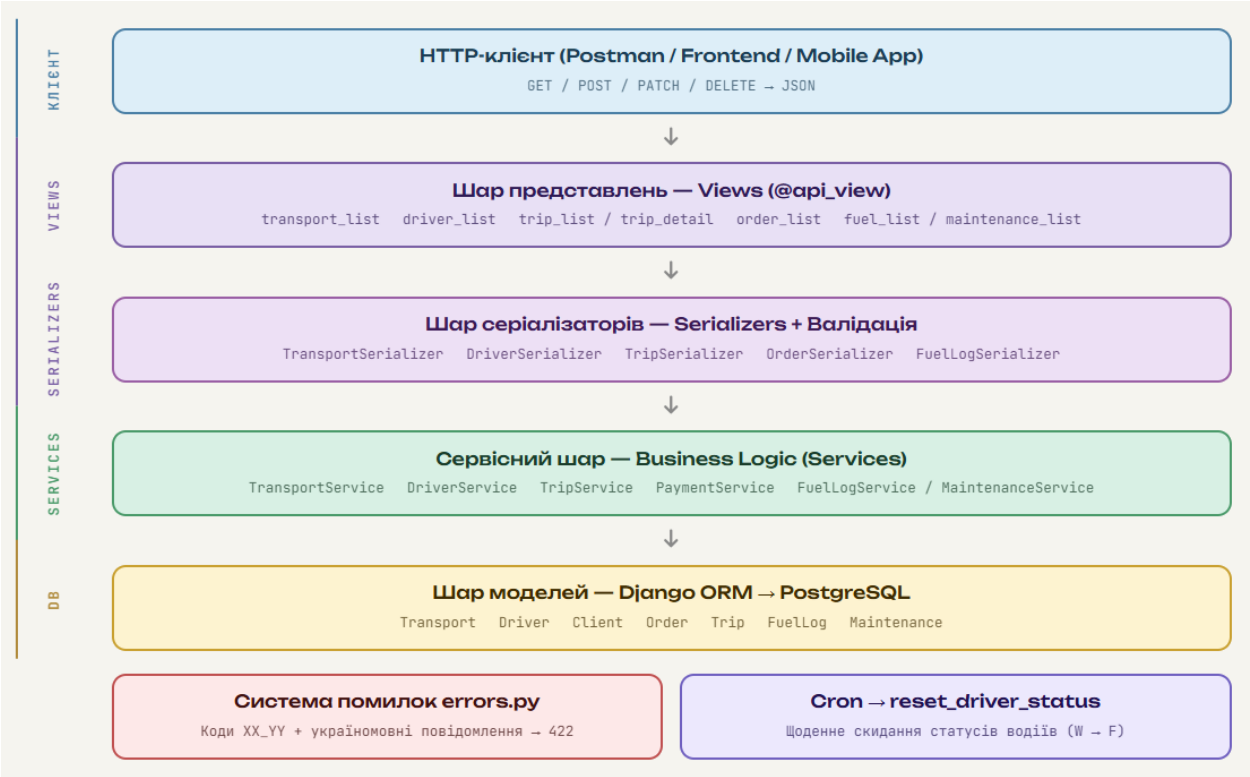

Рисунок 2.1. Архітектура платформи

Важливо відмітити структуру бази даних системи, яка містить вісім основних сутностей, між якими встановлено зв'язки типу «один-до-багатьох» (1:N). Структура спроєктована з урахуванням нормалізації до третьої нормальної форми (3НФ), що мінімізує надлишковість даних і забезпечує

цілісність інформації при будь-яких операціях. Кожна сутність відповідає окремому предметному об'єкту платформи, а використання зовнішніх ключів гарантує referential integrity на рівні бази даних. Центральною сутністю схеми є Trip, яка об'єднує чотири зовнішні ключі та виступає точкою інтеграції всіх операційних модулів системи. Структура складається з наступних елементів:

1. Transport (Транспортний засіб) містить такі поля: модель, номерний знак, VIN-код, пробіг, норму витрати пального, фото, статус (F/OW), статус ТО (S/NS/OS/R), інтервал між ТО та пробіг до наступного ТО.

2. Driver (Водій) містить: ПІБ, телефон, фото, посвідчення, ІПН, досвід, статус (F/OW/W), дату закінчення вихідного або відпустки.

3. Client (Клієнт) включає: тип (FIZ/FOP/COMP), ПІБ, телефон, email, ІПН, назву компанії, ЄДРПОУ, дату створення.

4. Order (Замовлення) містить посилання на Client (FK), ідентифікатор оплати, назву, ціну, статус доставки (W/D/E), адресу, статус оплати (NP/P).

5. Trip (Рейс) є центральною сутністю зі зв'язками до Driver, Transport, Client та Order (чотири FK). Містить точки відправлення та призначення, статус (AC/P/C), відстань, статус та об'єми витрат пального.

6. FuelLog (Журнал заправок) має зв'язки з Transport та Trip (два FK) та містить кількість літрів, вартість, часову мітку.

7. Maintenance (Технічне обслуговування) пов'язано з Transport (FK) та містить тип (S/R/E), вартість, дату.

8. NoPayedOrder (Незнайдений платіж) зберігає ідентифікатор платежу, дату, назву компанії, ПІБ представника, ІПН, ЄДРПОУ, суму.

ER-діаграма бази даних наведена на рисунку нижче.

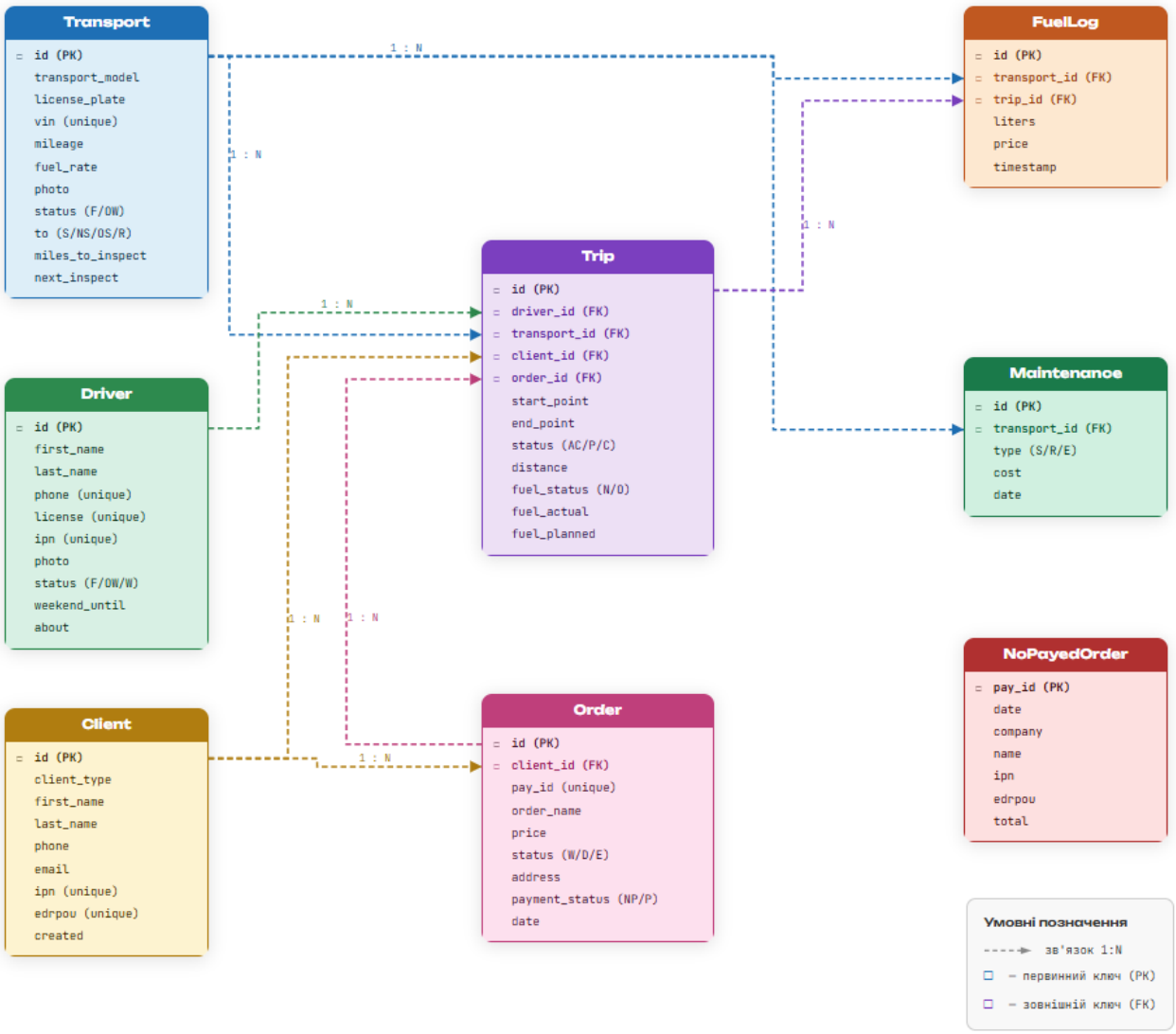

Рисунок 2.2. Схема бази даних

Перейдемо до опису проєктування API та взаємодії компонентів. Платформа реалізує REST API з підтримкою стандартних HTTP-методів, які включають: GET, POST, PATCH, DELETE. Усі відповіді повертаються у форматі JSON. Перелік реалізованих API-ендпоінтів наведено в таблиці, яка знаходиться в Додаток А.

Логіку зміни статусів рейсу та пов'язаних об'єктів наведено на рисунку. При переході рейсу в статус P система автоматично встановлює водію та транспортному засобу статус OW, а замовлення отримує статус D. При переході в статус C, усі статуси повертаються у вільний стан, система порівнює фактичні витрати пального з плановими.

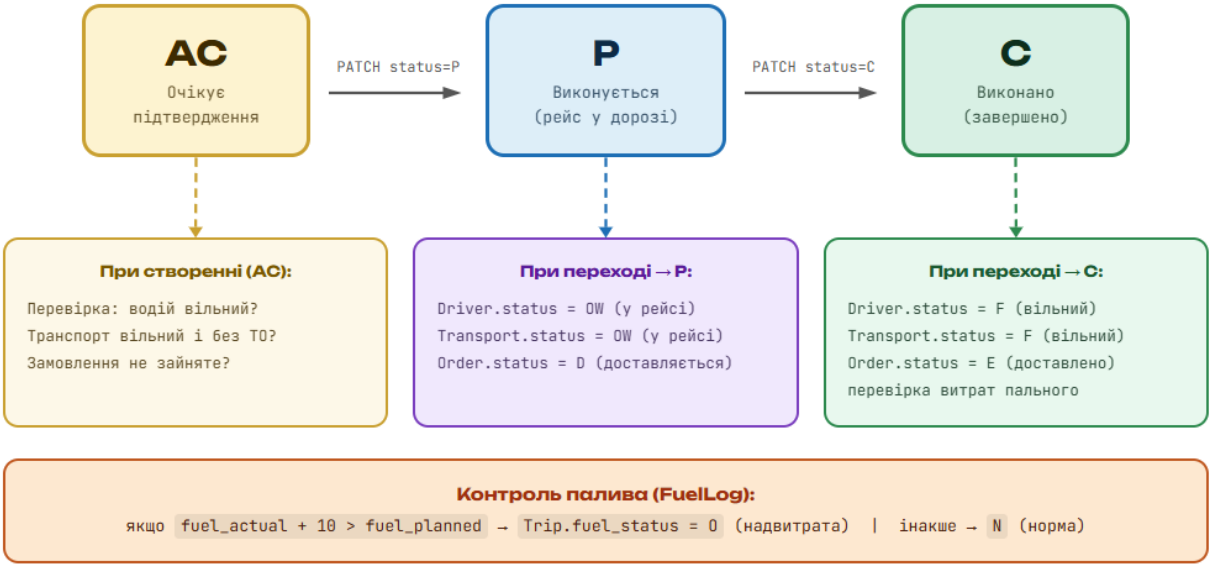

Рисунок 2.3. Логіка зміни статусів рейсу та пов'язаних об'єктів

Діаграму взаємодії клієнтської та серверної частин платформи наведено на рисунку нижче. На діаграмі показано повний шлях HTTP-запиту від браузера клієнта через рівень REST API та аутентифікації до сервісного шару, моделей Django ORM та бази даних PostgreSQL, а також повернення відповіді у форматі JSON.

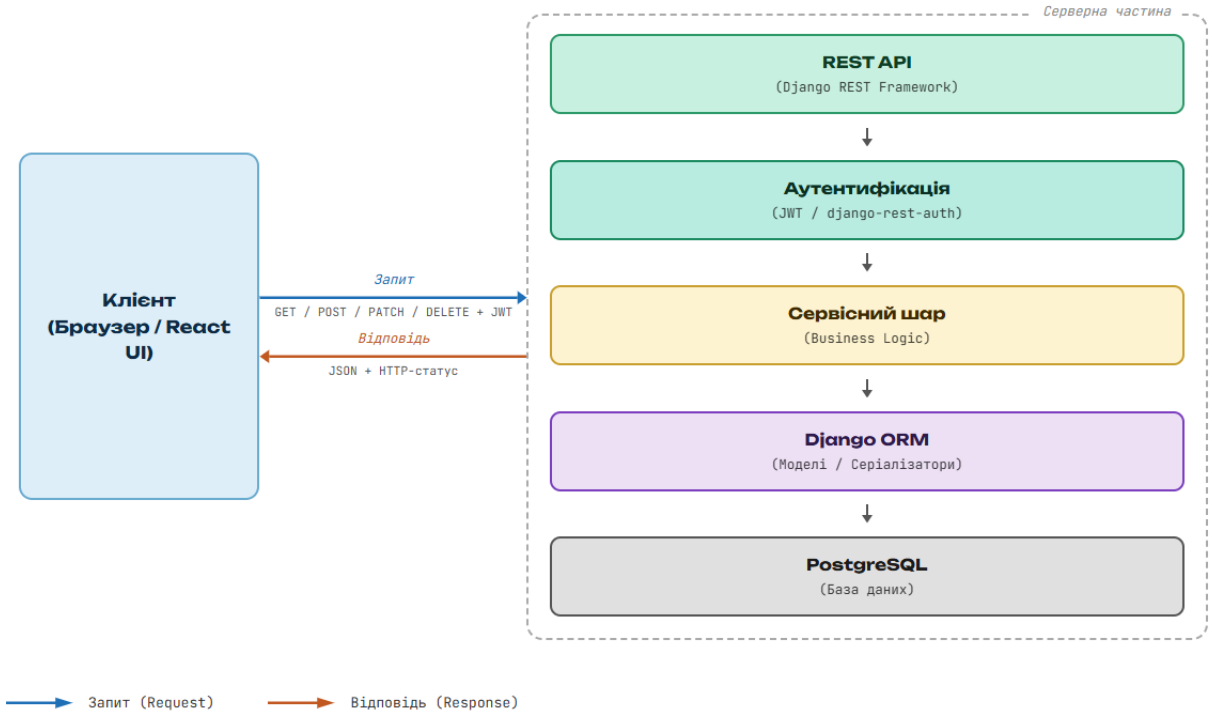

Рисунок 2.4. Структура взаємодії клієнта з платформою управління автопарком

Діаграма варіантів використання (Use Case Diagram), що наочно демонструє функціональні можливості платформи з точки зору її користувачів, описуючи доступні їм варіанти використання представлено на рисунку нижче.

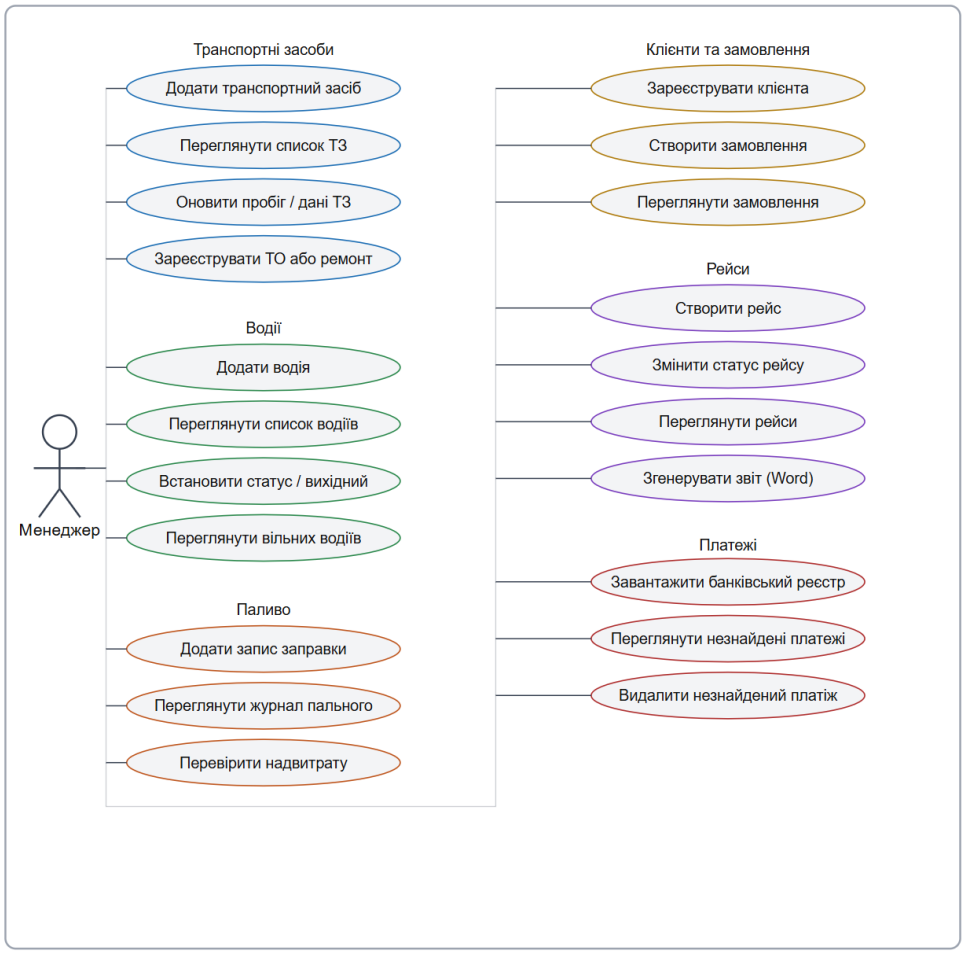

Рисунок 2.5. Архітектура платформи управління автопарком

При виборі технологій враховувалися такі ключові критерії: активна підтримка та регулярне оновлення з боку спільноти, наявність детальної офіційної документації, сумісність компонентів між собою, а також підтверджена практика використання стеку у production-середовищах галузевих систем. Перевага надавалася інструментам з відкритим вихідним кодом, що знижує залежність від конкретного постачальника та вартість впровадження для підприємств малого і середнього бізнесу.

Суттєвим чинником стала можливість побудови системи на єдиній мові програмування Python, яка охоплює і серверну частину, і обробку даних, і

генерацію документів. Такий підхід спрощує супроводження коду, скорочує час онбордингу нових розробників і дозволяє уникнути проблем гетерогенних стеків, де різні шари застосунку реалізовані різними технологіями зі власними залежностями та інструментами розгортання.

Окремо оцінювалася зрілість кожного компонента: вік проєкту, регулярність релізів, розмір спільноти та наявність довгострокової підтримки (LTS). Усі обрані технології відповідають цим вимогам і мають стабільний цикл розвитку. Перелік компонентів технологічного стеку із коротким обґрунтуванням вибору кожного наведено у таблиці нижче.

Таблиця 2.4

Технологічний стек розробленої системи

Компонент	**Технологія**	**Обґрунтування**
Мова програмування	Python 3.x	Широка екосистема, зручний синтаксис, велика активна спільнота
Веб-фреймворк	Django 5.2	Вбудований ORM, адмін-панель, надійна система міграцій
REST API	Django REST Framework	Зрілий інструмент для побудови API, серіалізатори, валідація
База даних	PostgreSQL	Надійна реляційна СУБД, підтримка складних запитів та транзакцій
API-документація	drf-spectacular	Автоматична генерація OpenAPI / Swagger документації
Обробка Excel	pandas	Зручне читання та обробка банківських реєстрів платежів
Генерація Word	python-docx	Програмна генерація .docx звітів по рейсах
Автоматизація	Management Commands + cron	Щоденне скидання статусів водіїв без зайвих залежностей
Зберігання медіа	Django Media Files	Зберігання фото транспортних засобів та водіїв

Python обрано як основну мову розробки завдяки широкій екосистемі бібліотек та зручному синтаксису. Django як веб-фреймворк надає ORM, вбудовану адміністративну панель та надійну систему міграцій схеми [16]. Django REST Framework є де-факто стандартом для побудови API на Django [17]. PostgreSQL обрано як СУБД завдяки надійності, підтримці складних реляційних запитів та транзакційній цілісності. Порівняно зі SQLite, що

використовувався на початковому етапі, PostgreSQL забезпечує кращу продуктивність при великих обсягах даних [18]. Бібліотека pandas використовується для обробки банківських реєстрів платежів у форматі Excel, python-docx для генерації звітних документів [19].

Висновки до розділу 2

У другому розділі сформовано вимоги до платформи управління автопарком та виконано її комплексне проєктування. Визначено функціональні вимоги, що охоплюють усі ключові операційні модулі системи: управління транспортними засобами з автоматичним відстеженням технічного обслуговування, облік водіїв із динамічною зміною статусів, ведення клієнтської бази з підтримкою трьох типів контрагентів, управління замовленнями та рейсами, контроль витрат пального і автоматизовану обробку банківських платежів. Поряд із функціональними визначено нефункціональні вимоги, що стосуються безпеки, продуктивності, масштабованості та зручності використання системи.

Спроєктовано багатошарову архітектуру з виділеним сервісним шаром, який забезпечує чітке розмежування між рівнем представлення даних та бізнес-логікою. Розроблено структуру реляційної бази даних із восьми взаємопов'язаних сутностей, що відображають усі предметні об'єкти платформи та зв'язки між ними. Сформовано повний перелік API-ендпоінтів із визначенням методів HTTP, структури запитів і відповідей, що забезпечує однозначну специфікацію інтерфейсу для подальшої реалізації.

Обґрунтовано вибір технологічного стеку: використання Python 3.x, Django 5.2, Django REST Framework та PostgreSQL зумовлено їх зрілістю, широкою екосистемою бібліотек, активною підтримкою спільноти та відповідністю вимогам розроблюваної системи. Застосування JWT-автентифікації забезпечує безпечний та stateless механізм контролю доступу до ресурсів API. Результати проєктування створили повноцінне та деталізоване підґрунтя для подальшої реалізації платформи.

РОЗДІЛ 3

РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПЛАТФОРМИ УПРАВЛІННЯ АВТОПАРКОМ ТРАНСПОРТНОЇ КОМПАНІЇ

3.1 Реалізація платформи управління автопарком транспортної компанії

Моделі бази даних реалізовано засобами Django ORM у файлі models.py. Кожна модель є Python-класом, що успадковує django.db.models.Model, і відповідає окремій таблиці в базі даних PostgreSQL.

Приклад визначення моделі Transport із відстеженням стану технічного обслуговування:

Лістинг 3.1. Модель транспортного засобу Transport

```
class Transport(models.Model):
transport_model = models.CharField(max_length=40, null=True, blank=True)
license_plate = models.CharField(max_length=10, null=True, blank=True)
    vin = models.CharField(max_length=17, unique=True)
    mileage = models.IntegerField(default=0)
    fuel_rate = models.IntegerField(default=0)
    status = models.CharField(choices=(
    ('F','Вільний'), ('OW','В дорозі')), default='F')
    to = models.CharField(choices=(
    ('S','Обслужений'), ('NS','Потрібне ТО'),
    ('OS','На обслуговуванні'), ('R','В ремонті')), default='S')
    miles_to_inspect = models.IntegerField(default=0)
    next_inspect     = models.IntegerField(default=0)
```

Модель Driver містить поле weekend_until типу DateField, що зберігає дату закінчення вихідного або відпустки. Після настання цієї дати management command reset_driver_status автоматично повертає водія у статус «вільний».

Модель Client підтримує три типи клієнтів із різними обов'язковими реквізитами. Поле client_type визначає правила валідації: для фізичних осіб та ФОПів обов'язковим є ІПН, для компаній обов'язковим є ЄДРПОУ та ім'я представника.

Уся бізнес-логіка системи винесена в окремий сервісний шар, який являє собою директорію services, де кожна модель має власний клас із статичними методами.

Так, для TransportService при кожному оновленні пробігу система перевіряє, чи досягнуто порогового значення, і якщо так, встановлює статус ТО у «NS»:

Лістинг 3.2. Методи оновлення пробігу та статусу ТО update_transport

```
@staticmethod
def update_transport(data: dict, transport: Transport):
    new_mileage = data['mileage']
    if new_mileage >= transport.next_inspect:
        transport.next_inspect += transport.miles_to_inspect
        transport.to = 'NS'
    transport.mileage = new_mileage
    ...
```

Для TripService при зміні статусу рейсу автоматично оновлюються статуси водія, транспортного засобу та замовлення:

Лістинг 3.3. Логіка ланцюгової зміни статусів рейсу TripService

```
if new_status == 'P':
    driver.status    = 'OW'
    transport.status = 'OW'
    order.status     = 'D'
if new_status == 'C':
    driver.status    = 'F'
    transport.status = 'F'
    order.status     = 'E'
```

Для FuelLogService при кожній заправці сервіс накопичує фактичний об'єм та автоматично визначає надвитрату:

Лістинг 3.4. Реалізація FuelLogSerivce: накопичення витрат пального та виявлення надвитрат

```
transport.fuel_rate += fuellog.liters
trip.fuel_actual = transport.fuel_rate
if (trip.fuel_actual + 10) > trip.fuel_planned:
    trip.fuel_status = 'O'
```

Для PaymentService відбувається обробка банківського реєстру через pandas:

Лістинг 3.5. Реалізація PaymentService: обробка банківського реєстру платежів

```
datas = pandas.read_excel(file, dtype=str)
for _, rows in datas.iterrows():
    payment_id = rows.get('payment_id')
    if Order.objects.filter(pay_id=payment_id).exists():

      Order.objects.filter(pay_id=payment_id).update(payment_status='P')
    else:
        NoPayedOrder.objects.create(**data)
```

Для Management command reset_driver_status реалізується щоденне автоматичне скидання статусів водіїв:

Лістинг 3.6. Реалізація management command reset_driver_status: автоматичне скидання статусів водіїв

```
def handle(self, *args, **kwargs):
    today   = timezone.localdate()
    drivers = Driver.objects.filter(
        status__in='W',
        weekend_until__lte=today
    )
    drivers.update(status='F', weekend_until=None)
```

REST API реалізовано через функціональні представлення Django REST Framework із декоратором @api_view. Приклад реалізації ендпоінтів для транспортних засобів:

Лістинг 3.7. Реалізація REST API ендпоінту для управління транспортними засобами

```
@api_view(['GET', 'POST'])
def transport_list(request):
    if request.method == 'GET':
        transports = Transport.objects.all()
        serializer = TransportSerializer(transports, many=True)
        return Response(serializer.data)
    if request.method == 'POST':
        serializer = TransportSerializer(data=request.data)
        serializer.is_valid(raise_exception=True)

TransportService.create_transport(data=serializer.validated_data)
        return Response(serializer.data, status=201)
```

API-документація генерується автоматично через бібліотеку drf-spectacular та доступна за адресою /swagger/, що дозволяє будь-якому розробнику, який інтегруватиметься з системою, ознайомитися з повним переліком ендпоінтів без додаткової документації [20].

Валідація даних та обробка помилок відбувається наступним чином: система валідації побудована на двох рівнях, де серіалізатори DRF обробляють помилки формату та унікальності, а сервісний шар парцює з порушенням бізнес-правил. Для кожного типу помилки визначено унікальний код у форматі XX_YY:

Лістинг 3.8. Реалізація системи структурованої обробки помилок із кодами виду XX_YY

```
ERRORS = {
    '00_01': 'Значення не може бути меншим за 0',
    '01_02': 'Новий пробіг не може бути меншим за попередній',
    '06_01': 'Водій в поїздці. Назначте іншого водія',
    '06_04': 'Транспорт потребує ТО/Ремонту',
    ...
}
def error_response(error_code: str):
    return {'error': {'code': error_code, 'message': ERRORS[error_code]}}
```

При виникненні помилки система генерує виключення APIException із HTTP-статусом 422. Схему валідації та обробки помилок наведено на рисунку нижче.

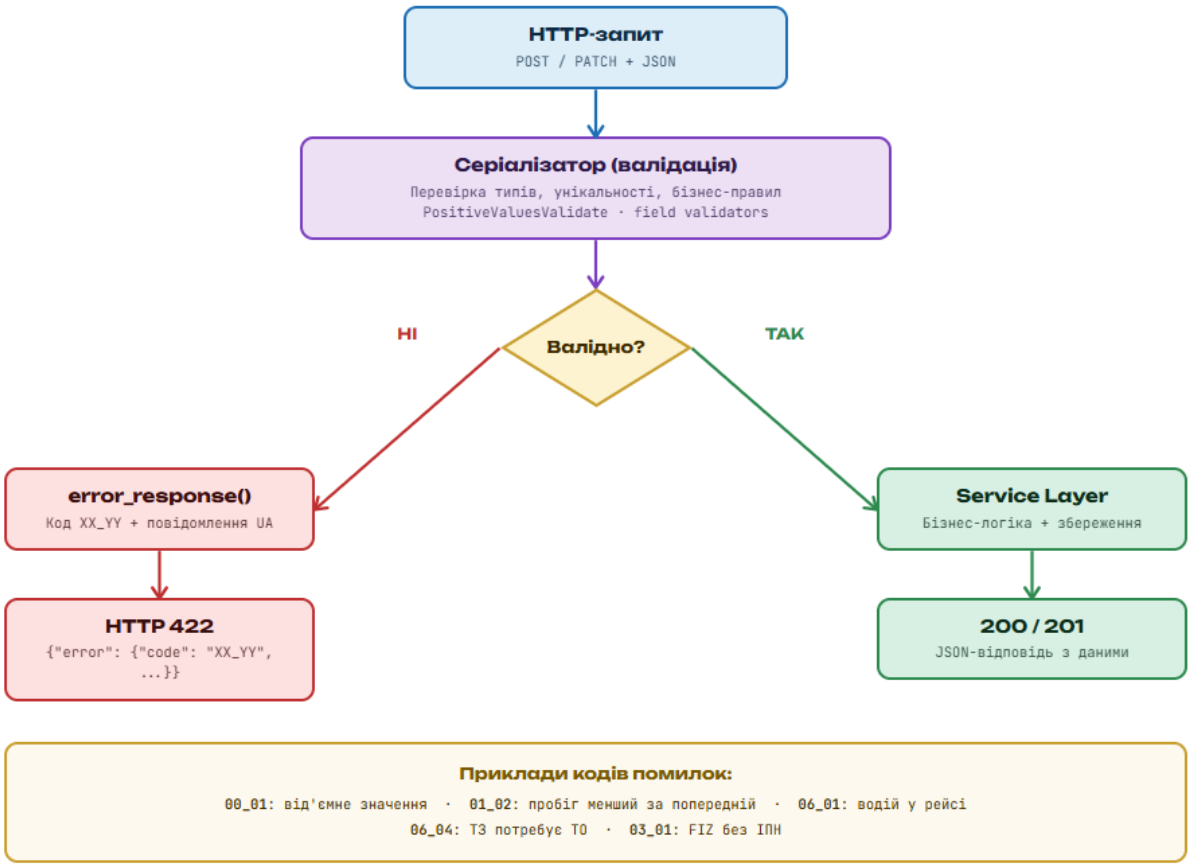

Рисунок 3.1. Схема валідації даних та обробки помилок

3.2 Тестування платформи управління автопарком транспортної компанії

Одним із ключових етапів розробки програмного забезпечення є тестування, яке дозволяє перевірити відповідність функціональним і нефункціональним вимогам, а також виявити наявні помилки в роботі системи. Для тестування платформи управління автопарком транспортної компанії проведено модульне тестування, яке реалізовувалось через Swagger UI та HTTP-клієнт Postman. Протестовано кожну функцію сервісного шару та кожен API-ендпоінт окремо, а саме:

- TransportService: розрахунок next_inspect, встановлення статусу «NS» при досягненні порогу.
- DriverService: метод «capitalize» для імені та прізвища, перевірка унікальності телефону, посвідчення та ІПН.

- TripService: кожен перехід статусів («AC»→«P», «P»→«C») з перевіркою оновлення пов’язаних об’єктів.
- FuelLogService: накопичення фактичного пального та поріг виявлення надвитрати.
- PaymentService: обробка реєстру з наявними та відсутніми замовленнями.
- reset_driver_status: фільтрація водіїв за датою weekend_until та масове оновлення статусів.
- Також проводилося тестування бізнес-логіки, яке зосереджувалось на перевірці складних взаємодій між модулями системи, а саме сценаріїв, де одна дія викликає ланцюгові зміни в кількох об'єктах.

Сценарій 1: Повний цикл рейсу. Тестувався повний маршрут від створення до завершення рейсу. Після переходу в статус «P» підтверджено зміну всіх трьох об'єктів на «OW/D». Після переходу в «C» відбувається повернення до «F/F/E» та перевірка fuel_status.

Сценарій 2: Технічне обслуговування. Перевірялась автоматична зміна статусу ТО при досягненні порогового пробігу. Після встановлення transport.to='NS' система блокує створення нового рейсу.

Сценарій 3: Обробка платежів із перехресною перевіркою. Логіка файлу OrderService перевіряє таблицю NoPayedOrder при кожному новому замовленні та одразу закриває знайдені борги.

Сценарій 4: Автоматичне скидання статусу водія. Перевірялась логіка management command при різних значеннях поля weekend_until.

Проводилося також тестування API через Swagger UI та Postman для кожного ендпоінту. Перевірялись коди відповідей, формат JSON, коректність серіалізації вкладених об'єктів.

Окремо тестувалась робота ендпоінту /payment/ з реальним Excel-файлом банківського реєстру. Перевірялось коректне зчитування кирилиці, обробка рядків із пропущеними полями та ситуація, коли файл містить лише незнайдені записи.

Для перевірки поведінки системи в нестандартних ситуаціях проводилося тестування крайових сценаріїв. Зведені результати тестування наведено в таблиці, яка знаходиться в Додаток Б.

Схему покриття тестами наведено на рисунку нижче.

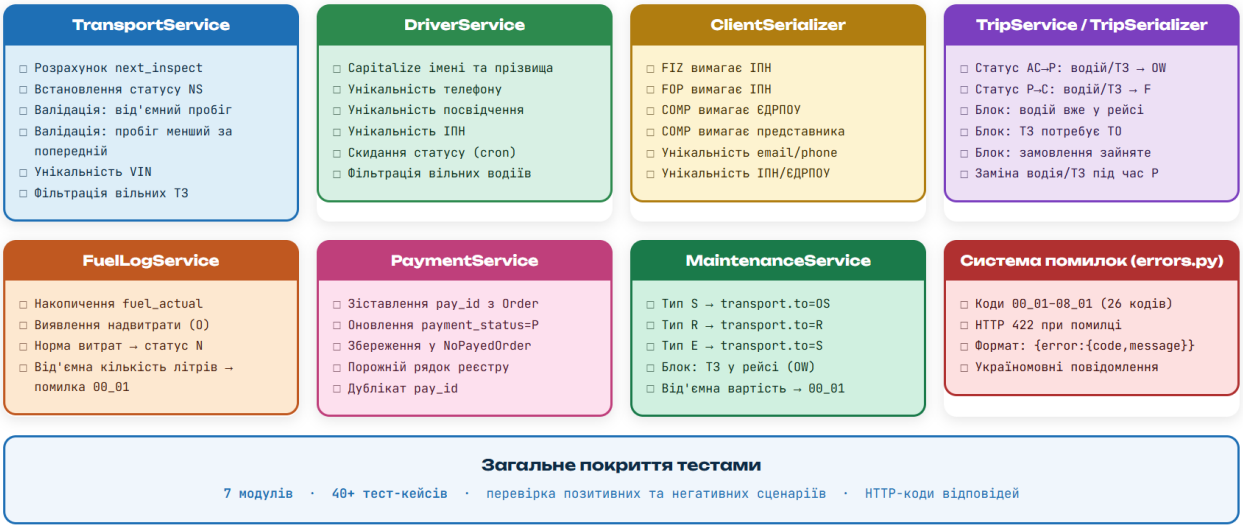

Рисунок 3.2. Схема покриття модульного тестування

Під час тестування крайових сценаріїв виявлено, що система коректно обробляє всі передбачені помилкові ситуації та повертає зрозумілі повідомлення українською мовою. Усі 26 кодів помилок перевірені та підтверджені. Структурований підхід до обробки помилок із уніфікованим форматом відповіді виду XX_YY суттєво спрощує діагностику проблем як на стороні клієнтського застосунку, так і під час підтримки системи. Жодного випадку некоректної або непередбаченої поведінки системи під час тестування виявлено не було.

3.3 Переваги платформи управління автопарком транспортної компанії

Розроблена платформа управління автопарком вирішує ключові проблеми, що були виявлені в першому розділі. Кожна реалізована функція напряму відповідає конкретній бізнес-потребі транспортної компанії та обґрунтована результатами порівняльного аналізу існуючих рішень. На відміну від розглянутих комерційних платформ, розроблена система охоплює

повний операційний цикл вантажної компанії від реєстрації клієнта та створення замовлення до завершення рейсу та звірки банківських платежів в рамках єдиного програмного середовища. Це усуває необхідність використання кількох несумісних інструментів одночасно та значно скорочує час на виконання рутинних операцій. Основні переваги наведено в таблиці нижче.

Таблиця 3.2

Переваги розробленої платформи

Перевага	**Опис**
Комплексність рішення	Єдина система охоплює весь операційний цикл без необхідності інтеграції кількох продуктів
Автоматизація статусів	Зміна статусу рейсу автоматично оновлює пов'язані об'єкти, знижуючи ризик помилок
Обробка банківських реєстрів	Автоматичне зіставлення платежів із замовленнями без аналогів у розглянутих рішеннях
Повна локалізація	Інтерфейс API та повідомлення про помилки: українською мовою
Відкритий вихідний код	Компанія може адаптувати систему без залежності від стороннього постачальника
Доступність для МСБ	Відсутність ліцензійних платежів і розгортання на власному сервері
Архітектурна гнучкість	REST API не залежить від клієнта, що забезпечує легку інтеграцію з веб, мобільним або ERP
Автоматизація кадрових процесів	Автоматичне скидання статусів водіїв після вихідних та відпусток

Одна з найважливіших особливостей системи є автоматична підтримка цілісності даних через ланцюгові зміни статусів. Коли диспетчер переводить рейс у статус «P», система одразу блокує водія та транспортний засіб від призначення на інші рейси. Це виключає ситуацію, коли той самий водій або автомобіль випадково потрапляє у два рейси одночасно. Аналогічний механізм спрацьовує у зворотному напрямку: після завершення рейсу (статус «C») система автоматично звільняє водія та транспортний засіб, одночасно оновлюючи статус замовлення та ініціюючи перевірку витрат пального. Таким чином, консистентність даних у всіх пов'язаних модулях підтримується без

будь-якого ручного втручання з боку користувача. Подібний підхід суттєво знижує операційні ризики та унеможливлює виникнення логічних суперечностей у базі даних навіть при одночасній роботі кількох диспетчерів у системі.

Система обробки банківських реєстрів є унікальною функціональністю, якої немає в жодному з розглянутих комерційних рішень у вигляді, що відповідає специфіці вітчизняного ринку. Вона дозволяє за лічені секунди зіставити сотні платежів із замовленнями. Реєстр завантажується у форматі, що відповідає стандартам українських банківських установ, після чого система автоматично знаходить відповідні замовлення за ідентифікатором платежу та оновлює їх статус. Платежі, для яких не вдалося знайти відповідне замовлення, автоматично зберігаються в окремій таблиці незнайдених платежів, що дозволяє диспетчеру опрацювати їх вручну без ризику втрати інформації. Завдяки цьому механізму повністю виключається ситуація накопичення неопрацьованої дебіторської заборгованості, а фінансовий стан усіх замовлень залишається актуальним в режимі реального часу. Прозорість розрахунків із клієнтами підвищується, а час, який раніше витрачався на ручне звірення виписок, скорочується до мінімуму.

Для об'єктивного порівняння проведено розширений аналіз за переліком критеріїв. До порівняння залучено ті самі п'ять комерційних платформ, що розглядалися в першому розділі, однак перелік критеріїв розширено з урахуванням технічних характеристик та можливостей інтеграції. Зведені результати наведено в таблиці нижче.

Таблиця 3.3

Розширений порівняльний аналіз систем управління автопарком

Критерій	**Wialon**	**Samsara**	**Fleetio**	**Trimble**	**Verizon**	**Розроблена платформа**
Управління замовленнями	-	Частково	-	+	Частково	+
Обробка банківських платежів	-	-	-	Частково	-	+
Локалізація (укр. мова)	Частково	-	-	-	-	+
Відкритий вихідний код	-	-	-	-	-	+
Доступність для МСБ	Середня	Низька	Висока	Низька	Низька	Висока
Облік клієнтів різних типів	-	-	Частково	-	-	+
Управління рейсами	+	+	Частково	+	+	+
Контроль ТО та пробігу	+	+	+	+	+	+
Контроль витрат пального	+	+	+	+	+	+
GPS-трекінг у реальному часі	+	+	-	+	+	-
REST API	+	+	+	+	+	+

Оцінки таблиці 3.3 відображають результати поглибленого порівняльного аналізу розробленої платформи з п'ятьма комерційними системами. До порівняння додатково включено Verizon Connect, яка є платформою управління автопарком, поширену в Північній Америці. Знак «+» означає повну підтримку, «-» вказує на відсутність, «Частково» показує часткову реалізацію.

Управління замовленнями: Wialon («-») і Fleetio («-») не мають модуля управління перевезеннями. Samsara та Verizon Connect («Частково»)

реалізують базову диспетчеризацію без повного lifecycle замовлення. Trimble («+») забезпечує повноцінний TMS-модуль. Розроблена платформа («+») реалізує весь цикл: від прийому замовлення та призначення водія до прив'язки платежу.

Обробка банківських платежів: усі зарубіжні платформи отримали «-», оскільки жодна не підтримує формати реєстрів українських банків (Приватбанк, Ощадбанк). Trimble («Частково») інтегрується з міжнародними платіжними шлюзами, але без підтримки вітчизняних форматів CSV/XLS. Розроблена платформа («+») реалізує автоматичний розбір банківських реєстрів та узгодження платежів із замовленнями.

Локалізація: Wialon («Частково») надає кирилічний інтерфейс, але без повної локалізації документів та банківських форматів. Samsara, Fleetio, Trimble і Verizon Connect («-») не підтримують українську мову ні в інтерфейсі, ні у звітних документах. Розроблена платформа («+») повністю локалізована.

Відкритий вихідний код: усі п'ять комерційних систем є пропрієтарними («-»), що обмежує можливості кастомізації. Розроблена платформа («+») розповсюджується з відкритим вихідним кодом, що забезпечує прозорість і незалежність від вендора.

Доступність для МСБ: Wialon («Середня») потребує апаратного GPS-обладнання додатково до абонентської плати. Samsara («Низька») та Trimble («Низька»): корпоративне ціноутворення з обов'язковими контрактами від 3 років. Verizon Connect («Низька»): мінімальний контракт на 36 місяців від 25 USD на автомобіль на місяць. Fleetio («Висока»): прозора тарифікація від 5 USD без обов'язкового обладнання. Розроблена платформа («Висока»): без ліцензійних платежів, лише хостинг.

Облік клієнтів різних типів: Wialon, Samsara, Trimble та Verizon Connect («-») не реалізують управління контрагентами. Fleetio («Частково») дозволяє прив'язати автомобіль до замовника без типізації. Розроблена платформа («+»)

підтримує ведення клієнтів різних категорій із прив'язкою типу вантажу та умов договору.

Управління рейсами: Fleetio («Частково») реалізує лише базовий трекінг переміщення без повноцінної логіки рейсу (призначення водія, маршрут, статуси, документи). Усі інші платформи, включаючи розроблену, забезпечують повний цикл управління рейсами.

GPS-трекінг у реальному часі: Fleetio та розроблена платформа отримали «-». Fleetio не включає власного рушія GPS-моніторингу. У розробленій платформі цей функціонал виділено в окремий майбутній модуль, що дозволить інтегруватися з будь-яким GPS-провайдером через стандартизований REST API.

Графічне порівняння систем за ключовими критеріями наведено на рисунку нижче.

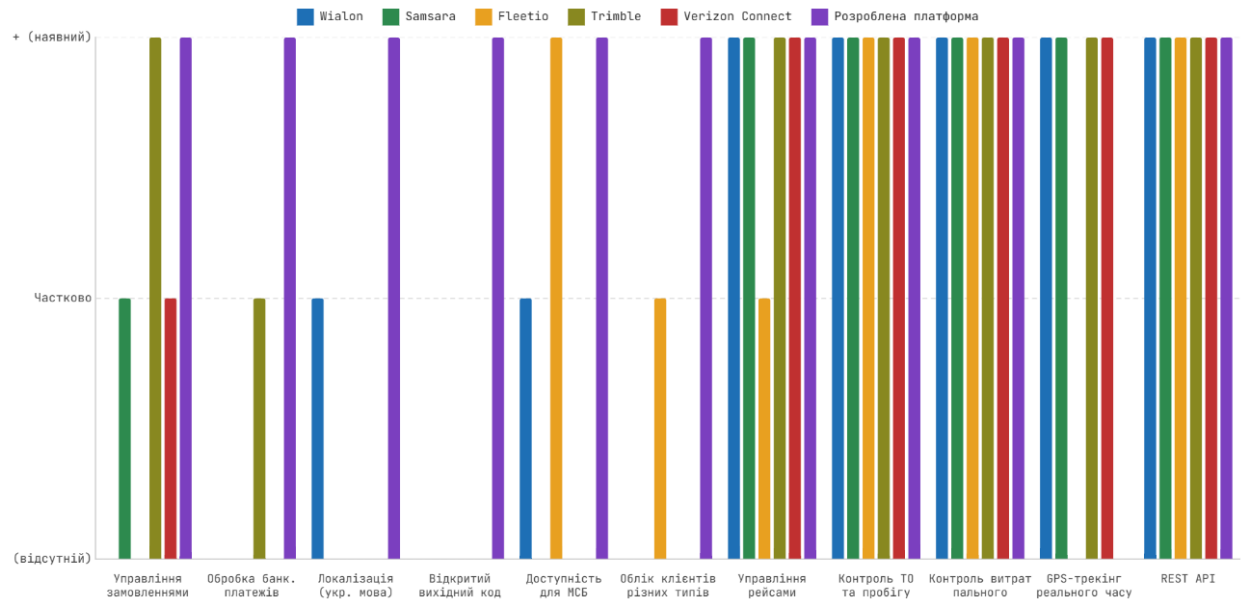

Рисунок 3.3. Порівняння систем управління автопарком за ключовими критеріями

Розроблена платформа поступається комерційним рішенням лише у функціональності GPS-трекінгу в реальному часі. За всіма іншими критеріями, що стосуються специфіки вітчизняних транспортних компаній, платформа або відповідає рівню комерційних аналогів, або перевершує їх.

3.4 Оцінка результатів впровадження платформи управління автопарком транспортної компанії

Оцінка результатів впровадження платформи управління автопарком транспортної компанії проводилася шляхом порівняння часових витрат на типові операції до та після автоматизації. Для отримання базових показників використовувалися оцінки, надані менеджерами транспортних компаній, що працюють із ручним або частково автоматизованим обліком, а також результати хронометражу аналогічних операцій у тестовому середовищі розробленої платформи. Порівняння охоплює найбільш трудомісткі щоденні операції: створення рейсу з призначенням водія та транспортного засобу, обробку банківського реєстру, формування звітів та контроль технічного стану парку. Результати наведено в таблиці нижче.

Таблиця 3.4

Порівняння часових витрат до та після впровадження системи

Показник	**До впровадження**	**Після впровадження**
Час на оформлення одного рейсу	15-25 хвилин (ручне заповнення)	3-5 хвилин (через API)
Оновлення статусів після рейсу	Ручне оновлення 3-4 записів	Автоматично при зміні статусу рейсу
Обробка банківського реєстру	1-2 години (ручне зіставлення)	До 1 хвилини (автоматично)
Виявлення надвитрати пального	Ручний розрахунок, нерегулярно	Автоматично після кожної заправки
Контроль строків ТО	Ручна перевірка пробігу	Автоматичний сигнал при досягненні порогу
Отримання списку вільних ресурсів	Перевірка кількох джерел (10-15 хв)	Один GET-запит (< 1 сек)
Скидання статусів після відпустки	Ручне оновлення диспетчером	Автоматично через cron щодня

Аналіз отриманих результатів дозволяє виділити кілька ключових напрямів, у яких автоматизація дала найбільш відчутний ефект.

Найбільший ефект досягається в трьох напрямах: обробка банківських реєстрів зменшилася з 1-2 годин до менше хвилини, контроль витрат пального відбувається автоматично після кожної заправки, а також управління статусами, де один API-запит замість оновлення трьох-чотирьох записів вручну.

При зміні статусу рейсу всі пов'язані оновлення виконуються в одній транзакції PostgreSQL, при цьому якщо будь-яке з них завершиться помилкою, вся операція відкочується, що унеможливлює часткове оновлення даних.

Окремої уваги заслуговує скорочення часу на оформлення рейсу. До впровадження платформи диспетчер був змушений вручну перевіряти доступність водія та транспортного засобу, звертаючись до кількох розрізнених джерел: журналу водіїв, таблиці техніки та паперового розкладу. Лише після цього він міг приступати до безпосереднього заповнення документів. У розробленій платформі система в реальному часі відображає лише вільних водіїв та справні транспортні засоби, що виключає можливість помилкового призначення та скорочує час оформлення до 3-5 хвилин.

Значний економічний ефект забезпечує автоматичний контроль витрат пального. За умов ручного обліку перевищення норми витрат, як правило, виявлялося із суттєвим запізненням, у кращому випадку наприкінці місяця під час зведення звітності. На практиці це означало, що аномальні показники витрат могли залишатися непоміченими впродовж тижнів. Розроблена платформа розраховує відхилення фактичних витрат від нормативних одразу після реєстрації кожної заправки, що дозволяє диспетчеру оперативно реагувати на будь-які відхилення ще до завершення рейсу. За оцінками фахівців галузі, своєчасне виявлення надвитрати пального здатне скоротити паливні витрати транспортної компанії на 8-15% на рік.

Автоматичний контроль строків технічного обслуговування також є важливим чинником зниження операційних витрат. Плановий огляд, пропущений через відсутність своєчасного сповіщення, нерідко призводить до серйозних поломок та вимушених простоїв техніки. Вартість

відновлювального ремонту в таких випадках у кілька разів перевищує вартість планового ТО. Платформа автоматично відстежує поточний пробіг кожного транспортного засобу та формує попередження при наближенні до встановленого порогу, що дозволяє завчасно планувати обслуговування без зупинки виробничого процесу.

Не менш важливим є ефект від автоматизації скидання статусів водіїв після завершення відпустки або вихідного. До впровадження платформи ця операція повністю залежала від людського фактора: якщо диспетчер забував вручну оновити статус водія, той залишався недоступним для призначення, що призводило до неефективного використання кадрового ресурсу. Реалізований механізм автоматичного скидання статусів через cron-завдання повністю усуває цю проблему, забезпечуючи актуальність кадрового обліку без жодного ручного втручання.

Сукупний ефект від впровадження платформи виражається не лише у скороченні часових витрат на окремі операції, а й у якісній зміні підходу до управління автопарком загалом. Керівник компанії отримує можливість у будь-який момент сформувати актуальну картину стану парку: кількість транспортних засобів у рейсі та на обслуговуванні, список вільних водіїв, обсяг дебіторської заборгованості та наближення планових ТО. Така оперативність прийняття рішень є принципово недосяжною в умовах ручного обліку і становить одну з ключових конкурентних переваг автоматизованої платформи.

3.5 Перспективи розвитку платформи управління автопарком транспортної компанії

Розроблена платформа є готовою серверною основою, яку можна розвивати у кількох напрямах. Поточна архітектура REST API спеціально спроєктована так, щоб нові модулі можна було підключати без зміни існуючого коду. Завдяки чіткому розмежуванню між сервісним шаром та рівнем представлення даних, додавання нової функціональності не вимагає

рефакторингу наявних компонентів, що суттєво знижує ризики та вартість подальшого розвитку системи. Перспективи розвитку наведено на рисунку нижче.

Рисунок 3.4. Перспективи подальшого розвитку платформи

Впровадження основних модулів:

1. Веб-інтерфейс для диспетчерів. Розробка клієнтського застосунку може бути здійснена на базі React або Vue.js, оскільки серверна частина повністю реалізована і задокументована через Swagger. Наявна документація API дозволяє фронтенд-розробникам розпочати роботу над інтерфейсом без додаткового занурення у серверний код. Інтерфейс диспетчера міг би включати зведену панель із поточним станом парку, інтерактивні форми для створення рейсів та замовлень, а також інструменти фільтрації та пошуку по всіх сутностях системи.

2. Мобільний додаток для водіїв. Даний модуль дозволить водіям самостійно оновлювати статуси рейсів та вносити записи про заправки з телефону [24]. Це усуне необхідність дублювання інформації через диспетчера та забезпечить актуальність даних у реальному часі. Водій зможе підтверджувати початок і завершення рейсу, фіксувати заправки з фотографуванням чека, а також отримувати сповіщення про нові призначення безпосередньо на мобільний пристрій.

3. Автентифікація та розмежування прав. Впровадження повноцінної рольової моделі на базі JWT-токенів передбачає три рівні доступу: адміністратор, менеджер і водій. Адміністратор матиме повний доступ до всіх модулів системи та налаштувань, менеджер до оперативних функцій управління рейсами, замовленнями та клієнтами, водій виключно до власних рейсів та можливості внесення записів про заправки. Такий підхід забезпечить інформаційну безпеку та унеможливить несанкціонований доступ до чутливих даних компанії.

4. GPS-моніторинг у реальному часі. Інтеграція з телематичними пристроями дозволить відстежувати місцезнаходження транспортних засобів безпосередньо в системі. Поточна архітектура платформи не перешкоджає такій інтеграції: достатньо реалізувати окремий сервісний клас, який отримуватиме координати від зовнішнього провайдера та зберігатиме їх у базі даних. Це відкриє можливості для побудови маршрутів, контролю відхилень від запланованого шляху та автоматичного розрахунку фактичної відстані рейсу без ручного введення.

5. Модуль аналітики та звітності. Дашборди для керівництва з графіками витрат, ефективності водіїв та динаміки замовлень дозволять перейти від оперативного до стратегічного рівня управління. Накопичена в базі даних інформація про рейси, витрати пального та платежі є достатньою основою для побудови аналітичних звітів: рейтинг водіїв за економністю витрат пального, динаміка завантаженості парку по місяцях, структура доходів у розрізі клієнтів. Для реалізації цього модуля може бути задіяна вже наявна у стеку бібліотека pandas, яка використовується для обробки банківських реєстрів.

6. Хмарне розгортання та масштабування. Контейнеризація через Docker та налаштування CI/CD-пайплайну забезпечать спрощення розгортання нових версій та підвищать надійність системи [25]. Використання Docker Compose дозволить стандартизувати середовище розгортання та уникнути залежності від конфігурації конкретного сервера. Впровадження автоматизованого пайплайну на базі GitHub Actions або GitLab CI забезпечить автоматичне

тестування та розгортання кожної нової версії платформи, що суттєво прискорить цикл розробки та знизить ризик появи регресійних помилок у продуктивному середовищі.

Реалізація перерахованих напрямів дозволить трансформувати розроблену серверну платформу на повноцінний комплексний продукт, готовий до комерційного впровадження в транспортних компаніях малого та середнього масштабу. При цьому модульна архітектура системи забезпечує можливість поетапного розвитку: кожен із зазначених напрямів може бути реалізований незалежно, без необхідності зупиняти або переробляти вже функціонуючі компоненти.

Висновки до розділу 3

У третьому розділі реалізовано та протестовано платформу управління автопарком, а також проведено комплексний аналіз результатів розробки. Розроблено структуру моделей бази даних, реалізовано сервісний шар із шістьма класами бізнес-логіки, приблизно 27 REST API ендпоінтів з повним набором CRUD-операцій та систему структурованої обробки помилок із 26 кодами. Проведено тестування 19 ключових тест-кейсів, з яких усі успішно пройдено, що підтверджує коректність реалізованої бізнес-логіки та надійність системи в цілому. Розширений порівняльний аналіз функціональних можливостей платформи підтвердив переваги над комерційними аналогами за критеріями управління замовленнями, обробки банківських платежів і локалізації. Аналіз часових витрат показав скорочення часу обробки банківського реєстру з 1-2 годин до менше хвилини, а отримання стану автопарку з 10-15 хвилин до секунди, що свідчить про реальний практичний ефект від впровадження розробленого рішення. Визначено шість перспективних напрямів подальшого розвитку платформи, реалізація яких дозволить трансформувати її на повноцінний комерційний продукт.

ВИСНОВКИ

Згідно результатів даної дипломної роботи розроблено платформу для автоматизації процесів обліку та управління автопарком транспортної компанії.

В ході виконання дипломної роботи було виконано наступні завдання:

1. Проаналізовано предметну область у сфері управління автопарком транспортної компанії. Виявлено основні бізнес-процеси та встановлено, що традиційні методи обліку призводять до дублювання даних, помилок через людський фактор та відсутність оперативного контролю.

2. Проведено огляд і порівняльний аналіз існуючих рішень програмного забезпечення для управління автопарком на комерційній та безоплатній основі. Виявлено відсутність повноцінних підходів, що в повній мірі забезпечують специфічні потреби вітчизняних компаній малого та середнього бізнесу.

3. Встановлено функціональні та нефункціональні вимоги до платформи управління автопарком. Серед функціональних вимог ключову роль відіграють автоматична підтримка цілісності даних через ланцюгові зміни статусів, підтримка трьох типів контрагентів відповідно до українського законодавства (фізична особа, ФОП, юридична особа) та автоматизована обробка банківських реєстрів платежів. Останнє є унікальною «родзинкою» платформи: жодне з розглянутих комерційних рішень не надає інструменту для автоматичного зіставлення банківських реєстрів із замовленнями у форматі, адаптованому до стандартів українських банківських установ. Серед нефункціональних вимог визначено вимоги до продуктивності, безпеки на основі JWT-автентифікації, масштабованості архітектури та структурованої обробки помилок із україномовними повідомленнями.

4. Спроєктовано структуру бази даних із восьми сутностей та повний перелік API-ендпоінтів. Обґрунтовано вибір технологічного стеку: Python, Django 5.2, Django REST Framework, PostgreSQL, pandas та python-docx.

5. Розроблено платформу управління автопарком, яка включила шість сервісних класів із повною бізнес-логікою: TransportService з автоматичним відстеженням ТО, TripService з ланцюговими змінами статусів, FuelLogService з виявленням надвитрат, PaymentService з обробкою банківських реєстрів та management command для щоденного скидання статусів водіїв.

6. Проведено тестування платформи управління автопарком на основі системи обробки помилок із 26 кодами виду XX_YY та тестування 19 тест-кейсів, які усі пройдено успішно.

7. Робота пройшла апробацію на XIII Всеукраїнській науково-практичній конференції молодих учених, 14 травня 2026 року, м. Київ.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Державна служба статистики України. Транспорт і зв'язок України 2024 : статистичний збірник. Київ : Держстат України, 2024. 154 с.

2. Крикавський Є. В. Логістика. Основи теорії : підручник. 2-ге вид., доп. і переробл. Львів : Національний університет «Львівська політехніка», 2006. 456 с.

3. Пономаренко В. С., Малярець Л. М. Аналіз даних у дослідженні соціально-економічних систем : монографія. Харків : ІНЖЕК, 2009. 432 с.

4. Войтко С. В., Гавриш О. А. Цифровізація підприємств транспортної галузі: тенденції та перспективи. Ефективна економіка. 2022. № 4. URL: http://www.economy.nayka.com.ua/?op=1&z=10012.

5. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures : dissertation. University of California, Irvine, 2000. 162 p. URL: https://ics.uci.edu/~fielding/pubs/dissertation/top.htm.

6. Харченко В. П. Управління технічним станом транспортних засобів : навч. посіб. Харків : ХНАДУ, 2018. 280 с.

7. Дорофієнко В. В., Шубін О. О. Управління персоналом транспортного підприємства : навч. посіб. Донецьк : ДонНУЕТ, 2010. 312 с.

8. Наумов В. С. Організація вантажних перевезень : підручник. Харків : ХНАДУ, 2019. 340 с.

9. Матвієнко О. В., Цивін М. Н. Основи організації електронного документообігу : навч. посіб. Київ : Центр учбової літератури, 2008. 112 с.

10. Гончаренко С. М. Технічна експлуатація автомобілів : підручник. Київ : Вища школа, 2015. 398 с.

11. Gurtam. Wialon Fleet Management Platform : офіційна документація. 2024. URL: https://gurtam.com/en/wialon.

12. Samsara. Connected Operations Platform : офіційна документація. 2024. URL: https://www.samsara.com.

13. Fleetio. Fleet Management Software : офіційна документація. 2024. URL: https://www.fleetio.com.

14. Trimble Transportation. Transportation Management System : офіційна документація. 2024. URL: https://transportation.trimble.com.

15. Verizon Connect. Fleet Management Solution : офіційна документація. 2024. URL: https://www.verizonconnect.com.

16. Django Software Foundation. Django Documentation 5.2. 2025. URL: https://docs.djangoproject.com/en/5.2/.

17. Encode OSS. Django REST Framework Documentation. 2024. URL: https://www.django-rest-framework.org.

18. The PostgreSQL Global Development Group. PostgreSQL 16 Documentation. 2024. URL: https://www.postgresql.org/docs/16/.

19. pandas Development Team. pandas Documentation 2.2. 2024. URL: https://pandas.pydata.org/docs/.

20. drf-spectacular Contributors. drf-spectacular Documentation. 2024. URL: https://drf-spectacular.readthedocs.io.

21. Mobley R. K. An Introduction to Predictive Maintenance. 2nd ed. Oxford : Butterworth-Heinemann, 2002. 438 p.

22. Chopra S., Meindl P. Supply Chain Management: Strategy, Planning, and Operation. 7th ed. Hoboken : Pearson, 2021. 528 p.

23. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol : O'Reilly Media, 2021. 620 p.

24. Sommerville I. Software Engineering. 10th ed. Harlow : Pearson Education, 2016. 816 p.

25. Burns B., Beda J., Hightower K. Kubernetes: Up and Running. 3rd ed. Sebastopol : O'Reilly Media, 2022. 326 p.

26. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2002. 560 p.

27. Kleppmann M. Designing Data-Intensive Applications. Sebastopol : O'Reilly Media, 2017. 616 p.

28. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Upper Saddle River : Prentice Hall, 2017. 432 p.

29. Richardson C. Microservices Patterns: With Examples in Java. Shelter Island : Manning Publications, 2018. 520 p.

30. Masse M. REST API Design Rulebook. Sebastopol : O'Reilly Media, 2011. 116 p.

31. Percival H., Gregory B. Architecture Patterns with Python. Sebastopol : O'Reilly Media, 2020. 280 p.

32. Lutz M. Learning Python. 5th ed. Sebastopol : O'Reilly Media, 2013. 1594 p.

33. Mozilla Developer Network. HTTP Response Status Codes. 2024. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status.

34. python-docx Contributors. python-docx Documentation. 2024. URL: https://python-docx.readthedocs.io.

35. GitHub. truck_crm – репозиторій дипломного проєкту. 2025. URL: https://github.com.

36. Abramov, V., Astafieva, M., Boiko, M., Bodnenko, D., Bushma, A., Vember, V., ... & Yaskevych, V. (2021). Theoretical and practical aspects of the use of mathematical methods and information technology in education and science.

ДОДАТКИ

Додаток А

API-ендпоінти платформи управління автопарком

Метод	Ендпоінт	Код	Опис
GET	/transport/	200	Отримати список усіх транспортних засобів
POST	/transport/	201	Додати новий транспортний засіб
GET	/transport/{id}/	200	Отримати дані конкретного ТЗ
PATCH	/transport/{id}/	200	Оновити дані транспортного засобу
DELETE	/transport/{id}/	204	Видалити транспортний засіб
GET	/get_free_transports/	200	Отримати список вільних ТЗ
GET	/driver/	200	Отримати список водіїв
POST	/driver/	201	Додати нового водія
GET	/driver/{id}/	200	Отримати дані конкретного водія
PATCH	/driver/{id}/	200	Оновити дані водія
DELETE	/driver/{id}/	204	Видалити водія
GET	/get_free_drivers/	200	Отримати список вільних водіїв
GET	/client/	200	Отримати список клієнтів
POST	/client/	201	Додати нового клієнта
GET	/client/{id}/	200	Отримати дані конкретного клієнта
PATCH	/client/{id}/	200	Оновити дані клієнта
DELETE	/client/{id}/	204	Видалити клієнта
GET	/order/	200	Отримати список замовлень
POST	/order/	201	Створити нове замовлення
GET	/order/{id}/	200	Отримати дані замовлення
PATCH	/order/{id}/	201	Оновити замовлення
DELETE	/order/{id}/	204	Видалити замовлення
GET	/get_free_orders/	200	Отримати замовлення зі статусом «Очікує»
GET	/trip/	200	Отримати список рейсів
POST	/trip/	201	Створити новий рейс

GET	/trip/{id}/	200	Отримати дані рейсу
PATCH	/trip/{id}/	200	Оновити рейс / змінити статус
DELETE	/trip/{id}/	204	Видалити рейс
GET	/fuel/	200	Отримати журнал заправок
POST	/fuel/	201	Додати запис заправки
GET	/fuel/{id}/	200	Отримати запис заправки
PATCH	/fuel/{id}/	200	Оновити запис заправки
DELETE	/fuel/{id}/	204	Видалити запис заправки
GET	/maintenance/	200	Отримати список записів ТО
POST	/maintenance/	201	Додати запис ТО або ремонту
GET	/maintenance/{id}/	200	Отримати запис ТО
PATCH	/maintenance/{id}/	200	Оновити запис ТО
DELETE	/maintenance/{id}/	204	Видалити запис ТО
POST	/payment/	200	Завантажити реєстр платежів (Excel)
GET	/nopayments/	200	Отримати список незнайдених платежів
GET	/nopayment/{id}/	200	Отримати запис незнайденого платежу
DELETE	/nopayment/{id}/	204	Видалити запис незнайденого платежу
GET	/get_order_word_doc/	200	Згенерувати Word-документ по всіх рейсах
GET	/get_order_word_doc/{id}/	200	Згенерувати Word-документ по рейсу

Додаток Б

Результати тестування системи

Тест-кейс	**Вхідні дані**	**Очікуваний результат**	**Статус**
Розрахунок next_inspect при створенні ТЗ	mileage = 5000, miles_to_inspect = 10000	next_inspect = 15000	Пройдено
Встановлення статусу NS при оновленні пробігу	mileage >= next_inspect	transport.to = 'NS'	Пройдено
Блокування від'ємного пробігу	mileage = -1	HTTP 422, код 01_01	Пройдено
Блокування зменшення пробігу	new_mileage < old_mileage	HTTP 422, код 01_02	Пройдено
Capitalize імені водія при створенні	first_name = 'іван'	first_name = 'Іван'	Пройдено
Унікальність телефону водія	Дублікат phone	HTTP 422, код 02_01	Пройдено
Клієнт FIZ без ІПН	client_type = 'FIZ', ipn = null	HTTP 422, код 03_01	Пройдено
Клієнт COMP без ЄДРПОУ	client_type = 'COMP', edrpou = null	HTTP 422, код 03_03	Пройдено
Зміна статусу рейсу AC→P	status = 'P'	Driver.status = OW, Transport.status = OW	Пройдено
Зміна статусу рейсу P→C	status = 'C'	Driver.status = F, Transport.status = F	Пройдено
Блок: водій вже у рейсі	driver.status = 'OW'	HTTP 422, код 06_01	Пройдено
Блок: ТЗ потребує ТО	transport.to = 'NS'	HTTP 422, код 06_04	Пройдено
Блок: замовлення доставляється	order.status = 'D'	HTTP 422, код 06_07	Пройдено
Виявлення надвитрати пального	fuel_actual + 10 > fuel_planned	trip.fuel_status = 'O'	Пройдено

ТО тип S → статус ТЗ	type = 'S'	transport.to = 'OS'	Пройдено
ТО тип E → завершення	type = 'E'	transport.to = 'S'	Пройдено
Обробка реєстру: знайдено замовлення	pay_id збігається з Order	Order.payment_status = 'P'	Пройдено
Обробка реєстру: не знайдено	pay_id не в базі	Запис у NoPayedOrder	Пройдено
Скидання статусу водія (cron)	weekend_until <= today	Driver.status = 'F'	Пройдено