ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ТЕХНОЛОГІЙ ЦИФРОВОГО РОЗВИТКУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Система автоматизації процесів логістики та адміністрування в управлінні кур'єрською службою з використанням мікросервісної архітектури»

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

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

_____ Данило БЄЛЯЄВ
(підпис) *(ім'я та прізвище)*

Виконав: здобувач вищої освіти групи ТЦР-41
Данило БЄЛЯЄВ

Керівник: Ігор СІНІЦИН
д-р техн. наук, проф. *(ім'я та прізвище)*

Рецензент: _____
(ім'я та прізвище)

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Технологій цифрового розвитку

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Технології цифрового розвитку»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Технологій цифрового розвитку
_____ Вікторія ЖЕБКА
« ___ » _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Бєляєву Данилу Анатолійовичу

1. Тема кваліфікаційної роботи: «Система автоматизації процесів логістики та адміністрування в управлінні кур'єрською службою з використанням мікросервісної архітектури»
керівник кваліфікаційної роботи д-р техн. наук, професор Ігор СІНІЦИН , затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «18» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про специфіку бізнес-процесів кур'єрських служб та математичні моделі маршрутизації (VRPTW) , опис архітектурних патернів побудови розподілених мікросервісних систем , технічна документація обраного технологічного стека (NestJS, Bun, Next.js, RabbitMQ, PostgreSQL, OR-Tools).

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

1. Аналіз предметної області, існуючих систем управління кур'єрськими службами та методів маршрутизації транспорту.

2. Архітектурне проєктування розподіленої системи управління логістикою, структури баз даних та патернів міжсервісної взаємодії.
3. Програмна реалізація API-шлюзу, ядра системи, математичного модуля маршрутизації та клієнтського веб-додатка.
4. Контейнеризація інфраструктури, навантажувальне тестування та оцінка ефективності системи.

5. Перелік графічного матеріалу: *презентація*
1. Аналіз існуючих платформ управління логістикою.
2. Вимоги до програмного забезпечення та топологія мікросервісної архітектури.
3. Схема бази даних (ER-діаграма) та розподіл відповідальності.
4. Патерни асинхронної комунікації між сервісами (RabbitMQ).
5. Математична модель та алгоритм розрахунку маршрутів (VRPTW).
6. Інтерфейси клієнтського застосунку диспетчера.
7. Результати навантажувального тестування мікросервісів.
8. Апробація результатів дослідження.

6. Дата видачі завдання «20» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури та аналіз існуючих аналогів логістичних систем	21.02-04.03.2026	Виконано
2	Дослідження методів розв’язання задач VRPTW та формування вимог до системи	05.03-15.03.2026	Виконано
3	Проєктування мікросервісної архітектури, схем даних та шини повідомлень	16.03-26.03.2026	Виконано
4	Реалізація сервісів логістики та воркера	27.03-17.04.2026	Виконано
5	Навантажувальне тестування та контейнеризація мікросервісів	18.04-27.04.2026	Виконано
6	Визначення перспектив розвитку та оформлення тексту пояснювальної записки	28.04-06.05.2026	Виконано
7	Розробка графічних та демонстраційних матеріалів	07.05-09.05.2026	Виконано
8	Попередній захист роботи	11.05.2026	Виконано

Здобувач вищої освіти _____ Данило БЄЛЯЄВ
(підпис) *(ім’я та прізвище)*

Керівник кваліфікаційної роботи _____ Ігор СІНІЦИН
(підпис) *(ім’я та прізвище)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 67 стор., 18 табл., 23 рис., 16 джерел.

Мета роботи – підвищення ефективності управління кур'єрською службою шляхом розробки та впровадження автоматизованої системи логістики та адміністрування на базі мікросервісної архітектури.

Об'єкт дослідження – процес автоматизації логістичних та адміністративних операцій у системах управління кур'єрською службою.

Предмет дослідження – архітектурні патерни, методи та програмні засоби побудови розподілених мікросервісних систем для оптимізації логістики.

Короткий зміст роботи: Проаналізовано бізнес-процеси служб доставки та обґрунтовано вибір мікросервісної архітектури. Досліджено задачу маршрутизації транспорту з часовими вікнами (VRPTW). Спроєктовано структуру системи: API-шлюз та мікросервіси замовлень, користувачів і логістики. Програмно реалізовано бекенд на базі NestJS та Bun із використанням брокера повідомлень RabbitMQ, концепції RPC, баз даних PostgreSQL (DrizzleORM) та Redis. Розроблено клієнтський веб-додаток на NextJS для адміністрування в реальному часі. Виконано контейнеризацію компонентів (Docker), налаштовано розгортання в Kubernetes та проведено навантажувальне тестування.

Сферою використання застосунку є автоматизація операційної діяльності середніх та великих логістичних компаній, що спеціалізуються на кур'єрській доставці.

КЛЮЧОВІ СЛОВА: МІКРОСЕРВІСНА АРХІТЕКТУРА, ЛОГІСТИКА, КУР'ЄРСЬКА СЛУЖБА, NESTJS, NEXTJS, RABBITMQ, KUBERNETES, VRPTW, BUN, DRIZZLEORM

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ACID (Atomicity, Consistency, Isolation, Durability) – набір вимог до транзакцій реляційних баз даних (атомарність, узгодженість, ізольованість, довговічність).

AMQP (Advanced Message Queuing Protocol) – відкритий протокол для передачі повідомлень між компонентами системи через черги.

API (Application Programming Interface) – інтерфейс прикладного програмування.

BFF (Backend-For-Frontend) – архітектурний патерн проєктування окремих бекенд-сервісів під конкретні клієнтські додатки.

CPU (Central Processing Unit) – центральний процесор (використовується для позначення обчислювально важких навантажень).

CRM (Customer Relationship Management) – система управління взаємовідносинами з клієнтами.

CRUD (Create, Read, Update, Delete) – базові операції роботи з даними (створення, зчитування, оновлення, видалення).

CSR (Client-Side Rendering) – стратегія рендерингу веб-сторінок на стороні клієнта у браузері.

CSS (Cascading Style Sheets) – каскадні таблиці стилів для оформлення інтерфейсу користувача.

CVRP (Capacitated Vehicle Routing Problem) – задача маршрутизації транспортних засобів з обмеженням їхньої вантажопідйомності.

DDD (Domain-Driven Design) – методологія предметно-орієнтованого проєктування складних програмних систем.

DLQ (Dead Letter Queue) – ізольована черга нерозпізнаних або помилкових повідомлень у брокері черг.

DLX (Dead Letter Exchange) – обмінник неуспішно оброблених повідомлень у RabbitMQ.

DRY (Don't Repeat Yourself) – принцип розробки ПЗ, спрямований на уникнення дублювання бізнес-логіки.

DTO (Data Transfer Object) – шаблон проєктування, призначений для передачі строго типізованих даних між підсистемами.

E2E (End-to-End) – наскрізне тестування повного життєвого циклу процесів застосунку.

EDA (Event-Driven Architecture) – подіє-орієнтована архітектура розподілених систем.

ER-діаграма (Entity-Relationship) – графічна модель структури даних та зв’язків реляційної бази даних.

ERP (Enterprise Resource Planning) – комплексна система планування та управління ресурсами підприємства.

ETA (Estimated Time of Arrival) – розрахунковий (прогнозований) час прибуття кур’єра до точки доставки.

EWKT (Extended Well-Known Text) – розширений текстовий формат ГІС-даних для представлення просторових об’єктів у PostGIS.

FK (Foreign Key) – зовнішній ключ, що забезпечує зв’язок між таблицями реляційної БД.

GLS (Guided Local Search) – алгоритм керованого локального пошуку для розв’язання NP-складних логістичних задач.

GPS (Global Positioning System) – супутникова система глобального позиціонування об’єктів у реальному часі.

HPA (Horizontal Pod Autoscaler) – механізм автоматичного горизонтального масштабування обчислювальних реплік (подів) у Kubernetes.

HTTP (HyperText Transfer Protocol) – прикладний протокол передачі гіпертексту мережею.

IAM (Identity and Access Management) – підсистема управління автентифікацією, ідентифікацією та контролем доступу.

I/O (Input/Output) – операції введення-виведення даних (мережа, файлова система, СУБД).

JSON (JavaScript Object Notation) – текстовий формат обміну структурованими даними.

JSONB (Binary JSON) – розширений бінарний формат збереження оптимізованих JSON-документів у PostgreSQL.

JWT (JSON Web Token) – компактний стандарт безпечної передачі токенів доступу та ідентифікації сесій.

K8s (Kubernetes) – відкрита платформа автоматизації оркестрації та масштабування контейнеризованих додатків.

ML (Machine Learning) – методи машинного навчання та предиктивної аналітики.

NP (Non-deterministic Polynomial) – клас обчислювальної складності задач (застосовується для позначення класу NP-складних логістичних проблем комбінаторної оптимізації).

ORM (Object-Relational Mapping) – технологія об’єктно-реляційного відображення баз даних у коді застосунку.

OSRM (Open Source Routing Machine) – високоефективний відкритий клієнт-серверний рушій для розрахунку найкоротших шляхів на дорожній графі.

OTP (One-Time Password) – одноразовий пароль/код для безпечного підтвердження фіксації факту доставки.

PVC (PersistentVolumeClaim) – запит мікросервісу на динамічне виділення дискової пам’яті в Kubernetes.

QA (Quality Assurance) – комплекс процесів забезпечення та контролю якості програмного продукту.

RBAC (Role-Based Access Control) – модель розмежування доступу користувачів до інтерфейсів на основі їхніх ролей.

REST (Representational State Transfer) – архітектурний стиль взаємодії компонентів розподіленої системи за допомогою безстанів HTTP-запитів.

RPC (Remote Procedure Call) – концепція та набір протоколів віддаленого виклику процедур між сервісами.

RPS (Requests Per Second) – основна метрика пропускної здатності системи (кількість запитів на секунду).

SEO (Search Engine Optimization) – комплекс заходів щодо пошукової оптимізації та просування веб-ресурсів.

SLA (Service Level Agreement) – договір/угода між виконавцем і замовником про гарантований рівень якості сервісу. SOLID – набір з п’яти базових принципів об’єктно-орієнтованого проєктування чистих та гнучких програмних архітектур.

SSE (Server-Sent Events) – технологія асинхронної односторонньої передачі подій у реальному часі від сервера до клієнта.

SSL (Secure Sockets Layer) – криптографічний протокол захисту з’єднань мережі.

TLS (Transport Layer Security) – сучасний криптографічний протокол захисту транспортного рівня мережі (наступник SSL).

TTL (Time to Live) – параметр обмеження часу життя пакета, повідомлення у черзі або сесії кешу.

UI (User Interface) – інтерфейс користувача.

URL (Uniform Resource Locator) – уніфікована адреса розташування цифрового ресурсу в інтернет-мережі.

UUID (Universally Unique Identifier) – універсальний унікальний 128-бітний ідентифікатор об’єктів у розподілених системах.

VRP (Vehicle Routing Problem) – класична комбінаторна задача маршрутизації транспортних засобів.

VRPTW (Vehicle Routing Problem with Time Windows) – розширена математична задача маршрутизації транспорту з урахуванням жорстких часових вікон клієнтів.

WMS (Warehouse Management System) – автоматизована система управління складськими процесами та залишками.

XML (eXtensible Markup Language) – розширювана мова розмітки для збереження та передачі структурованих даних.

ВСТУП

Актуальність теми. Сфера електронної комерції та кур’єрської доставки є однією з найбільш динамічних галузей сучасної економіки. Швидке зростання обсягів відправлень вимагає від логістичних компаній впровадження надійних, масштабованих та швидкодіючих систем управління (CRM/ERP). Традиційні монолітні архітектури, які часто використовуються для адміністрування таких процесів, досягли своєї межі: вони важко піддаються масштабуванню при пікових навантаженнях, ускладнюють впровадження нового функціоналу та мають низьку відмовостійкість.

Використання мікросервісної архітектури дозволяє ізолювати окремі бізнес-домени (управління користувачами, трекінг відправлень, побудова маршрутів), забезпечуючи їх незалежне розгортання та масштабування. Застосування подіє-орієнтованої архітектури (Event-Driven Architecture) дозволяє реалізувати асинхронну взаємодію компонентів системи для ефективної обробки великих масивів даних у реальному часі. Таким чином, розробка автоматизованої системи управління логістикою на базі мікросервісів є критично важливою для забезпечення конкурентоспроможності кур’єрських служб, що робить дану тему актуальною.

Мета роботи – підвищення ефективності управління кур’єрською службою шляхом розробки та впровадження автоматизованої системи логістики та адміністрування на базі мікросервісної архітектури.

Для досягнення мети у кваліфікаційній роботі успішно виконано такі завдання:

1. Проведено аналіз предметної області та специфіки бізнес-процесів сучасної кур’єрської служби.
2. Досліджено та обґрунтовано вибір мікросервісної архітектури та технологічного стека для реалізації високонавантаженої системи.

3. Спроєктовано архітектуру баз даних, API-шлюзу та патернів взаємодії між мікросервісами.
4. Реалізовано програмні модулі бекенду системи з інтеграцією алгоритмів оптимізації маршрутів (зокрема задачі маршрутизації транспорту з часовими вікнами – VRPTW).
5. Розроблено клієнтський вебдодаток для адміністрування логістичних процесів.
6. Проведено контейнеризацію, налаштовано оркестрацію інфраструктури та виконано тестування продуктивності розробленої системи.

Об'єкт дослідження – процес автоматизації логістичних та адміністративних операцій у системах управління кур'єрською службою.

Предмет дослідження – архітектурні патерни, методи та програмні засоби побудови розподілених мікросервісних систем для оптимізації логістики.

Методи дослідження. Під час виконання кваліфікаційної роботи були використані методи системного аналізу (для формування вимог до системи), методи архітектурного проєктування програмного забезпечення (для побудови мікросервісної інфраструктури), методи математичного моделювання (для алгоритмів оптимізації маршрутизації), а також методи навантажувального та модульного тестування (для перевірки надійності розробленого рішення).

Наукова новизна одержаних результатів. У ході дослідження набув подальшого розвитку підхід до побудови високонавантажених логістичних інформаційних систем шляхом застосування гібридних середовищ виконання у поєднанні з подіє-орієнтованою архітектурою. Це дозволило оптимізувати процес обробки обчислювально складних задач (зокрема, маршрутизації з часовими вікнами – VRPTW) в асинхронному режимі, мінімізувати системні затримки та забезпечити узгодженість даних у розподіленому середовищі.

Практична значущість одержаних результатів. Розроблена система забезпечує готове, масштабоване програмне рішення для адміністрування кур'єрської доставки. Використання сучасних технологій контейнеризації та оркестрації дозволяє швидко розгортати систему на будь-яких хмарних

платформах, а реалізовані алгоритми оптимізують час доставки та знижують операційні витрати компанії.

Апробація результатів дослідження. Ключові аспекти, концептуальні рішення та отримані результати кваліфікаційної роботи були представлені й обговорені в межах науково-практичних заходів (конференцій), організованих Державним університетом інформаційно-комунікаційних технологій:

1. Бєляєв Д. А., Ніщеменко Д. О. Безпечна автентифікація користувачів у WebSocket-з'єднаннях на основі одноразового токена // Всеукраїнська науково-технічна конференція «Виклики та рішення в програмній інженерії». Збірник тез. Київ: ДУІКТ, 2025 – ст. 14.
2. Бєляєв Д. А., Жебка В. В. Оптимізація часових вікон доставки в розподілених логістичних системах на основі методів комбінаторної оптимізації // VII Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. Київ: ДУІКТ, 2026 – ст. 215.
3. Бєляєв Д. А. Мікросервісна архітектура логістичної CRM-системи для асинхронної обробки високонавантажених задач маршрутизації // Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. Київ: ДУІКТ, 2026.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА МЕТОДІВ АВТОМАТИЗАЦІЇ ЛОГІСТИКИ

1.1. Специфіка бізнес-процесів у сучасних кур'єрських службах

Сучасна логістика останньої милі вимагає переходу від статичного планування до динамічного управління ресурсами в режимі реального часу. Основна складність полягає в необхідності обробляти великі масиви даних (геопозиціювання, статуси замовлень, телеметрія) за умов постійно мінливого зовнішнього середовища.

1.1.1. Життєвий цикл замовлення

Процес доставки в інформаційній системі являє собою складний скінченний автомат, де переходи між станами ініціюються транзакціями або зовнішніми подіями.

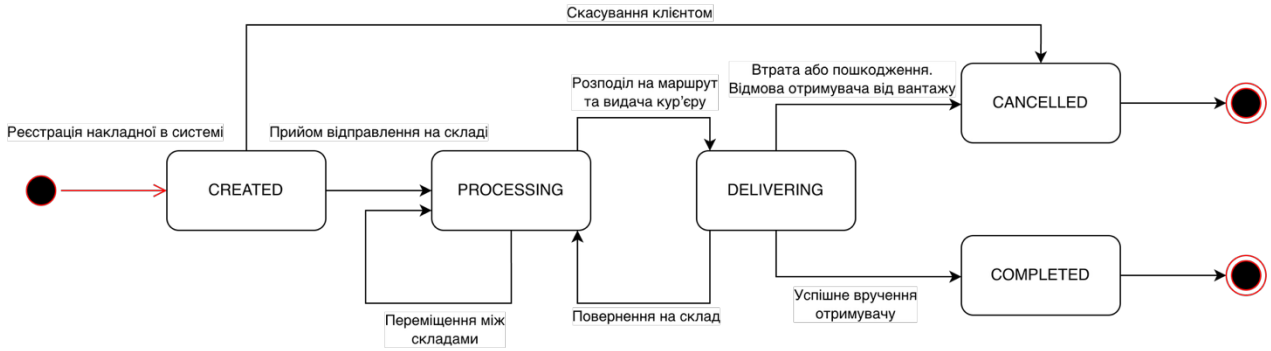


Рис. 1.1 Скінченний автомат станів замовлення

Життєвий цикл включає такі базові етапи:

1. *Реєстрація та перевірка.* Фіксація запиту, нормалізація та геокодування адреси, перевірка габаритів вантажу та доступності ресурсів;
2. *Розподіл та маршрутизація.* Призначення замовлення на транспортний засіб або кур'єра. Система обчислює оптимальний маршрут, враховуючи чергу нерозподілених замовлень, геопозицію автопарку та дорожню ситуацію;

3. *Перевезення.* Фізичне переміщення вантажу. Супроводжується безперервною генерацією телеметричних подій, перерахунком прогнозованого часу прибуття та трансляцією оновлень у клієнтські інтерфейси;
4. *Вручення.* Фіксація факту передачі вантажу отримувачу (електронний підпис, фотофіксація, OTP-код), ініціалізація фінансових взаєморозрахунків та переведення замовлення у кінцевий стан.

1.1.2. Обробка виняткових ситуацій та зворотна логістика

Реальний процес виконання замовлення рідко проходить ідеальним шляхом. Відмови клієнтів, хибні адреси, відсутність отримувача на місці або пошкодження вантажу генерують асинхронні події, що вимагають негайного втручання системи.

Процес зворотної логістики порушує початково сформований розклад: у транспортному засобі змінюється доступний об’єм та вантажопідйомність, проте витрачається незапланований час на оформлення скасування чи повернення. Це вимагає механізмів динамічного перерахунку маршруту в реальному часі для мінімізації впливу локальних збоїв на глобальне виконання логістичного плану.

1.1.3. Координація учасників процесу

Ефективність бізнес-процесу критично залежить від синхронізації інформаційного контексту між трьома основними учасниками:

1. *Диспетчер (Адміністратор)*. Потребує цілісного огляду всієї системи з мінімальною затримкою. Його завдання – моніторинг аномалій, ручне втручання у нестандартних ситуаціях та перерозподіл навантаження;
2. *Кур’єр.* Взаємодіє з системою в умовах нестабільного мережевого з’єднання. Мобільний додаток має підтримувати автономну роботу (offline-first) із локальним кешуванням та надійною синхронізацією стану при відновленні зв’язку;

3. *Клієнт.* Вимагає абсолютної прозорості. Клієнту критично отримувати миттєві сповіщення та мати доступ до відстеження місцеперебування кур'єра в реальному часі.

Невідповідність даних між цими вузлами призводить до каскадних збоїв, зниження лояльності та прямих фінансових втрат.

1.1.4. Проблеми масштабування монолітної архітектури під час пікових навантажень

Традиційні монолітні системи управління логістикою стикаються з фундаментальними обмеженнями при обробці пікових навантажень (наприклад, у періоди розпродажів)[1].

У моноліті завдання, що інтенсивно використовують процесор, такі як графові перерахунки маршрутів, жорстко конкурують за ресурси з операціями вводу-виводу, наприклад, з обробкою REST-запитів. Коли алгоритм оптимізації вичерпує всі доступні ресурси сервера, деградує продуктивність усього застосунку: API-шлюз перестає відповідати, диспетчери втрачають контроль над транспортним парком, а клієнти не можуть перевірити статус замовлення.

Крім того, неможливість масштабувати окремі компоненти системи змушує інженерів копіювати весь застосунок цілком замість виділення ресурсів виключно під вузьке місце. Це призводить до надлишкового використання серверних потужностей та стрімкого зростання інфраструктурних витрат.

1.1.5. Критичність часових вікон

Еволюція угод про рівень надання послуг (SLA) детермінує посилення вимог до часових параметрів логістичних операцій, що відображається у концептуальному переході бізнес-моделей від парадигми доставки протягом робочого дня до обслуговування в межах жорстко специфікованих часових інтервалів. З математичної точки зору, імплементація цього просторово-часового обмеження трансформує класичну задачу маршрутизації транспорту (VRP) у

вищою мірою комплексну задачу маршрутизації з часовими вікнами (VRPTW). Строге дотримання визначених часових параметрів виступає критичним фактором операційної ефективності системи. Зокрема, синхронізація часу вручення з присутністю реципієнта радикально знижує ймовірність повторних виїздів, тим самим мінімізуючи транзакційні витрати на зворотну логістику. Паралельно з цим, імплементація оптимізаційних алгоритмів класу VRPTW забезпечує максимізацію щільності покриття маршрутів, ефективно нівелюючи холостий пробіг транспортних засобів та непродуктивні періоди їхнього очікування. Додатковим вагомим фактором є нівелювання ризиків застосування прямих фінансових рестрикцій, які неминуче виникають у корпоративному сегменті у разі порушення регламентованих термінів. З огляду на високу алгоритмічну складність розв'язання задачі VRPTW, архітектурний дизайн платформи вимагає інкапсуляції цього обчислювального апарату в межах спеціалізованого, ізольованого вузла. Даний компонент проєктується для забезпечення можливості асинхронного споживання масивів вхідних параметрів із подальшим генеруванням та поверненням оптимізованих графових структур без блокування суміжних підсистем.

1.2. Огляд та порівняльний аналіз існуючих систем управління

Стрімкий розвиток електронної комерції та підвищення вимог клієнтів до швидкості доставки (від формату Next-Day Delivery до On-Demand та Same-Day) стимулювали еволюцію програмного забезпечення для управління логістикою. Сучасний ринок пропонує широкий спектр готових рішень, що розповсюджуються за моделлю SaaS. Зазвичай ці платформи надають стандартизований набір інструментів: панель диспетчера для маршрутизації, мобільний застосунок для кур'єра та модуль інформування клієнтів.

Проте, при проєктуванні високонавантажених корпоративних систем виникає фундаментальна проблема: універсальні SaaS-продукти проєктуються для задоволення потреб масового ринку. Вони базуються на усереднених бізнес-моделях, що робить їх недостатньо гнучкими для великих логістичних операторів

зі складною внутрішньою інфраструктурою. Щоб обґрунтувати необхідність розробки власного рішення, необхідно провести глибокий архітектурний аналіз провідних платформ за ключовими технічними критеріями.

1.2.1. Критерії архітектурного та функціонального оцінювання

Для об’єктивного порівняння існуючих систем визначено чотири критичні вектори оцінювання, які безпосередньо впливають на стабільність, масштабованість та інтеграційні можливості:

1. *Архітектурний патерн та ізоляція відмов.* Аналіз базової архітектури (моноліт проти сервіс-орієнтованої архітектури). Оцінюється здатність системи переживати часткові збої (наприклад, відмову модуля геокодування) без повної зупинки процесу маршрутизації.
2. *Моделі інтеграції та асинхронна комунікація.* Оцінка механізмів обміну даними із зовнішніми корпоративними системами (ERP, CRM, WMS). Аналізується підтримка сучасних патернів, таких як подіє-орієнтована архітектура через брокери повідомлень, на противагу традиційним REST API зі строгими лімітами частоти запитів.
3. *Ефективність розв’язання задачі VRPTW.* Здатність математичного апарату системи вирішувати NP-складну задачу маршрутизації з урахуванням багатовимірних обмежень: часових вікон, вантажопідйомності, габаритів та сумісності вантажів.
4. *Синхронізація станів у реальному часі*. Оцінка механізмів підтримки консистентності даних між сервером, мобільним клієнтом кур’єра та веб-інтерфейсом замовника (використання постійних з’єднань типу WebSocket чи SSE замість ресурсомісткого механізму періодичних опитувань – Polling).

Для детального розбору обрано чотири репрезентативні системи: глобальні платформи Onfleet та Tookan, спеціалізований математичний рушій OptimoRoute та сервіс «Мурашина логістика» (Ant-Logistics) від українського розробника.

1.2.2. Аналіз глобальної платформи Onfleet

Onfleet (США) є одним із найпопулярніших рішень на глобальному ринку, фокусуючись на якості користувацького досвіду та предиктивній аналітиці. Платформа використовує моделі машинного навчання для прогнозування часу прибуття (ETA) на основі історичних даних про затори та швидкість роботи конкретних водіїв.

Основою взаємодії з Onfleet є синхронний REST API. Незважаючи на високу стабільність API-шлюзу компанії, такий підхід стає "вузьким місцем" при масштабуванні. Платформа накладає жорсткі обмеження на частоту запитів – зазвичай не більше кількох десятків запитів на секунду.

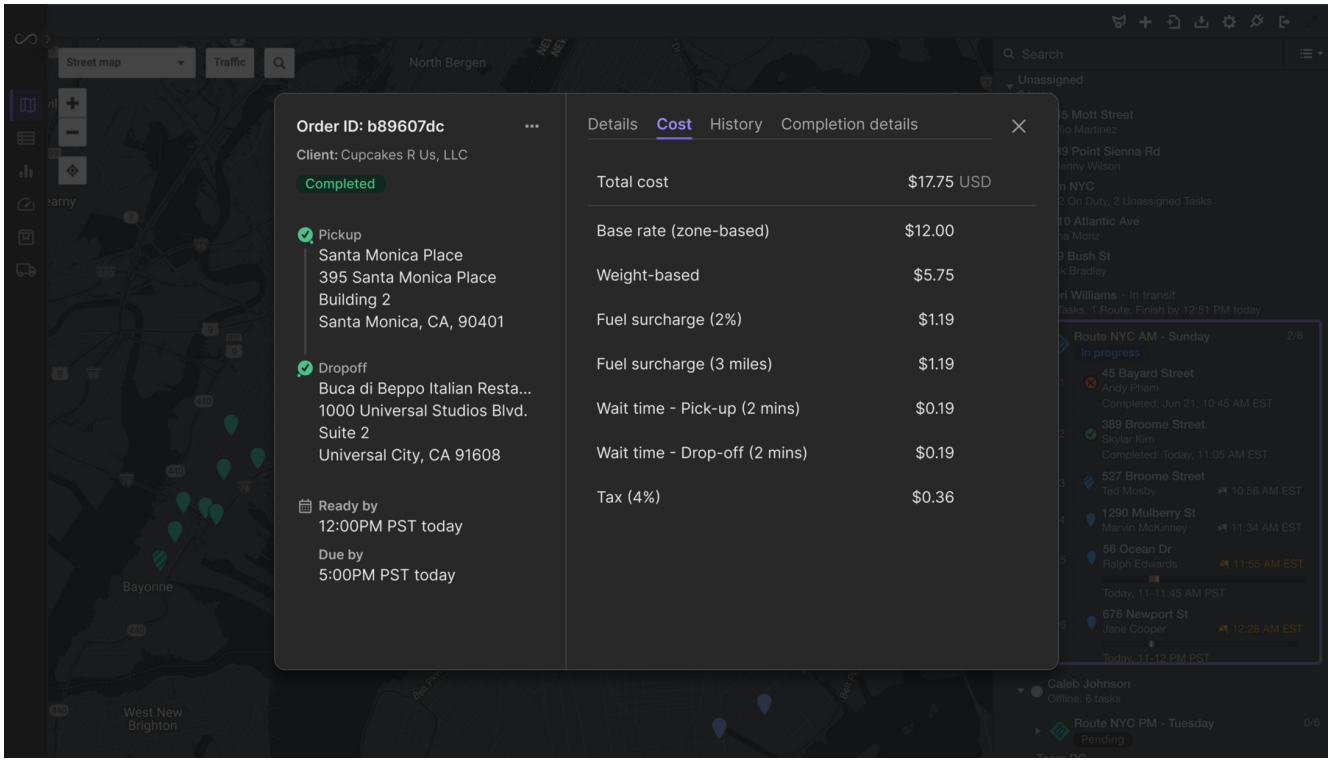


Рис. 1.2 Вигляд сторінка дашбороду Onfleet

В умовах великого підприємства, де кожна зміна статусу посилки має миттєво транслюватися в ERP-систему, ліміти API призводять до необхідності створення проміжних буферів на боці клієнта та затримки синхронізації. Крім того, система пропонує механізм вебхуків для сповіщень про події, що вимагає від підприємства підтримки публічно доступних, безперебійних HTTP-ендпоінтів. На

відміну від брокерів повідомлень, вебхуки не гарантують надійної доставки повідомлень у разі тимчасової недоступності сервера-отримувача, що може призвести до втрати критично важливих транзакцій.

З точки зору бази даних, закритість платформи не дозволяє модифікувати сутності під унікальні вимоги бізнесу. Скінченний автомат статусів замовлення є жорстко зашитим, що унеможливлює введення додаткових станів перевірки якості або багатиетапного сортування на транзитних хабах.

1.2.3. Аналіз екосистеми Tookan

Tookan (Індія) позиціонується як гнучка платформа для управління виїзними співробітниками. Її архітектура побудована навколо концепції "маркетплейсу розширень", що дозволяє додавати модулі сканування штрихкодів, підтримки платіжних шлюзів або інтеграції із зовнішніми картографічними сервісами.

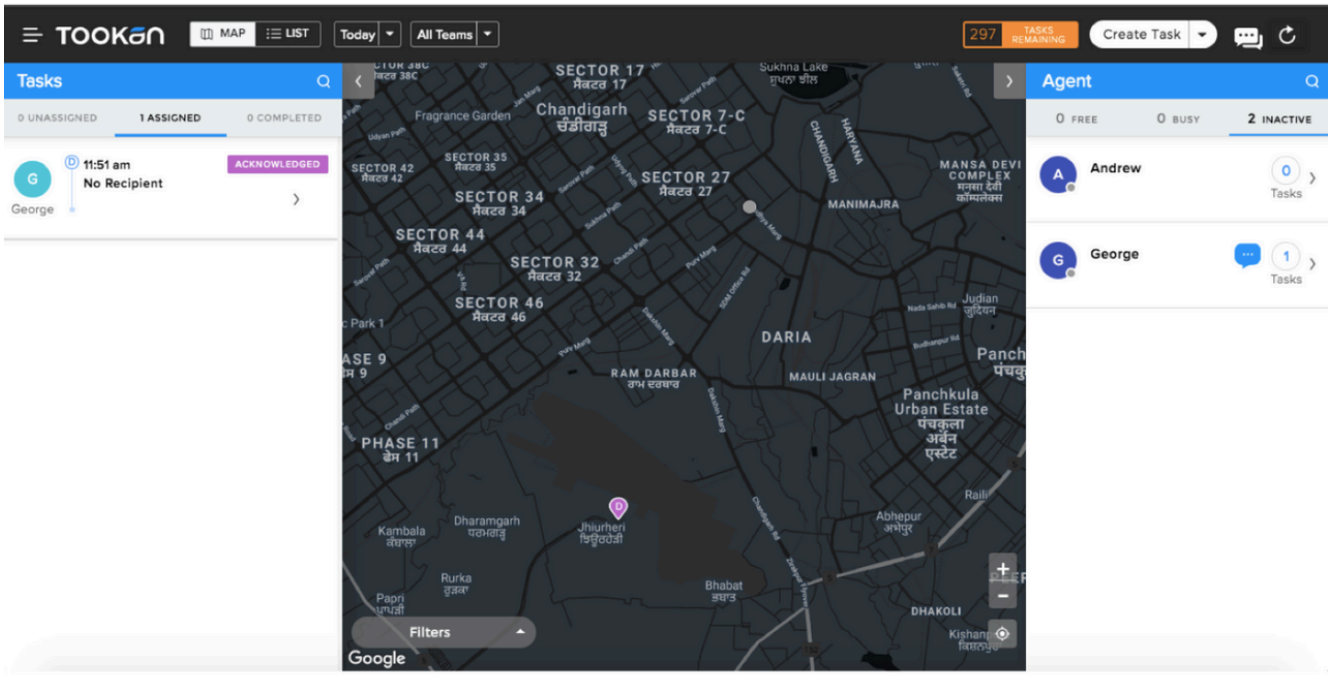


Рис. 1.3 Вигляд панелі Tookan

Незважаючи на видиму модульність інтерфейсу, базове ядро системи має яскраво виражені ознаки монолітної архітектури. Це особливо проявляється під час обробки масових транзакцій. Наприклад, щоранкове імпортування десятків тисяч

замовлень з ERP-системи створює колосальне навантаження на систему управління базами даних. Оскільки операції запису та перерахунку маршрутів виконуються в єдиному обчислювальному просторі, спостерігається деградація продуктивності всього сервісу: мобільні додатки водіїв починають відповідати із затримками, а диспетчерська панель може тимчасово зависати.

Алгоритмічний апарат Tookan використовує спрощені евристичні алгоритми маршрутизації (найчастіше – різновиди "жадібних" алгоритмів). Хоча такий підхід забезпечує швидке формування маршрутів для простих задач (наприклад, доставка їжі з найближчого ресторану), він виявляється неефективним при вирішенні задач із жорсткими часовими вікнами та складними матрицями відстаней для великого автопарку. Система схильна оптимізувати локальні ділянки маршруту, ігноруючи глобальний оптимум, що призводить до збільшення загального пробігу транспортних засобів та нераціонального використання робочого часу.

1.2.4. Аналіз спеціалізованого рушія OptimoRoute

OptimoRoute (Хорватія/США) має принципово інший архітектурний фокус. Замість розширення операційного функціоналу розробники зосередилися на створенні потужного ізольованого математичного рушія.

Серверна частина OptimoRoute здатна розв’язувати складні варіації задачі маршрутизації, обробляючи багатовимірні обмеження: навантаження на осі вантажівок, температурні режими, навички водіїв та багатопересічні часові вікна. Імовірно, на рівні внутрішньої реалізації платформи використовуються метаевристичні алгоритми (пошук із заборонами, імітація відпалу) або комерційні солвери.

Головним недоліком OptimoRoute є слабка адаптація до динамічних змін у реальному часі. Архітектура оптимізована під пакетне офлайн-планування. Розрахунок складного плану для сотень автомобілів може займати від кількох хвилин до півгодини. Якщо в процесі виконання маршруту виникає виняткова ситуація (наприклад, поломка автопарку або скасування замовлення клієнтом),

система не здатна миттєво перерахувати залишок маршрутів без переривання роботи всього автопарку.

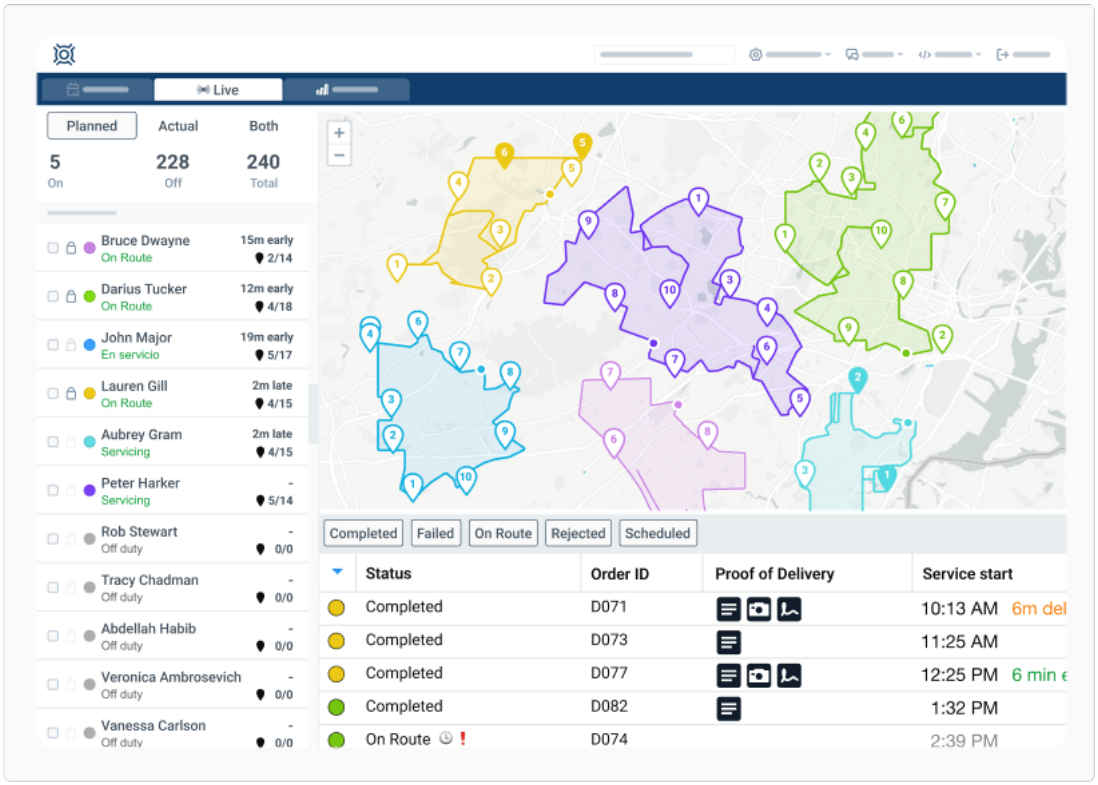


Рис. 1.4 Приклад інтерфейсу OptimoRoute

Крім того, механізм взаємодії мобільних клієнтів із сервером опирається на періодичне опитування замість утримання постійних з'єднань. Це створює надлишковий мережевий трафік, швидше розряджає акумулятори мобільних пристроїв кур'єрів та створює штучну затримку між фактичною подією та її відображенням у системі.

1.2.5. Аналіз локалізованого рішення «Мурашина логістика»

«Мурашина логістика» (Ant-Logistics, Україна) розроблялася з урахуванням специфіки локального ринку. Важливою перевагою системи є глибока інтеграція з вітчизняними картографічними сервісами, якісна нормалізація адрес, що не розпізнаються глобальними геокодерами (наприклад, Google Maps), та підтримка місцевих стандартів звітності.

Система побудована за класичною трирівневою архітектурою (Клієнт – Сервер додатків – Реляційна БД). Монолітна природа платформи накладає суворі обмеження на можливості інтеграції. Взаємодія з корпоративними ERP-системами зазвичай реалізується через файловий обмін (імпорт/експорт реєстрів у форматах XML/CSV) або через синхронний REST API із тривалим часом очікування відповіді.

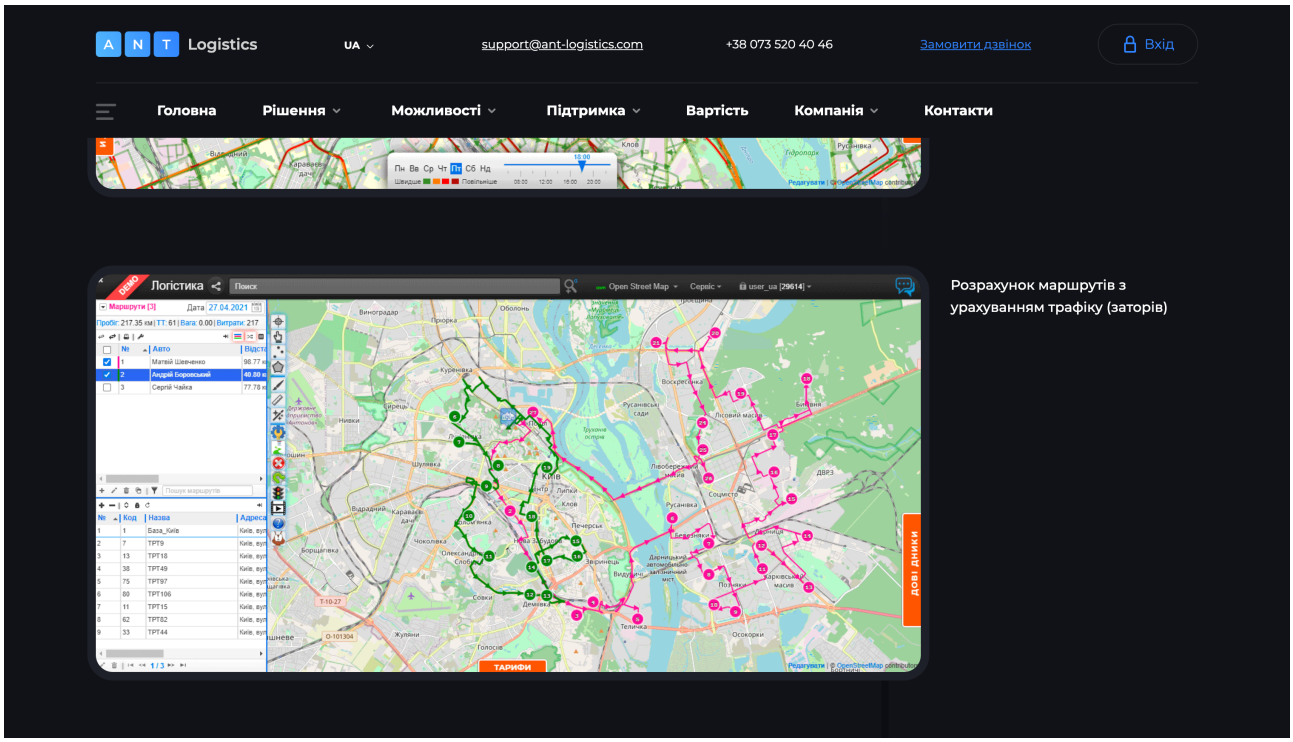


Рис. 1.5 Демонстрація можливостей Ant-Logistics

Такий підхід повністю блокує можливість побудови сучасної мікросервісної екосистеми на боці замовника. Відсутність прямої трансляції подій через сучасні шини даних робить неможливим побудову складних пайплайнів обробки даних (наприклад, одночасну трансляцію геопозиції водія в аналітичне сховище Data Warehouse та сервіс розрахунку бонусів). У випадку стрімкого масштабування бізнесу архітектурні обмеження цієї системи змушують компанії шукати інші рішення.

1.2.6. Зведена технічна характеристика

Для структурування результатів аналізу, ключові архітектурні та функціональні показники розглянутих систем зведено у порівняльну таблицю 1.1.

Детальний аналіз ринку підтверджує, що використання пропрієтарних монолітних систем створює критичні ризики для великих логістичних операторів. Насамперед, бізнес стає заручником технологічного стека постачальника (явище Vendor Lock-in). Оскільки жодна з розглянутих платформ не підтримує нативну взаємодію з брокерами повідомлень або gRPC-комунікацію, підприємства змушені будувати "тимчасові неоптимальні рішення" у вигляді проміжних адаптерів-перекладачів, які постійно опитують API сервісу. Це генерує "паразитне" мережеве навантаження та підвищує загальну відмову інфраструктури.

Таблиця 1.1

Порівняльний аналіз існуючих систем

Критерій	Onfleet	Tookan	OptimoRoute	Мурашина логістика
Базова архітектура	Закрита SaaS-платформа	Монолітне ядро з модулями	Ізольований рушій планування	Класичний трирівневий моноліт
Інтеграційний інтерфейс	REST API (строгий Rate Limit), Webhooks	REST API, базові Webhooks	API (орієнтація на Batch), Polling	REST API, імпорт/експорт файлів
Комунікація з клієнтами	WebSocket/SSE (real-time)	HTTP Polling / WebSockets	HTTP Polling (висока затримка)	HTTP Polling (запити за розкладом)
Математичний рушій VRPTW	Базові часові вікна	Евристичні ("жадібні") алгоритми	Повноцінний солвер обмежень	Базова підтримка обмежень
Динамічний перерахунок маршруту	Частковий (ручне втручання)	Частковий (за умовними правилами)	Відсутній (лише глобальний ребілд)	Відсутній в автоматичному режимі
Підтримка Event-Driven патернів	Відсутня (лише синхронні HTTP-виклики)	Відсутня	Відсутня	Відсутня
Архітектурне вузьке місце (Bottleneck)	Затримки синхронізації через ліміти API	Блокування БД при масових I/O операціях	Неможливість обробки даних у реальному часі	Несумісність із мікросервісними екосистемами

Економічний аспект також є суттєвим. SaaS-системи тарифікують використання за кількістю транзакцій (створених замовлень або розрахованих маршрутів). В умовах масштабування логістичного бізнесу витрати на оренду такого програмного забезпечення зростають експоненційно, значно перевищуючи витрати на підтримку власної хмарної інфраструктури.

З технічного погляду, головною проблемою існуючих рішень є відсутність гранулярного масштабування. У монолітних системах обчислювально важкі завдання (перерахунок графів, оптимізація матриць відстаней) виконуються в тому самому середовищі, що й обробка швидких запитів клієнтів.

Ці обмеження беззаперечно доводять необхідність проєктування кастомної платформи. Ідеальним рішенням є впровадження мікросервісної архітектури, яка дозволить застосувати принципи предметно-орієнтованого проєктування (DDD). Розподіл системи на незалежні вузли дозволить використовувати легковики і швидкі технології (наприклад, NestJS) для обробки мережевих запитів (API-шлюз), а важкі математичні розрахунки VRPTW делегувати спеціалізованим сервісам (наприклад, на мові Python з використанням бібліотеки OR-Tools).

Комунікація через асинхронні черги повідомлень забезпечить буферизацію навантажень та захист бази даних від перевантажень, гарантуючи безперебійну роботу системи навіть у періоди пікових навантажень.

1.3. Обґрунтування вибору мікросервісної архітектури

Як було продемонстровано під час аналізу існуючих рішень, монолітний підхід до побудови систем управління логістикою має фундаментальні архітектурні обмеження. Об’єднання в єдиному обчислювальному просторі операцій із високою конкурентністю мережевих запитів (збір телеметрії, опитування клієнтських додатків) та складних математичних розрахунків (маршрутизація з часовими вікнами) неминуче призводить до деградації загальної продуктивності. Для подолання цих бар’єрів та забезпечення високої масштабованості доцільно застосувати мікросервісну архітектуру, спираючись на принципи предметно-орієнтованого проєктування[2].

1.3.1. Декомпозиція системи на основі обмежених контекстів

Перехід до мікросервісної парадигми базується на декомпозиції складної логістичної системи на ізольовані обмежені контексти (Bounded Contexts)[3]. Замість монолітної структури даних та спільної бізнес-логіки, кожен контекст відповідає за конкретний бізнес-домен, інкапсулює власну модель даних і розвивається незалежно від інших компонентів системи.

Такий концептуальний розподіл (наприклад, відокремлення управління користувачами від обчислення маршрутів) мінімізує зв’язність коду та максимізує його цілісність. Це дозволяє уникати ситуацій, коли локальна зміна в логіці одного модуля призводить до непередбачуваних каскадних збоїв у суміжних підсистемах.

1.3.2. Ізоляція обчислювальних навантажень та поліглотне програмування

Одним із найвагоміших аргументів на користь мікросервісів у високонавантаженій логістиці є можливість ізоляції різних типів навантажень (I/O-bound та CPU-bound). Більшість компонентів системи вимагають ефективної обробки тисяч одночасних, але «легких» мережевих запитів, що потребує асинхронних, неблокувальних середовищ виконання. Водночас розв’язання задачі маршрутизації вимагає інтенсивних обчислень та роботи зі складними графами, що здатне заблокувати однопоточні процеси.

Мікросервісна архітектура дозволяє застосувати підхід поліглотного програмування. Це означає, що для кожного ізольованого домену можна обрати той технологічний стек і середовище виконання, які найбільш ефективно вирішують його специфічний клас задач, не нав’язуючи компромісних рішень усій системі.

1.3.3. Подіє-орієнтована архітектура

Для розв’язання проблеми жорстких лімітів синхронної взаємодії, властивих монолітам та закритим SaaS-рішенням, мікросервісна інфраструктура проєктується

на базі подіє-орієнтованого підходу. У цій парадигмі міжсервісна взаємодія здійснюється переважно асинхронно через проміжні брокери повідомлень.

Такий підхід забезпечує низку концептуальних переваг:

1. *Згладжування навантаження.* У періоди пікових навантажень запити акумулюються в чергах, звідки мікросервіси споживають їх із контрольованою швидкістю, захищаючи бази даних від перевантажень.
2. *Гарантована доставка.* У разі тимчасової недоступності певного вузла повідомлення зберігаються на рівні брокера до моменту відновлення працездатності споживача.
3. *Реактивність.* Компоненти системи генерують події зміни стану, а інші сервіси незалежно на них реагують, що дозволяє розширювати функціонал без модифікації наявного коду ядра.

1.3.4. Гранулярна масштабованість та відмовостійкість

Класичне горизонтальне масштабування моноліту вимагає клонування всього застосунку цілком, що призводить до надлишкового використання інфраструктурних ресурсів. Мікросервісна архітектура забезпечує гранулярне масштабування: при виникненні "вузького місця" ресурси виділяються виключно для того мікросервісу, який відчуває дефіцит пропускної здатності.

Крім того, розподілена природа системи гарантує високу доступність через ізоляцію відмов. Критичний збій у модулі генерації аналітики або сповіщень не блокує фундаментальні процеси збору телеметрії від кур’єрів чи реєстрації нових замовлень.

Таким чином, використання мікросервісної архітектури є не просто технічною оптимізацією, а стратегічною необхідністю, що забезпечує гнучкий, стійкий до навантажень та адаптивний фундамент для автоматизації логістичних процесів.

1.4. Математичні моделі маршрутизації транспорту

Автоматизація логістики останньої милі не обмежується створенням зручних інтерфейсів для диспетчерів чи мобільних додатків для кур'єрів; її фундаментальним ядром є математичний апарат, що відповідає за ефективний розподіл замовлень. В основі цього процесу лежить класична задача маршрутизації транспорту та її більш складна, наближена до реальних бізнес-вимог модифікація – задача маршрутизації з часовими вікнами. З погляду теорії обчислювальної складності, дана проблема належить до класу NP-складних задач [4]. Це означає, що явище комбінаторного вибуху робить неможливим знаходження абсолютно ідеального глобального оптимуму за поліноміальний час. Зі збільшенням кількості точок доставки точні математичні методи, такі як метод гілок і меж або динамічне програмування, вимагають експоненційного зростання обчислювальних ресурсів. Оскільки в умовах реального бізнесу розрахунок маршруту має виконуватися в умовах жорстких часових обмежень, сучасні підходи покладаються на строгі математичні моделі у поєднанні з оптимізаційними евристиками. Формалізація транспортної мережі починається з її подання у вигляді повного орієнтованого графа $G = (V, E)$. Множина вершин $V = \{0, 1, 2, \dots, N\}$ містить усі географічні точки, які беруть участь у логістичному процесі. Вершина 0 концептуально позначає центральний склад (депо) або розподільчий хаб, з якого стартує і куди обов'язково повертається транспорт після завершення робочої зміни. Множина вершин $V_C = \{1, \dots, N\}$ відповідає безпосереднім адресам клієнтів. Множина ребер $E = \{(i, j) \mid i, j \in V, i \neq j\}$ представляє всі можливі шляхи переміщення між будь-якими двома вершинами графа. Кожному ребру (i, j) зіставляється дві ключові вагові характеристики: вартість переходу c_{ij} (фізична відстань або грошовий еквівалент витрат) та час у дорозі t_{ij}. Для опису доступного ресурсу вводиться множина транспортних засобів $K = \{1,2, \dots, |K|\}$. Основною змінною прийняття рішень у цій моделі є бінарна змінна x_{ijk}. Вона набуває значення 1, якщо транспортний засіб k здійснює переїзд безпосередньо від вершини i до вершини j, та 0 – у будь-якому іншому випадку. Наступним кроком є визначення цільової

функції оптимізаційної системи. Головна мета полягає в мінімізації загальних операційних витрат логістичної компанії. У класичному вигляді цільова функція формулюється як мінімізація сумарної відстані, пройденої всім задіяним автопарком:

$$min \sum_{k \in K} \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ijk} \tag{1.1}$$

Проте в сучасних системах цільова функція часто набуває гібридного, багатокритеріального характеру. Наприклад, першочерговим завданням алгоритму може бути мінімізація кількості залучених автомобілів (що радикально знижує постійні витрати на оплату праці та амортизацію), а вже другорядним критерієм – оптимізація пробігу обраного пулу машин. Крім того, цільова функція може включати штрафи за порушення так званих "м'яких" часових вікон, коли запізнення допускається, але економічно пеналізується. Критичним аспектом математичної моделі, що дозволяє їй відповідати фізичним реаліям доставки, є система багатовимірних обмежень. Першим базовим правилом є обмеження цілісності маршруту. Воно гарантує, що кожен клієнт буде відвіданий рівно один раз і гарантовано одним кур'єром. Крім того, дотримується принцип нерозривності потоку: якщо транспортний засіб прибуває до вершини клієнта, він обов'язково повинен від неї від'їхати до наступної точки або повернутися в депо:

$$\sum_{k \in K} \sum_{j \in V} x_{ijk} = 1, \quad \forall i \in V_C \tag{1.2}$$

Другим фундаментальним правилом є обмеження вантажопідйомності, яке перетворює задачу на CVRP. Згідно з цим правилом, сумарна маса або об'єм замовлень q_i, які транспортує автомобіль k протягом одного рейсу, за жодних обставин не може перевищувати його максимальну фізичну вантажопідйомність Q_k:

$$\sum_{i \in V_C} q_i \sum_{j \in V} x_{ijk} \leq Q_k, \quad \forall k \in K \tag{1.3}$$

Третій, найбільш складний блок рівнянь, стосується безпосередньо часових вікон. Нехай s_{ik} позначає час початку обслуговування клієнта i кур'єром k. Згідно

з бізнес-вимогами, кожен клієнт i має жорстко заданий часовий інтервал $[e_i, l_i]$, де e_i визначає найраніший дозволений час початку контакту з клієнтом, а l_i – найпізніший допустимий час:

$$e_i \leq s_{ik} \leq l_i, \quad \forall i \in V_C, \forall k \in K \tag{1.4}$$

У випадку, якщо алгоритм вибудовує маршрут так, що транспортний засіб прибуває до локації клієнта раніше граничного часу e_i, кур'єр змушений очікувати настання дозволеного вікна. Цей час простою система також намагається мінімізувати. Часовий зв'язок між двома послідовними точками маршруту розраховується з урахуванням часу прибуття, тривалості самого процесу передачі посилки w_i та часу транзиту між точками t_{ij}. Математично цей зв'язок виражається через лінеаризацію умовного обмеження за допомогою методу достатньо великого числа M (Big-M method):

$$s_{ik} + w_i + t_{ij} - M\big(1 - x_{ijk}\big) \leq s_{jk} \tag{1.5}$$

Дана нерівність активується виключно тоді, коли автомобіль k дійсно їде від вузла i до вузла j (тобто $x_{ijk} = 1$). Якщо ж такого переходу не відбувається, велике значення M нівелює обмеження, роблячи його тривіально істинним для будь-яких часових показників. Теоретичне розв'язання такої моделі зазвичай базується на двоетапній алгоритмічній стратегії. На першому етапі використовуються конструктивні евристики, такі як алгоритм заощаджень Кларка-Райта або евристики найближчого сусіда та вставки. Їхня мета полягає у швидкій генерації первинного допустимого розкладу, який не порушує базових обмежень вантажопідйомності та часових вікон, хоча й може бути далеким від глобального оптимуму. На другому етапі застосовуються метаевристики для покращення знайденого рішення через механізми локального пошуку. У теорії маршрутизації найчастіше використовують алгоритми пошуку із заборонами, імітації відпалу та генетичні алгоритми. Ці методи ітеративно модифікують графи маршрутів – наприклад, обмінюючи вузли між різними гілками або перерозподіляючи замовлення між транспортними засобами – і використовують спеціальні механізми (такі як тимчасове погіршення цільової функції або накладання динамічних штрафів) для уникнення застрягання в точках локальних мінімумів. Використання

такої комплексної математичної моделі є необхідною умовою для створення стійкої системи автоматизації логістики, здатної масштабуватися в умовах мінливого середовища доставки. У практичній площині імплементація наведеної математичної моделі вимагає залучення спеціалізованих обчислювальних апаратів, здатних ефективно оперувати графовими структурами та системою багатовимірних обмежень. Сучасні програмні рішення для автоматизації маршрутизації використовують потужні фреймворки дослідження операцій, які інкапсулюють у собі описані конструктивні евристики та алгоритми локального пошуку. Такий підхід дозволяє абстрагувати складність низькорівневої реалізації метаевристик, переносячи фокус на декларативний опис цільової функції та специфікацію просторово-часових обмежень. Під час виконання математичний рушій конструює дерево пошуку рішень, застосовуючи парадигми програмування в обмеженнях та методи відсікання гілок для надшвидкого відкидання недопустимих конфігурацій логістичного графа. З архітектурної точки зору, інтеграція такого ресурсомісткого математичного апарату в розподілену програмну інфраструктуру вимагає суворої ізоляції обчислювального процесу. Оскільки розв'язання задачі маршрутизації є алгоритмічно складною операцією, що належить до класу NP-важких, відповідний модуль проєктується як незалежний воркер, орієнтований на максимальну утилізацію процесорного часу. Делегування графових обчислень в ізольоване середовище гарантує, що інтенсивні математичні операції не провокуватимуть блокування асинхронного циклу подій основних бекенд-сервісів, які відповідають за управління операціями введення-виведення. Взаємодія з цим обчислювальним вузлом організовується виключно через брокери повідомлень за принципом асинхронного обміну, що забезпечує загальну відмовостійкість кластера та його здатність до безперебійної обробки транзакцій під час пікових напливів замовлень.

2 АРХІТЕКТУРНЕ ПРОЄКТУВАННЯ СИСТЕМИ УПРАВЛІННЯ

2.1. Формування функціональних та нефункціональних вимог

Проєктування мікросервісної архітектури системи управління логістикою вимагає чіткої формалізації вимог, що випливають із проведеного у першому розділі аналізу предметної області. Основний акцент робиться на абстрагуванні від конкретних технологій та фокусуванні на архітектурних атрибутах, які дозволять подолати обмеження монолітних систем, забезпечити консистентність даних між ізольованими доменами та підтримати високу пропускну здатність під час пікових навантажень.

2.1.1. Функціональні вимоги

Функціональні вимоги описують бізнес-можливості системи, необхідні для забезпечення повного циклу доставки та ефективної координації між диспетчером, кур’єром та клієнтом.

1. *Управління життєвим циклом замовлення*
    1.1. Система повинна підтримувати строгий скінченний автомат переходів статусів замовлення (створено, розподілено, в дорозі, доставлено, скасовано/повернено).
    1.2. Забезпечення механізмів зворотної логістики: можливість ініціації повернення та автоматичний перерахунок доступної вантажопідйомності транспортного засобу.
2. *Динамічна маршрутизація з урахуванням часових вікон*
    2.1. Система повинна включати ізольований математичний модуль для розрахунку оптимальних графових маршрутів.

2.2. Алгоритм оптимізації зобов’язаний враховувати жорсткі часові вікна доставки, пріоритетність клієнтів, об’ємно-масові характеристики вантажу та пропускну здатність транспорту.

2.3. Наявність механізму часткового перерахунку маршруту в режимі реального часу при виникненні непередбачуваних подій (наприклад, поломка авто, відмова клієнта).

3. *Координація та моніторинг*

3.1. Забезпечення доступу диспетчерів до телеметрії автопарку та статусів замовлень у реальному часі без необхідності ручного оновлення інтерфейсу.

3.2. Можливість ручного втручання: примусове перепризначення замовлень або зміна часових вікон із подальшим автоматичним перерахунком глобального плану.

4. *Автономність та синхронізація*

4.1. Підтримка автономної роботи мобільного клієнта з локальним кешуванням маршрутного листа на випадок втрати з’єднання.

4.2. Фіксація факту доставки (електронний підпис, фотофіксація) та безперервна трансляція геолокаційних координат на серверний вузол.

5. *Клієнтський сервіс*

5.1. Надання клієнту тимчасового доступу до відстеження кур’єра на просторовій карті.

5.2. Автоматичне предиктивне сповіщення про зміну розрахункового часу прибуття.

2.1.2. Нефункціональні вимоги

Нефункціональні вимоги визначають архітектурні атрибути якості системи (QA). Вони формують базу для подальшого вибору патернів та інфраструктурних рішень.

1. *Масштабованість та ізоляція навантажень*

1.1. Архітектура повинна підтримувати гранулярне горизонтальне масштабування окремих доменів.

1.2. Необхідно фізично розділити обробку I/O-інтенсивних задач (підтримка десятків тисяч конкурентних підключень телеметрії) та CPU-інтенсивних задач (математична графова оптимізація маршрутів), щоб запобігти блокуванню основного циклу подій.

2. *Асинхронна комунікація та буферизація*

2.1. Взаємодія між мікросервісами повинна здійснюватися переважно асинхронно через проміжні брокери повідомлень. Це необхідно для згладжування пікових навантажень на бази даних.

2.2. Синхронна комунікація (через HTTP/RPC) має застосовуватися виключно для публічних інтерфейсів та операцій, що вимагають миттєвої консистентності (наприклад, перевірка авторизації чи первинна валідація DTO).

3. *Консистентність та відмовостійкість*

3.1. Для узгодження станів між розподіленими базами даних повинен застосовуватися патерн узгодженості в кінцевому рахунку та саги.

3.2. Система повинна зберігати часткову працездатність при збоях окремих вузлів. Відмова підсистеми сповіщень не повинна блокувати процес маршрутизації.

3.3. Необхідна імплементація механізмів гарантованої доставки та обробки «мертвих» черг повідомлень для нерозпізнаних чи помилкових транзакцій.

4. *Продуктивність роботи з даними*

4.1. Для збереження транзакційної цілісності фінансових та логістичних операцій (відповідність вимогам ACID) повинна використовуватися надійна реляційна система управління базами даних.

4.2. Для високочастотних операцій зчитування, керування сесіями та миттєвого геопросторового індексування координат автопарку вимагається застосування in-memory сховища (розподіленого кешу).

5. *Розгортання та інфраструктура*
   5.1. Усі програмні компоненти повинні бути ізольовані за допомогою технологій контейнеризації для забезпечення ідемпотентності середовищ розробки та промислової експлуатації.
   5.2. Керування життєвим циклом контейнерів має здійснюватися платформою оркестрації з підтримкою автоматичного автомасштабування на основі метрик (навантаження на CPU або черги повідомлень) та самовідновлення.

2.2. Проєктування мікросервісної інфраструктури та патернів взаємодії

Для забезпечення високої доступності, масштабованості та ізоляції обчислювальних навантажень, інфраструктура системи реалізована у вигляді набору незалежних контейнеризованих сервісів, розгорнутих у середовищі Kubernetes. Кожен мікросервіс відповідає за власний обмежений контекст і взаємодіє з іншими вузлами через строго визначені контракти зв’язку.

2.2.1. Топологія системи та розподіл відповідальності

Архітектура розробленої платформи ґрунтується на декомпозиції системи на чотири ключові обчислювальні вузли, кожен з яких структурно та технологічно оптимізовано для опрацювання специфічних векторів навантаження.

Першим компонентом виступає API-шлюз (API Gateway), імплементований на базі фреймворку NestJS та середовища виконання Bun. Цей вузол функціонує як єдина точка входу (Single Point of Entry) для всього зовнішнього клієнтського трафіку. Використання рантайму Bun дозволяє мінімізувати мережеві затримки під час виконання ресурсомістких операцій на рівні шлюзу, зокрема термінації протоколу SSL, криптографічної валідації JWT-токенів та балансування запитів.

Водночас API-шлюз забезпечує інкапсуляцію внутрішньої топології кластера, формуючи безпечний бар’єр між мікросервісами та зовнішніми споживачами.

Централізація бізнес-логіки реалізована в межах логістичного ядра (Logistics Core Service), яке також базується на стеку NestJS та Bun. Цей доменний вузол керує повним життєвим циклом замовлень, профілями користувачів та операціями автопарку. Важливою архітектурною конвенцією є надання цьому сервісу монопольного права на ініціацію транзакцій запису до критичних таблиць реляційної бази даних PostgreSQL, а також на централізоване управління переходами станів доменних сутностей (State Machine), що гарантує сувору консистентність даних.

Для вирішення оптимізаційних задач виділено ізольований математичний рушій – сервіс маршрутизації (Routing Solver Service), розроблений мовою програмування Python. Даний вузол спеціалізується виключно на обчислювально інтенсивних (CPU-bound) процесах. Із залученням бібліотеки Google OR-Tools сервіс здійснює побудову матриць відстаней та розв’язання задачі маршрутизації транспорту з часовими вікнами (VRPTW). Архітектурне рішення щодо ізоляції цих алгоритмів в окремому Python-воркері є критично важливим, оскільки воно унеможливлює блокування асинхронного циклу подій (Event Loop) основних I/O-орієнтованих бекенд-сервісів під час виконання тривалих графових обчислень.

Завершальним елементом топології є сервіс сповіщень та трекінгу (Notification & Tracking Service), побудований на NestJS. Цей вузол спроєктовано спеціально для обслуговування великого обсягу персистентних (довготривалих) з’єднань із мобільними терміналами кур’єрів. Для забезпечення ефективної маршрутизації транзакційних сповіщень та безперервної трансляції просторової телеметрії в умовах горизонтального масштабування кластера сервіс використовує брокер повідомлень Redis у режимі публікації та підписки (Pub/Sub).

2.2.2. Патерни міжсервісної комунікації

Організація взаємодії між компонентами обчислювального кластера базується на гібридній архітектурній моделі, що інтегрує синхронні та асинхронні

канали передачі даних залежно від експлуатаційних вимог до узгодженості та часу відгуку. Синхронний патерн комунікації, реалізований за допомогою архітектурного стилю REST API, імплементується виключно для транзакцій типу Request-Reply, які критично залежать від забезпечення миттєвої консистентності стану системи. Репрезентативним прикладом такого підходу є механізм перевірки прав: перед допуском запиту на ініціацію маршруту API-шлюз виконує блокуючий HTTP-запит до логістичного ядра для строгої валідації мандатів користувача згідно з моделлю управління доступом на основі ролей (RBAC).

Натомість фундаментальним транспортним рівнем для оркестрації тривалих та ресурсомістких бізнес-процесів виступає асинхронна взаємодія, побудована на базі брокера повідомлень RabbitMQ. Інфраструктурна конфігурація цього рівня спирається на використання топічних обмінників, що гарантує високу гнучкість та просторову детермінованість маршрутизації датаграм на основі ієрархічних ключів. Експлуатаційна модель вказаного механізму чітко простежується в алгоритмі обробки масивів замовлень. Під час надходження вхідного пулу заявок логістичне ядро здійснює їх первинну валідацію, персистує дані в реляційному сховищі зі статусом очікування та генерує доменну подію orders.batch.created. Мікросервіс маршрутизації, будучи підписником відповідної черги, асинхронно екстрагує корисне навантаження, виконує цільові математичні обчислення для побудови логістичного графа та публікує результати оптимізації як подію routes.calculated. На завершальному етапі транзакційного циклу логістичне ядро виступає споживачем цієї події, що ініціює консистентне оновлення статусів сутностей та фінальний запис сформованих маршрутних листів до постійного сховища бази даних.

2.2.3. Управління розподіленими транзакціями

Для збереження узгодженості даних між PostgreSQL та станами зовнішніх сервісів реалізовано патерн Сага з підходом хореографії (Choreography Saga). Замість використання глобальних блокувань на рівні баз даних, які б знизили

пропускну здатність системи, транзакційність підтримується через ланцюжок доменних подій.

У випадку переривання бізнес-логіки ініціюються компенсувальні транзакції. Наприклад, якщо Routing Solver не може знайти допустимий розв’язок задачі (через нестачу вільного транспорту або порушення часових вікон), він генерує подію routes.calculation.failed із зазначенням причини. Logistics Core перехоплює це повідомлення, скасовує попередньо зарезервовані слоти для замовлень та переводить їх у статус unassigned, генеруючи відповідний алерт для диспетчера.

2.2.4. Патерни відмовостійкості

Інфраструктурний базис досліджуваної системи спроєктовано з урахуванням парадигми терпимості до відмов (Design for Failure), що реалізується через імплементацію двох фундаментальних механізмів ізоляції та обробки виняткових ситуацій.

Вагомим елементом забезпечення надійності асинхронної комунікації є застосування політик маршрутизації недоставлених повідомлень (DLX) у поєднанні з алгоритмом експоненційної затримки на рівні брокера повідомлень RabbitMQ. У процесі функціонування системи, за умови виникнення помилки обробки події – зокрема, внаслідок тимчасової деградації зовнішніх картографічних API під час роботи обчислювального рушія маршрутизації – інфраструктура ініціює алгоритм повторної доставки з експоненційним збільшенням часового інтервалу. У випадку вичерпання ліміту спроб, детермінованого параметром часу життя (TTL), транзакція автоматично переспрямовується до спеціалізованої черги недоставлених повідомлень (DLQ). Цей підхід гарантує сувору персистентність критичних даних та формує надійний базис для їх подальшого ретроспективного аналізу експлуатаційним персоналом.

Паралельно з цим, для превентивного захисту синхронних каналів зв’язку від виникнення каскадних відмов, на рівні внутрішніх REST-клієнтів API-шлюзу інтегровано архітектурний патерн Circuit Breaker (Запобіжник). Логіка роботи

даного патерну спирається на безперервний евристичний моніторинг показників доступності: якщо питома вага серверних помилок класу HTTP 5xx або затримка відповіді від підпорядкованих мікросервісів (наприклад, логістичного ядра) перевищує встановлені порогові значення, механізм здійснює перехід у розімкнений стан. Як наслідок, усі наступні мережеві запити миттєво відхиляються з кодом статусу 503 Service Unavailable без ініціації реального мережевого виклику до деградованого сервісу. Таке архітектурне рішення ефективно запобігає вичерпанню пулу з'єднань у масштабах усього кластера та вивільняє системні ресурси, надаючи пошкодженому вузлу необхідне часове вікно для відновлення працездатності, що штатно реалізується шляхом автоматизованого перезапуску контейнера підсистемами оркестратора Kubernetes.

2.3. Моделювання структури баз даних та механізмів кешування

Фундаментом надійності будь-якої мікросервісної архітектури є грамотно спроєктований шар збереження даних. Для забезпечення цілісності бізнес-процесів логістики та дотримання принципів ACID при обробці транзакцій, основним персистентним сховищем обрано реляційну систему управління базами даних (СУБД). Взаємодія з БД реалізується через систему об'єктно-реляційного відображення (ORM), що гарантує строгу типізацію на рівні коду та мінімізує накладні витрати на мапінг сутностей. Водночас для високочастотних операцій, просторових розрахунків у реальному часі та буферизації застосовується спеціалізоване високошвидкісне in-memory сховище структур даних.

2.3.1. Реляційна модель даних

База даних спроєктована з урахуванням предметно-орієнтованого підходу.

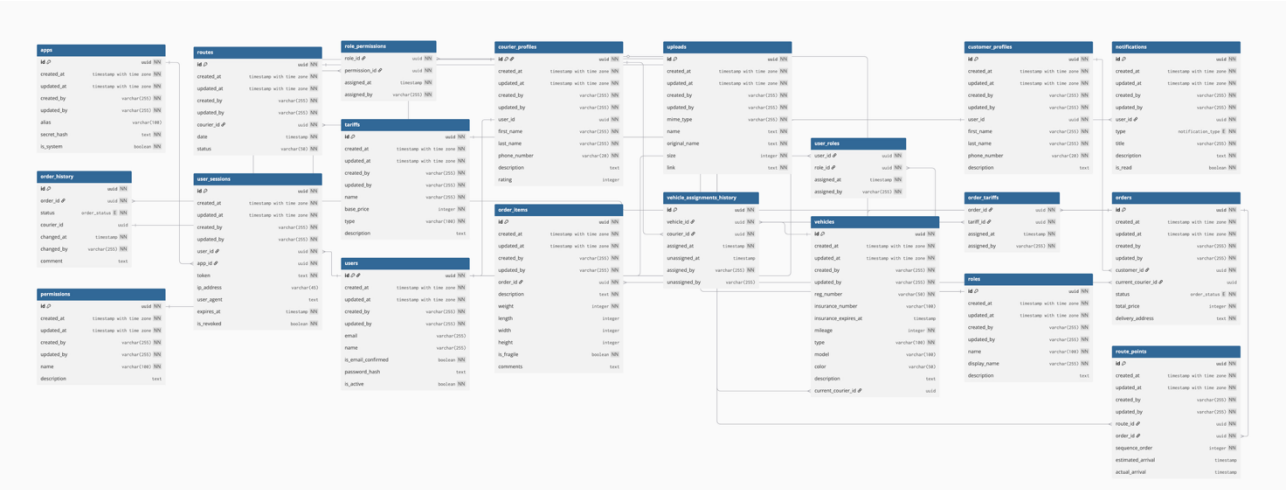


Рис. 2.1 Діаграма структури бази даних

З метою дотримання принципу DRY та забезпечення прозорості аудиту, кожна таблиця в системі (окрім специфічних junction-таблиць) містить стандартизований набір полів аудиту та ідентифікації:

Таблиця 2.1

Загальні колонки сутностей

Назва колонки	Тип	Обов’язковість	Опис
id	uuid	Так	Первинний ключ, генерується автоматично
created_at	timestamp	Так	Часові мітки створення запису
updated_at	timestamp	Так	Часові мітки останнього оновлення запису
created_by	varchar	Так	Ідентифікатори ініціаторів створення (UUID користувача або ідентифікатор мікросервісу)
updated_by	varchar	Так	Ідентифікатори ініціаторів оновлення (UUID користувача або ідентифікатор мікросервісу)

Першим логічним блоком системи є домен ідентифікації та управління доступом (IAM). Його центральною сутністю є таблиця користувачів, яка забезпечує базу для автентифікації всіх учасників процесу: від клієнтів до адміністраторів.

Таблиця 2.2

Структура сутності users (Користувачі)

Назва колонки	Тип	Обов’язковість	Опис
email	varchar(255)	Ні	Унікальний Email (дозволяє null для гнучкості реєстрації)
name	varchar(255)	Ні	Прізвище та ім’я (для персоналу)
is_email_confirmed	boolean	Так	Статус верифікації пошти
password_hash	text	Ні	Хеш пароля (Argon2id/Bcrypt)
is_active	boolean	Так	Прапорець активності акаунта

Для забезпечення безпечної взаємодії між різними клієнтськими застосунками (веб-кабінет, мобільні додатки) та API-шлюзом, необхідно вести реєстр самих додатків. Це дозволяє розмежовувати рівні доступу на рівні цілих платформ.

Таблиця 2.3

Структура сутності apps (Клієнтські додатки)

Назва колонки	Тип	Обов’язковість	Опис
alias	varchar(100)	Ні	Кодовий ідентифікатор застосунку (унікальний)
secret_hash	text	Так	Хеш-код для автентифікації додатка
is_system	boolean	Так	Ознака внутрішнього системного сервісу

Оскільки система передбачає роботу в розподіленому середовищі, критично важливо ефективно управляти сесіями користувачів. Таблиця сесій (Таблиця 2.4) дозволяє реалізувати механізм відкликання токенів та моніторинг активності з різних пристроїв, що є стандартом для сучасних Enterprise-рішень.

Таблиця 2.4

Структура сутності user_sessions (Сесії користувачів)

Назва колонки	Тип	Обов’язковість	Опис
user_id	uuid	Так	Зовнішній ключ на користувача (FK)
app_id	uuid	Так	Зовнішній ключ на додаток (FK)
token_id	uuid	Так	Унікальний ідентифікатор Refresh Token (jti)
ip_address	varchar(45)	Ні	Мережева адреса клієнта
user_agent	text	Ні	Дані про пристрій та браузер
expires_at	timestamp	Так	Час закінчення дії сесії
is_revoked	boolean	Так	Статус анулювання сесії

Для гнучкого управління доступом використовується модель RBAC. Вона складається з ролей (Таблиця 2.5) та конкретних дозволів (Таблиця 2.6), що дозволяє динамічно змінювати права груп користувачів без зміни коду системи.

Таблиця 2.5

Структура сутності roles (Ролі користувачів)

Назва колонки	Тип	Обов’язковість	Опис
name	varchar(100)	Так	Технічна назва ролі (наприклад, 'COURIER')
display_name	varchar(255)	Так	Назва ролі для відображення в інтерфейсі
description	text	Ні	Детальний опис повноважень ролі

Таблиця 2.6

Структура сутності permissions (Права)

Назва колонки	Тип	Обов’язковість	Опис
name	varchar(100)	Так	Кодова назва дії (наприклад, 'CANCEL_ORDER')
description	text	Ні	Опис того, що саме дозволяє це право

Наступним важливим доменом є профілювання учасників логістичного ланцюга. Розділення базового акаунта (users) та прикладних профілів дозволяє зберігати специфічну інформацію, необхідну суто для операційної діяльності, не перевантажуючи механізми автентифікації.

Таблиця 2.7

Структура сутності customer_profiles (Профілі клієнтів)

Назва колонки	Тип	Обов’язковість	Опис
user_id	uuid	Так	Посилання на акаунт користувача (1:1)
first_name	varchar(255)	Так	Ім’я клієнта
last_name	varchar(255)	Так	Прізвище клієнта
phone_number	varchar(20)	Так	Контактний телефон для зв’язку
description	text	Ні	Додаткова інформація (нотатки)

Профіль кур’єра, на відміну від клієнтського, містить додаткові метрики ефективності, такі як рейтинг. Ці дані згодом можуть бути використані алгоритмами автоматичного розподілу замовлень для вибору найбільш надійного виконавця.

Таблиця 2.8

Структура сутності courier_profiles (Профілі кур’єрів)

Назва колонки	Тип	Обов’язковість	Опис
user_id	uuid	Так	Посилання на акаунт користувача (1:1)
first_name	varchar(255)	Так	Ім’я кур’єра
last_name	varchar(255)	Так	Прізвище кур’єра
phone_number	varchar(20)	Так	Робочий номер телефону
rating	integer	Так	Поточний рейтинг (від 0 до 100)

Управління фізичними ресурсами (автопарком) є невід'ємною частиною логістики. Таблиця транспортних засобів зберігає не тільки технічні дані, а й поточну прив'язку до кур'єра, що дозволяє оперативно відстежувати відповідальність за майно компанії.

Таблиця 2.9

Структура сутності vehicles (Транспортні засоби)

Назва колонки	Тип	Обов'язковість	Опис
reg_number	varchar(50)	Так	Державний реєстраційний номер (унікальний)
insurance_number	varchar(100)	Ні	Номер страхового поліса
mileage	integer	Так	Поточний пробіг у кілометрах
type	varchar(100)	Так	Тип ТЗ (вело, авто, вантажівка)
current_courier_id	uuid	Ні	Кур'єр, за яким закріплено авто зараз (FK)

Для забезпечення прозорості та розбору інцидентів (наприклад, штрафів або пошкоджень) система веде детальну історію закріплення транспортних засобів. Це дозволяє дізнатися, хто керував автомобілем у будь-який момент часу в минулому.

Таблиця 2.10

Структура сутності vehicle_assignments_history (Історія закріплення ТЗ)

Назва колонки	Тип	Обов'язковість	Опис
id	uuid	Так	Первинний ключ (унікальний ідентифікатор запису)
vehicle_id	uuid	Так	Посилання на ТЗ (FK)
courier_id	uuid	Так	Посилання на кур'єра (FK)
assigned_at	timestamp	Так	Дата та час початку володіння
unassigned_at	timestamp	Ні	Дата та час повернення авто
assigned_by	varchar(255)	Так	Хто видав авто (адмін/система)

Центральним компонентом логістичної інфраструктури є склади (хаби). Окрім фізичної адреси, таблиця складів містить геопросторову точку (location), що є фундаментальною для розрахунку матриць відстаней у мікросервісі оптимізації.

Таблиця 2.11

Структура сутності warehouses (Склади)

Назва колонки	Тип	Обов’язковість	Опис
name	varchar(255)	Так	Назва складу/хабу
address	text	Так	Повна поштова адреса
location	geometry	Так	PostGIS точка (Point, SRID 4326)
working_hours	jsonb	Так	Графік роботи (JSON структура по днях)
is_hub	boolean	Так	Ознака центрального розподільчого центру

Ключовим об’єктом усієї системи є замовлення. Спроєктована таблиця orders інтегрує дані про клієнта, кур’єра, склад відправлення та параметри доставки.

Таблиця 2.12

Структура сутності orders (Замовлення)

Назва колонки	Тип	Обов’язковість	Опис
customer_id	uuid	Так	Посилання на замовника (FK)
status	enum	Так	Поточний статус (created, processing, etc.)
origin_warehouse_id	uuid	Так	Склад відвантаження (FK)
delivery_address	text	Так	Текстова адреса пункту призначення
location	geometry	Так	PostGIS координати точки доставки
delivery_window_start	timestamp	Так	Початок інтервалу доставки
delivery_window_end	timestamp	Так	Кінець інтервалу доставки

Поля delivery_window_start та delivery_window_end реалізують підтримку часових вікон, що є вихідними даними для задачі VRPTW.

Кожне замовлення може складатися з багатьох одиниць вантажу. Таблиця order_items описує фізичні характеристики кожної позиції (вага, габарити), що необхідно для розрахунку завантаженості транспортного засобу та вибору відповідного типу авто.

Таблиця 2.13

Структура сутності order_items (Елементи замовлення)

Назва колонки	Тип	Обов’язковість	Опис
order_id	uuid	Так	Посилання на батьківське замовлення (FK)
weight	integer	Так	Вага одиниці у грамах
length, width, height	integer	Ні	Габарити у міліметрах (L/W/H)
is_fragile	boolean	Так	Позначка «Крихкий вантаж»

Для відстеження прогресу доставки та аналізу ефективності роботи кур’єрів система фіксує кожен перехід статусу замовлення в окремому лозі. Це дозволяє будувати звіти про час обробки на кожному етапі.

Таблиця 2.14

Структура сутності order_history (Історія змін статусів)

Назва колонки	Тип	Обов’язковість	Опис
order_id	uuid	Так	Ідентифікатор замовлення (FK)
status	enum	Так	Статус, у який перейшло замовлення
changed_at	timestamp	Так	Час здійснення переходу
changed_by	varchar(255)	Так	Хто змінив статус (UUID або 'system')

Завершальним етапом проєктування реляційної моделі є домен маршрутизації. Маршрут (Таблиця 2.15) агрегує послідовність точок доставки для конкретного кур’єра на вибрану зміну, що фактично є реалізацією розв’язку задачі оптимізації.

Таблиця 2.15

Структура сутності routes (Маршрути)

Назва колонки	Тип	Обов’язковість	Опис
courier_id	uuid	Так	Виконавець маршруту (FK)
date	timestamp	Так	Планова дата виконання
status	varchar(50)	Так	Поточний стан маршрутного листа

Деталізація маршруту (Таблиця 2.16) відбувається в таблиці точок маршруту. Поле sequence_order визначає чітку черговість відвідування адрес, яку розрахував мікросервіс оптимізації маршрутів, а поля estimated_arrival та actual_arrival дозволяють порівнювати планову та фактичну продуктивність.

Таблиця 2.16

Структура сутності route_points (Точки маршруту)

Назва колонки	Тип	Обов’язковість	Опис
route_id	uuid	Так	Посилання на маршрут (FK)
order_id	uuid	Так	Посилання на замовлення у цій точці (FK)
sequence_order	integer	Так	Порядковий номер у черзі
estimated_arrival	timestamp	Ні	Розрахунковий час прибуття (ETA)
actual_arrival	timestamp	Ні	Фактичний час прибуття за даними GPS

Така деталізована реляційна модель забезпечує повне покриття бізнес-процесів кур’єрської служби та створює надійний фундамент для роботи мікросервісів логістики та математичної оптимізації.

2.3.2. Механізми кешування та обробки телеметрії

У розподіленій мікросервісній архітектурі взаємодія компонентів не обмежується лише транзакціями, що підлягають персистентному запису в

реляційну базу даних. Високочастотний потік даних, такий як телеметрія, проміжні результати обчислень алгоритмів маршрутизації та сесійні метрики, створює надмірне навантаження на підсистему вводу-виводу основної БД. Для розв'язання цієї архітектурної проблеми та забезпечення мілісекундної затримки (latency) інтегровано in-memory сховище типу ключ-значення (key-value).

Архітектура використання in-memory кешування у досліджуваній системі становить комплексну стратегію оптимізації продуктивності, що реалізується через чотири ключові вектори.

Першим напрямом є геопросторове індексування автопарку. З огляду на специфіку кур'єрського додатка, який генерує безперервний потік трансляцій GPS-координат із високою частотою, прямий запис цих даних у реляційну базу даних є неефективним через ризик вичерпання пулу з'єднань в умовах пікових навантажень. Для розв'язання цієї проблеми застосовуються нативні геопросторові структури in-memory сховища. У процесі надходження нових просторових даних від клієнтських додатків API-шлюз виконує оновлення локації безпосередньо в оперативному індексі. Відповідно, під час ініціації запитів від диспетчерської панелі або логістичного ядра на пошук найближчого транспорту, система здійснює просторовий пошук у межах заданого радіуса. Такий підхід забезпечує досягнення логарифмічної часової складності обчислювальної операції та гарантує мінімізацію затримок при отриманні результату.

Наступним важливим вектором оптимізації виступає буферизація обчислень матриці відстаней. Математичний апарат, що використовується для маршрутизації, потребує наявності повної матриці відстаней та часу транзиту між усіма вузлами графа доставки. Зважаючи на високу ресурсомісткість звернень до зовнішніх картографічних провайдерів для розрахунку ваги кожного ребра, in-memory сховище імплементується як розподілений кеш для попередньо обчислених сегментів. Роль ключа в цій структурі відіграє хеш-сума просторових координат двох точок, а значенням є розрахований час транзиту. У процесі роботи мікросервіс маршрутизації масово запитує наявні дані з кешу, ініціюючи дорогі обчислення виключно для невідомих ділянок. Згодом отримані результати асинхронно

записуються до оперативної пам’яті із заданим часом життя (TTL), що є необхідною умовою для врахування динаміки дорожньої ситуації.

Третій аспект архітектурної оптимізації стосується валідації сесій та забезпечення безпеки API. Попри те, що фундаментальна інформація про користувацькі сесії перманентно зберігається в реляційній базі даних, постійні звернення до дискової підсистеми під час кожного мережевого запиту утворюють вузьке місце на рівні API-шлюзу. Для усунення цієї проблеми ідентифікатори створених сесій реплікуються в оперативний кеш, де і відбувається їхня подальша швидка валідація. Окрім того, in-memory інфраструктура використовується як базис для механізму обмеження частоти запитів за алгоритмом Token Bucket, що дозволяє в автоматичному режимі ідентифікувати та блокувати клієнтів, які генерують аномально високий мережевий трафік.

Зрештою, критичним завданням є синхронізація WebSocket-з’єднань у розподіленому середовищі. Оскільки мікросервіс сповіщень функціонує в межах кластера з налаштованим горизонтальним масштабуванням, екземпляри сервісу створюються та знищуються динамічно, що ускладнює процес маршрутизації подій до конкретного клієнта. Для забезпечення коректної взаємодії in-memory сховище виконує функцію брокера публікації та підписки (Pub/Sub). Цей патерн забезпечує високу швидкість трансляції подій між усіма вузлами кластера та підтримує консистентність тимчасової карти мережевих підключень, гарантуючи миттєву та точно адресовану доставку оновлень на мобільні термінали кур’єрів.

2.4. Вибір технологічного стека та інструментальних засобів реалізації

Вибір технологічного стека для високонавантаженої логістичної системи базується на вимогах до відмовостійкості, швидкості масштабування та зручності підтримки коду. Для реалізації мікросервісної архітектури було обрано гібридний підхід, що поєднує екосистему TypeScript для обробки мережевих запитів та

інфраструктурної логіки з екосистемою Python для розв'язання складних математичних задач.

2.4.1. Середовище виконання та базова мова

Фундаментом розробки як бекенд, так і фронтенд частини є строга типізація за допомогою TypeScript. Це дозволяє реалізувати надійний мапінг DTO, інтерфейсів бази даних та спільних типів між мікросервісами, знижуючи ризик помилок під час виконання.

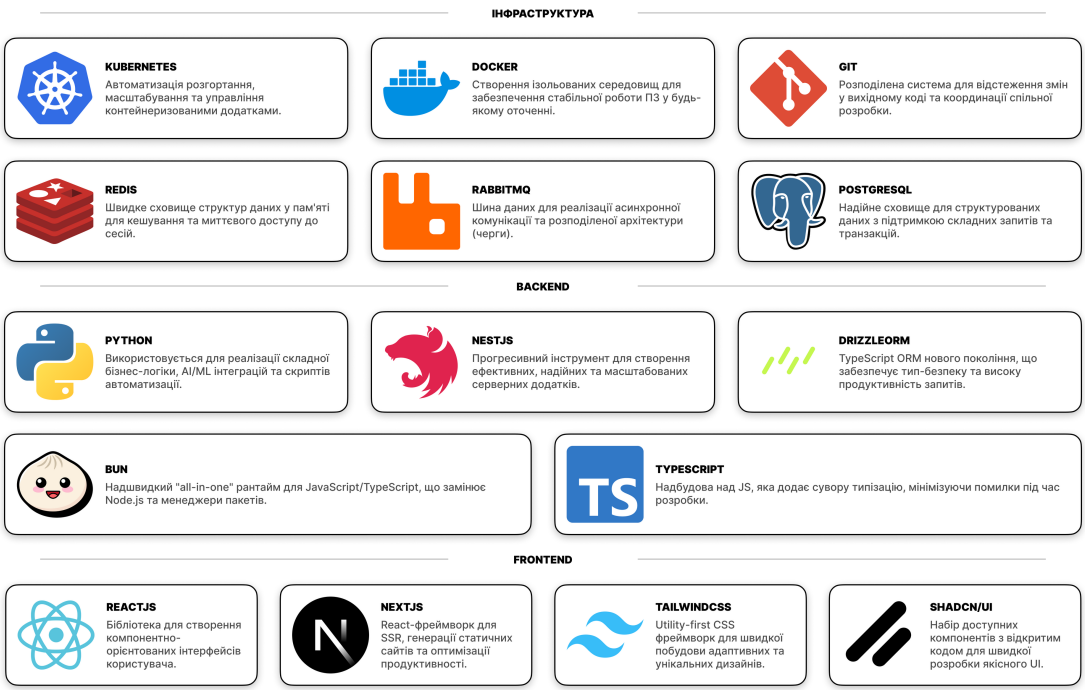


Рис. 2.2 Архітектурна схема технологій

Ключовим архітектурним рішенням під час проєктування системи стала відмова від використання класичного середовища виконання Node.js на користь сучасного рантайму Bun. На противагу Node.js, архітектура якого спирається на рушій V8 та вимагає залучення додаткових інструментальних засобів (зокрема, ts-node або tsc) для компіляції вихідного коду з TypeScript у JavaScript, середовище Bun розроблено з використанням мови програмування Zig та безпосередньо інтегрує рушій JavaScriptCore. Цей підхід забезпечує низку критичних оптимізаційних переваг для функціонування мікросервісної інфраструктури[5].

Передусім імплементація нативної підтримки TypeScript дозволяє рантайму виконувати програмний код без етапу попередньої транспаляції, що сприяє радикальному скороченню часових витрат на збірку Docker-образів. Вагомим експлуатаційним фактором також є суттєве зменшення часу ініціалізації: мікросервіси на базі Bun демонструють у 3–4 рази вищу швидкість холодного запуску порівняно з аналогами на Node.js. У контексті управління контейнерами за допомогою оркестратора Kubernetes ця характеристика набуває визначального значення, оскільки гарантує майже миттєве розгортання та перехід нових обчислювальних вузлів (подів) API-шлюзу до стану готовності обробляти вхідний трафік в умовах раптових пікових навантажень. Зрештою, підвищена продуктивність операцій введення-виведення (I/O), зумовлена глибокою внутрішньою оптимізацією вбудованого HTTP-сервера та механізмів обробки протоколу WebSockets, створює оптимальне середовище для мікросервісу сповіщень, специфіка якого вимагає стабільного утримання тисяч персистентних з’єднань із клієнтськими терміналами.

2.4.2. Бекенд-фреймворк та комунікація

Основним фреймворком для побудови API-шлюзу та логістичного ядра обрано NestJS [6]. Його модульна архітектура, вбудована підтримка Dependency Injection (DI) та строга орієнтація на принципи SOLID і DRY дозволяють будувати складні предметні області без перетворення коду на монолітну архітектурну деградацію.

RabbitMQ використовується як головний брокер повідомлень для асинхронної міжсервісної взаємодії [7]. Вибір RabbitMQ зумовлений підтримкою гнучкої маршрутизації (Exchange-RoutingKey) та надійним механізмом підтвердження доставки (acknowledgements), що є необхідним для реалізації розподілених транзакцій за патерном Saga.

Redis виконує роль проміжного In-memory сховища для геопросторових індексів (відстеження кур’єрів у реальному часі), механізмів Rate Limiting, а також

для буферизації матриць відстаней, розвантажуючи основну реляційну базу даних [8].

2.4.3. Сховище даних та ORM

Як було зазначено у попередніх розділах, центральним сховищем є PostgreSQL (з розширенням PostGIS) [9]. Для взаємодії з БД на рівні коду замість важких традиційних ORM (таких як TypeORM чи Prisma) обрано DrizzleORM [10].

Відмова від Prisma обґрунтована тим, що вона використовує у базовій архітектурі бінарний рушій на Rust, який споживає багато оперативної пам’яті в середовищі мікросервісів. DrizzleORM, натомість, є легким та незалежним (zero-dependency) інструментом, який використовує SQL-подібний синтаксис та генерує чисті SQL-запити без прихованих абстракцій. Це забезпечує максимальну продуктивність запитів, 100% типізацію схеми бази даних та повний контроль над складними JOIN-операціями.

2.4.4. Мікросервіс оптимізації маршрутів

Задача маршрутизації транспорту з часовими вікнами є NP-складною та вимагає інтенсивних процесорних обчислень. Виконання таких розрахунків у середовищі Node.js/Bun заблокувало б асинхронний цикл подій. Тому модуль оптимізації винесено в ізольований мікросервіс, написаний на Python. Вибір Python безальтернативний завдяки його лідерству у сфері Operations Research, а використання відкритої бібліотеки Google OR-Tools дозволяє швидко імплементувати сучасні евристики та метаевристики (наприклад, Guided Local Search) для розрахунку оптимальних маршрутів з урахуванням багатовимірних обмежень.

2.4.5. Клієнтський застосунок

Фронтенд для диспетчерської панелі та адміністративної частини реалізовано на базі бібліотеки React та фреймворку Next.js. Використання Next.js дозволяє

оптимізувати рендеринг сторінок, знизити навантаження на клієнтський пристрій та зручно управляти маршрутизацією.

Для побудови інтерфейсу користувача застосовується утилітарний CSS-фреймворк Tailwind CSS, який дозволяє стилізувати компоненти без створення сотень розрізнених CSS-файлів. UI-бібліотекою обрано Shadcn/ui – сучасний архітектурний підхід, який, на відміну від Material UI чи Ant Design, не постачається як npm-пакет. Shadcn/ui ін’єктує сирцевий код компонентів безпосередньо у проєкт, що дає розробнику повний контроль над логікою, доступністю (a11y) та зовнішнім виглядом без проблеми Vendor Lock-in.

Інтеграція підсистеми управління клієнтським станом та синхронізації даних у режимі реального часу відіграє критичну роль у забезпеченні консистентності диспетчерської панелі. Зважаючи на високу динаміку оновлення телеметричних показників та статусів замовлень, фронтенд-архітектура імплементує реактивні патерни управління станом у строгій комбінації з протоколом WebSockets. Такий архітектурний підхід забезпечує підтримку персистентного дуплексного каналу зв'язку з бекенд-інфраструктурою, дозволяючи миттєво відображати транзиції станів доменних сутностей на рівні клієнтського інтерфейсу без ініціації надлишкових HTTP-запитів, що пропорційно знижує загальне мережеве навантаження.

Окремим фундаментальним вектором оптимізації клієнтської частини виступає програмний механізм візуалізації геопросторових масивів даних, необхідний для репрезентації топології маршрутів та актуальної дислокації кур'єрського екіпажу. З огляду на об'єктивну ресурсомісткість рендерингу складних графових полігонів та великої кількості динамічних векторних об'єктів, на клієнтському рівні інтегрується спеціалізований картографічний апарат з апаратним прискоренням на базі WebGL. Делегування обчислень графічному процесору (GPU) диспетчерського термінала гарантує стабільну частоту розгортки кадрів під час масштабування картографічної проєкції та нівелює ризики деградації продуктивності інтерфейсу користувача навіть за умов граничного скупчення об'єктів моніторингу в єдиному сегменті.

2.4.6. Контейнеризація та оркестрація

Для забезпечення ідемпотентності середовищ (розробка, тестування, продакшен) усі компоненти системи (бази даних, брокери, фронтенд та кожен мікросервіс) запаковуються в ізольовані Docker-контейнери.

Делегування завдань з управління життєвим циклом контейнеризованих компонентів оркестратору Kubernetes формує інфраструктурний базис для забезпечення високої доступності та адаптивності досліджуваної системи. У межах розгорнутої архітектури платформа здійснює автоматизоване балансування вхідного мережевого навантаження між реплікованими екземплярами API-шлюзу, оптимізуючи утилізацію обчислювальних ресурсів. Рівень відмовостійкості програмного комплексу підвищується завдяки імплементації механізмів автоматичного відновлення які гарантують миттєвий перезапуск обчислювальних одиниць (подів) у разі виникнення критичних винятків на етапі виконання коду. Окрім цього, реакція системи на динамічні зміни обсягу обчислювальних потреб реалізується через підсистему горизонтального масштабування. Цей інструмент забезпечує динамічну провізію додаткових екземплярів мікросервісу оптимізації у відповідь на зростання телеметричних показників довжини черги повідомлень у брокері RabbitMQ, що є необхідною умовою для збереження стабільної пропускної здатності та нівелювання ризиків деградації продуктивності під час пікових навантажень.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ КОМПОНЕНТІВ СИСТЕМИ

3.1. Розробка ядра системи та API-шлюзу

Проєктування програмного ядра та вхідного API-шлюзу базується на парадигмі об’єднання цих двох компонентів у єдиний високонавантажений вузол. Таке рішення дозволяє мінімізувати мережеві затримки між етапом термінації зовнішнього запиту та його обробкою в доменній області логістики. Основою для реалізації обрано фреймворк NestJS, який виконується в середовищі Bun, що забезпечує нативну підтримку TypeScript та значно вищу швидкість запуску порівняно з класичним Node.js.

3.1.1. Ініціалізація програмного середовища та конфігурація вхідного вузла

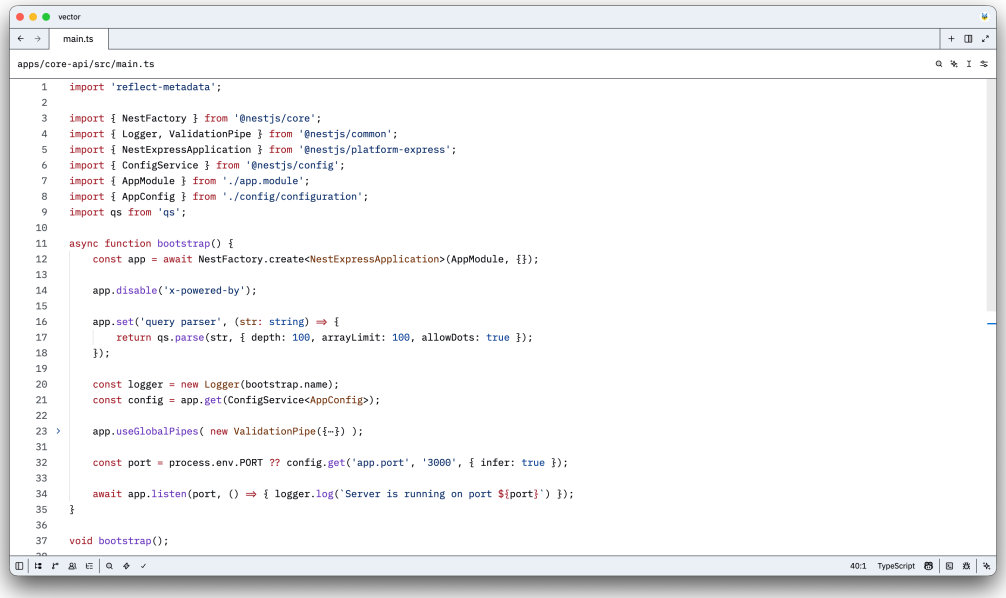


Рис. 3.1 Точка входу в систему

Точкою входу в систему є конфігураційний модуль main.ts, який ініціалізує екземпляр застосунку та встановлює глобальні правила обробки мережевого трафіку. Оскільки API-шлюз є публічним інтерфейсом системи, процес його

конфігурації вимагає імплементації строгих політик безпеки та валідації на найнижчому рівні.

Архітектурні рішення, імплементовані на етапі бутстрапінгу, вирішують три критичні задачі:

Безпека на рівні HTTP-заголовків. Виклик методу app.disable('x-powered-by') є базовим стандартом безпеки. Приховування інформації про стек технологій ускладнює проведення автоматизованих атак на інфраструктуру.

Адаптивний парсинг складних URL-запитів. Стандартний парсер запитів у Express має жорсткі обмеження на глибину вкладеності об’єктів. Логістична система вимагає складної фільтрації даних ресурсомістких сутностей. Заміна базового парсера на qs із конфігурацією depth: 100 та allowDots: true дозволяє клієнтським додаткам передавати складні багатовимірні параметри безпосередньо через Query String.

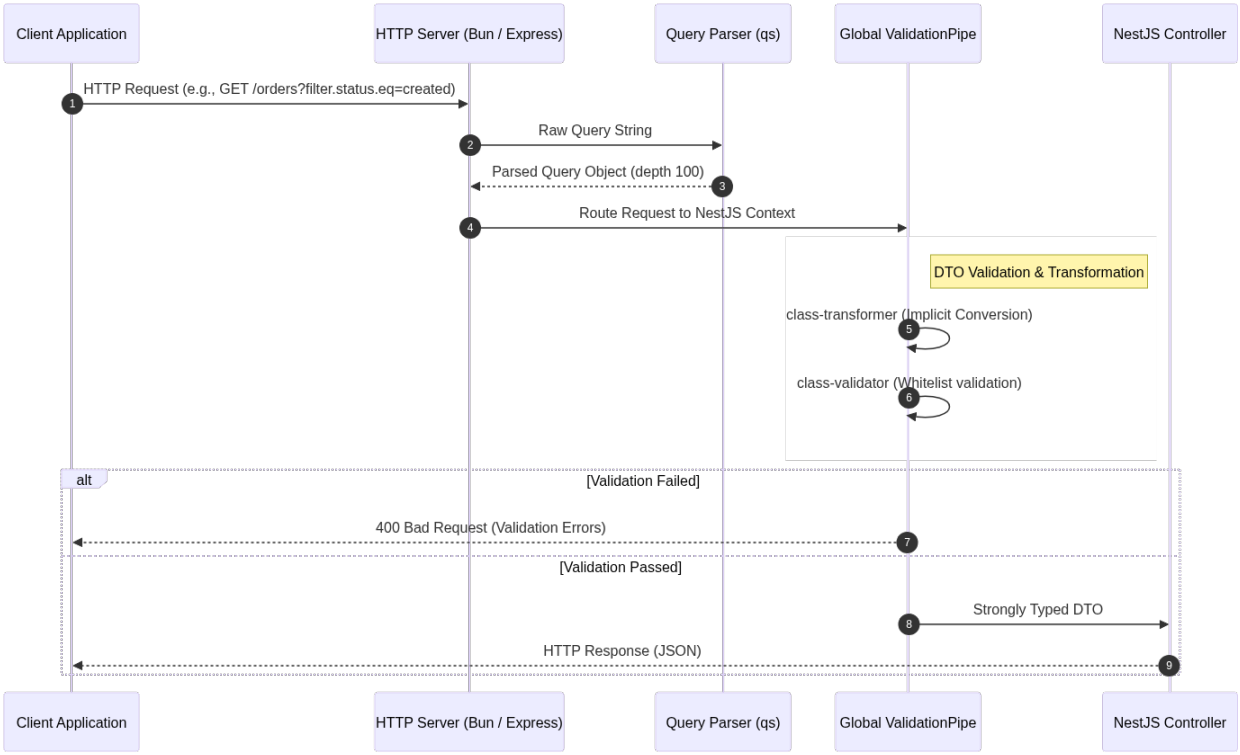


Рис. 3.2 Блок-схема потоку запиту

Глобальна строга типізація та валідація DTO. Реєстрація глобального ValidationPipe створює надійний бар’єр між зовнішнім середовищем та контролерами системи. Активація параметрів whitelist та forbidNonWhitelisted

гарантує відхилення будь-яких корисних навантажень (payloads), що містять непередбачені контрактом поля, нівелюючи ризики Prototype Pollution. Опція transform у поєднанні з enableImplicitConversion автоматично приводить примітивні типи мережевого протоколу до строгих типів TypeScript.

3.1.2. Багаторівнева система ідентифікації та управління доступом

У розподілених високонавантажених системах процес контролю доступу не обмежується лише перевіркою кінцевого користувача. API-шлюз як єдина точка входу повинен чітко ідентифікувати та верифікувати джерело кожного мережевого запиту. Для розв'язання цієї архітектурної задачі імплементовано гібридну модель безпеки, яка поєднує автентифікацію клієнтського застосунку та авторизацію користувача на базі JSON Web Token.

Процес ідентифікації починається з аналізу кастомних HTTP-заголовків X-App-Id та X-App-Key. Ідентифікатором застосунку може виступати як первинний ключ формату UUID, так і зрозумілий текстовий аліас системного сервісу. Після верифікації клієнтського середовища система переходить до валідації користувацького токена. Кожен JWT жорстко прив'язаний до контексту і містить у своєму корисному навантаженні ідентифікатор суб'єкта sub, унікальний ідентифікатор самого токена jti та аудиторію aud, значення якої має строго відповідати верифікованому ідентифікатору застосунку. При успішному проходженні обох етапів об'єкти app та user ін'єктуються безпосередньо в об'єкт мережевого запиту Request, стаючи доступними для всього подальшого ланцюга виконання.

Оскільки криптографічна перевірка підписів та постійні звернення до реляційної бази даних створюють вузьке місце під час пікових навантажень, система використовує агресивне кешування на базі in-memory сховища Redis. Комбінації ідентифікаторів та секретних ключів застосунків зберігаються в оперативній пам'яті для миттєвого доступу. Redis також виконує функцію розподіленого чорного списку, акумулюючи ідентифікатори jti відкликаних чи скомпрометованих токенів доступу. Для мінімізації затримок при перевірці

складних рольових моделей розраховані матриці прав доступу кешуються і підлягають інвалідації виключно при фізичній зміні конфігурації ролей у базі даних.

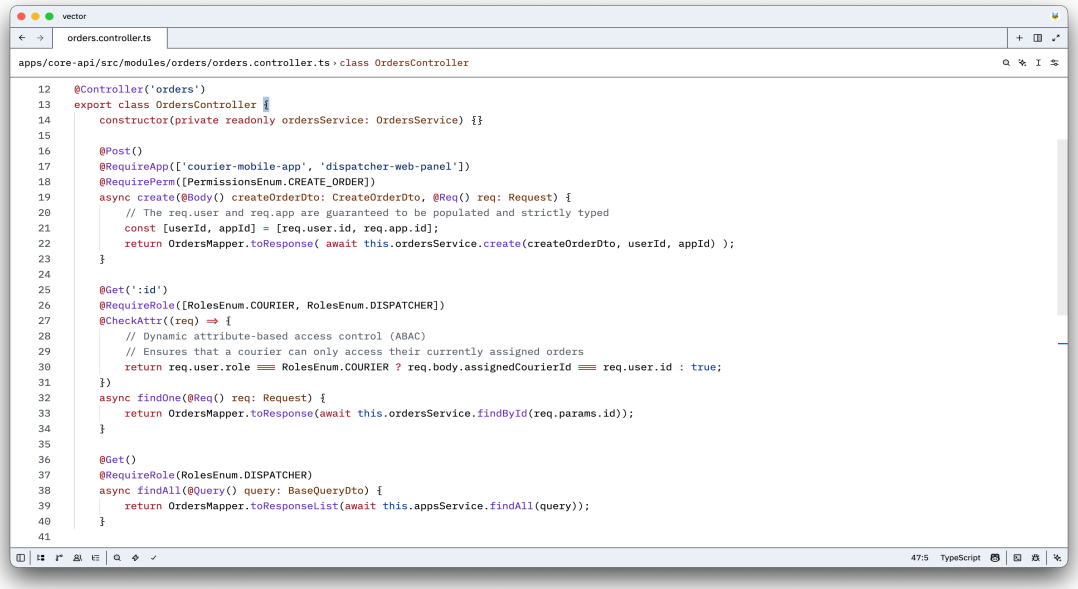


Рис. 3.3 Реалізація контролера замовлень

Для імплементації принципу DRY та забезпечення чистоти коду в контролерах розроблено набір кастомних декораторів. Вони реалізують декларативний підхід до маршрутизації, делегуючи складну логіку перевірок ізольованим гвардам NestJS.

Використання маперів забезпечує зменшення залежностей між схемою бази даних та публічним API. Це дозволяє вносити зміни у структуру таблиць без порушення зворотної сумісності для клієнтських застосунків.

Крім того, мапери спрощують агрегацію даних з декількох джерел або трансформацію форматів – наприклад, перерахунок грошових одиниць із найменших номіналів чи нормалізацію геопросторових координат, забезпечуючи консистентність відповідей на рівні всього API-шлюзу.

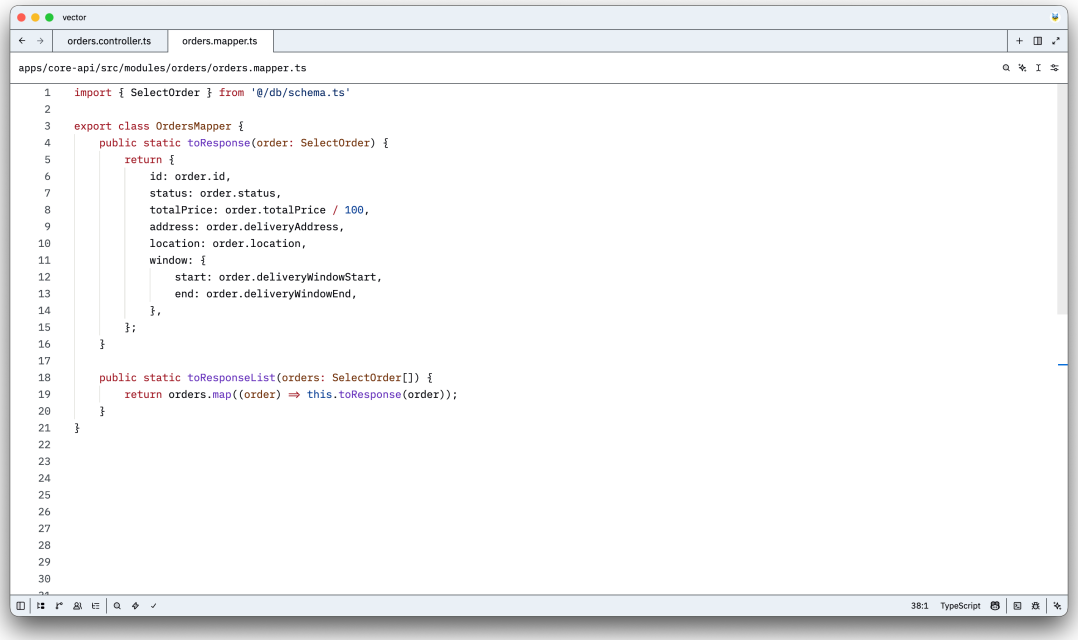


Рис. 3.4 Реалізація маперу для замовлень

3.2. Налаштування асинхронної комунікації та брокера повідомлень

Для реалізації подіє-орієнтованої архітектури та забезпечення надійної взаємодії між ізольованими доменами системи обрано брокер повідомлень RabbitMQ. На відміну від стандартного підходу з використанням інтегрованого пакета NestJS Microservices, інфраструктуру побудовано на базі кастомного модуля. Таке архітектурне рішення забезпечує повний контроль над топологією протоколу AMQP: явне управління обмінниками типу Topic, чергами, ключами маршрутизації та механізмами підтвердження доставки.

3.2.1. Конфігурація підключення та ініціалізація модуля

Підключення до RabbitMQ реалізовано через централізований сервіс конфігурацій, який гарантує строгу типізацію та валідацію змінних середовища на етапі запуску застосунку.

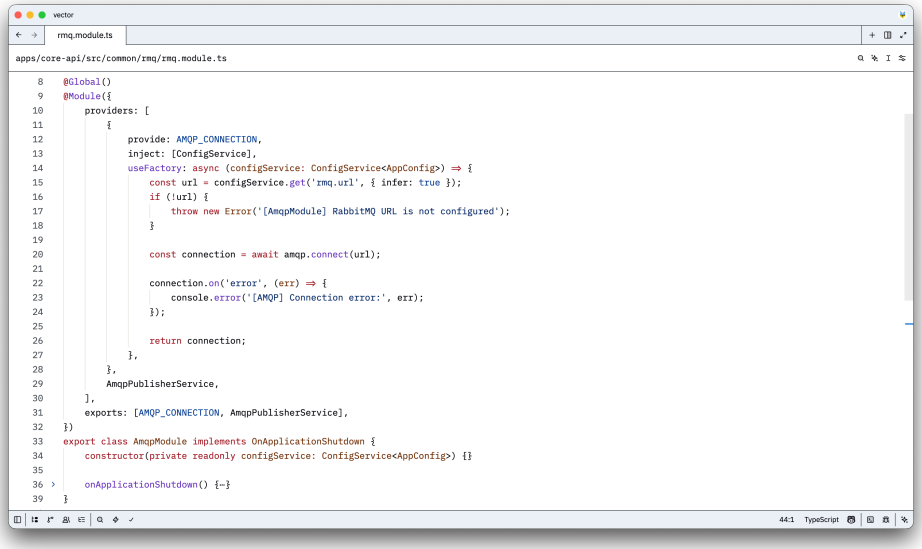


Рис. 3.5 Реалізація модуля RabbitMQ

Об’єкт конфігурації генерується автоматично, спираючись на визначені інтерфейси, що унеможливлює помилки доступу до неіснуючих ключів.

3.2.2. Типізація подій та контракти взаємодії

Для запобігання помилкам маршрутизації та забезпечення цілісності даних уся система повідомлень побудована на строгих контрактах. Ключі маршрутизації винесені в єдиний перелік, а структура корисного навантаження регламентується TypeScript-інтерфейсами. Це гарантує, що сервіс-публікатор та фонові воркери використовують ідентичний формат даних.

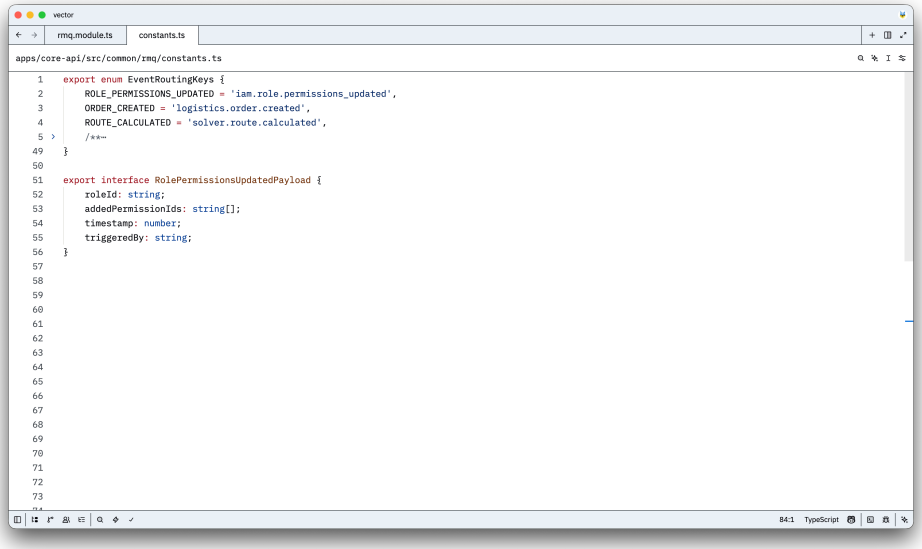


Рис. 3.6 Контракти подій

3.2.3. Імплементація механізму Publish-Subscribe

Практичне застосування асинхронної комунікації реалізовано на прикладі системи управління доступом. При додаванні нових дозволів до ролі необхідно інвалідувати кеш у Redis для всіх користувачів, які мають цю роль, та перерахувати їхні права. Виконання цієї операції синхронно в межах HTTP-запиту створює блокування та збільшує час відповіді сервера. Натомість сервіс оновлює реляційну базу даних DrizzleORM та публікує асинхронну подію.

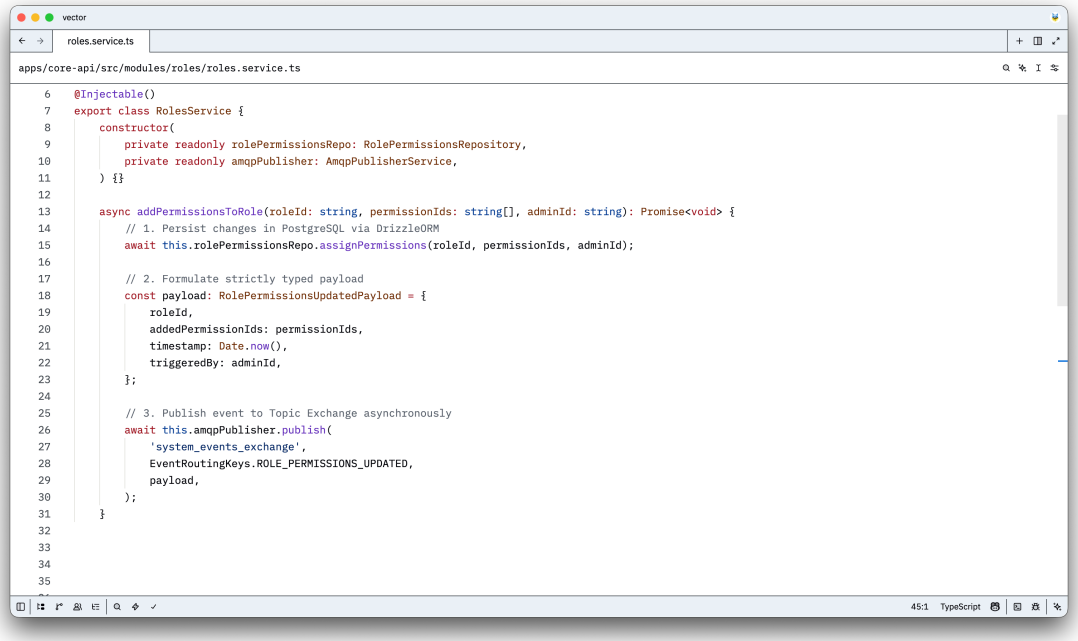


Рис. 3.7 Сервіс управління ролями. Приклад публікації події

Використання патерну Publish-Subscribe дозволяє делегувати обчислювально важку задачу інвалідації розподіленого кешу фоновим процесам. Оскільки рантайм Bun оптимізовано для неблокуючого вводу-виводу, публікація повідомлення в мережевий сокет RabbitMQ виконується майже миттєво, звільняючи основний потік виконання для опрацювання наступних мережевих запитів. Фонові мікросервіси, підписані на ключ iam.role.permissions_updated, самостійно споживають подію та виконують транзакції в Redis, забезпечуючи підсумкову узгодженість даних без деградації продуктивності API-шлюзу.

3.3. Реалізація логіки роботи з даними

Шар доступу до даних спроєктовано з урахуванням вимог високої продуктивності, строгої типізації та мінімізації накладних витрат на мапінг сутностей. Для взаємодії з PostgreSQL інтегровано DrizzleORM. На відміну від важковагових альтернатив, цей інструмент не містить прихованих рушіїв або бінарних залежностей, що робить його ідеальним вибором для швидкого середовища виконання Bun. Він компілює TypeScript-код безпосередньо у високопродуктивні SQL-запити, забезпечуючи абсолютну безпеку типів від визначення схеми до отримання результату.

3.3.1. Оптимізація структур даних та інтеграція PostGIS

При проєктуванні схем бази даних застосовано принцип DRY. Повторювані поля аудиту та ідентифікації винесено в окремі конфігураційні об’єкти. Це гарантує структурну консистентність усіх таблиць системи та спрощує подальший рефакторинг.

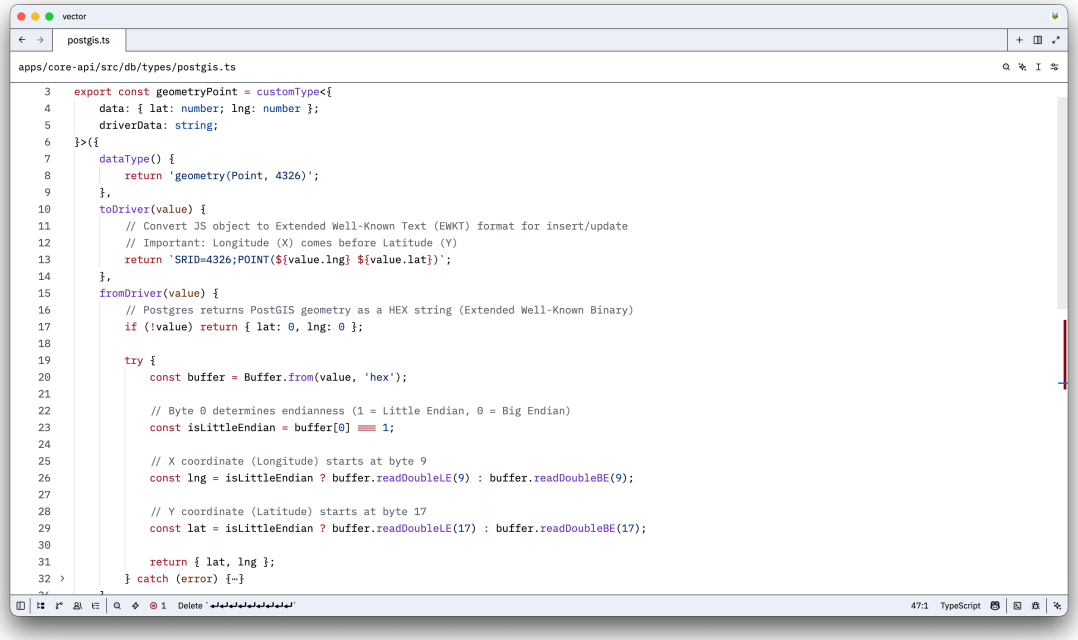


Рис. 3.8 Опис індивідуального типу для роботи з PostGIS

Критичним архітектурним викликом є обробка геопросторових даних для розрахунку логістичних маршрутів. Стандартні засоби ORM не мають нативної підтримки складних типів PostGIS. Для розв’язання цієї проблеми реалізовано кастомний тип geometryPoint, який виконує автоматичну серіалізацію та десеріалізацію координат між форматами Extended Well-Known Text та JavaScript-об’єктами.

Бінарний парсинг на рівні драйвера усуває необхідність додаткових трансформацій у бізнес-сервісах, забезпечуючи мілісекундну швидкість обробки телеметрії автопарку.

3.3.2. Уніфікація доступу через патерн абстрактного репозиторію

Для інкапсуляції SQL-запитів та уникнення дублювання CRUD-операцій імплементовано патерн абстрактного репозиторію.

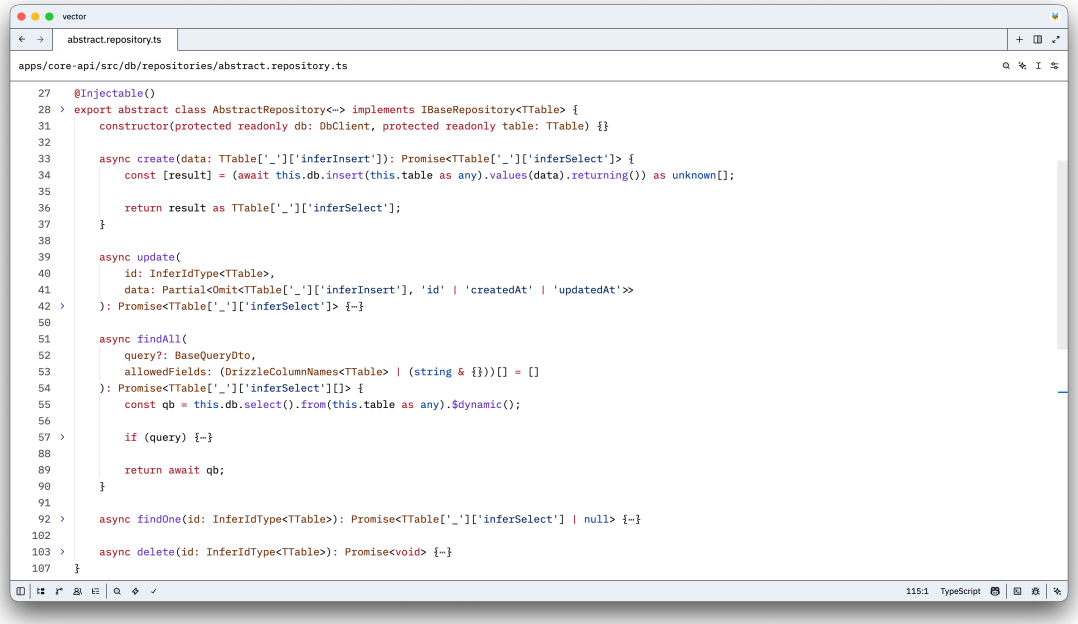


Рис. 3.9 Реалізація абстрактного репозиторію

Базовий клас AbstractRepository використовує розширені можливості TypeScript для динамічного виведення типів вставки та вибірки залежно від переданої схеми таблиці. Це ізолює доменні сервіси від прямої взаємодії з клієнтом бази даних.

Використання оператора returning дозволяє атомарно виконувати запис та зчитування оновленого стану сутності за одне мережеве звернення до бази даних.

3.3.3. Розширення абстракцій на рівні доменних репозиторіїв

Абстрактний клас слугує фундаментом, який розширюється конкретними імплементаціями для кожної доменної сутності. Успадкування гарантує наявність базового функціоналу, водночас дозволяючи реалізовувати специфічні бізнес-вибірки з використанням ін’єкції залежностей.

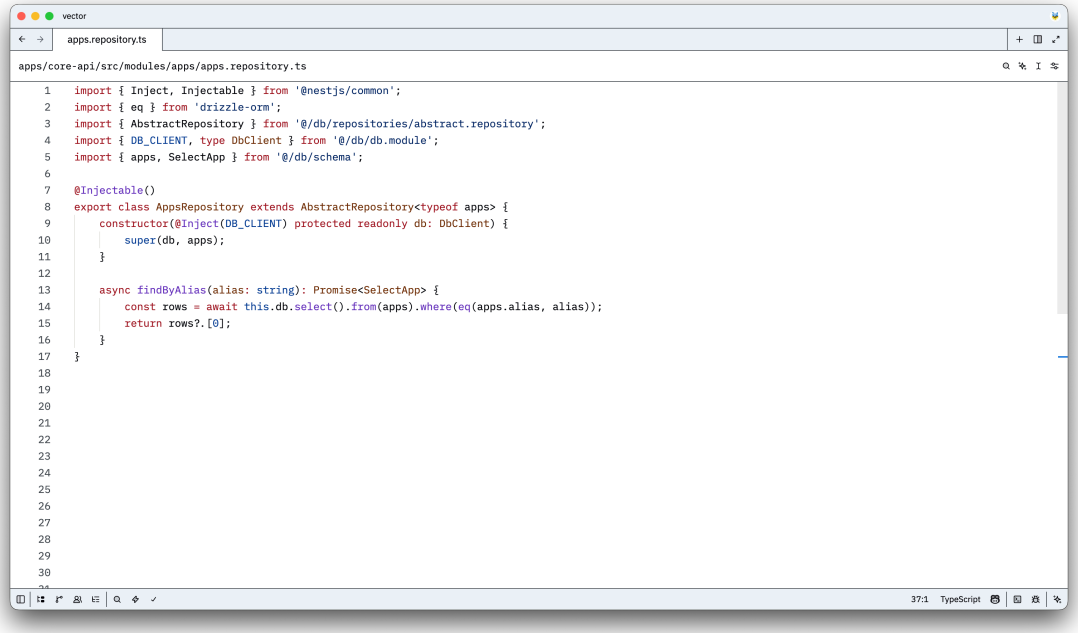


Рис. 3.10 Розширення абстрактного репозиторію на прикладі репозиторію застосунків

Модульність такого рішення дозволяє легко покривати шар роботи з даними модульними тестами, ізолюючи бізнес-сервіси від реальної бази даних шляхом імітації методів конкретного репозиторію. Уніфікація інтерфейсів взаємодії істотно пришвидшує масштабування логістичної системи при додаванні нових модулів.

3.3.4. Безпечна динамічна фільтрація та побудова запитів

Вимоги до API передбачають підтримку складних клієнтських запитів із фільтрацією, сортуванням та пагінацією. Конкатенація SQL-рядків є вразливою до

ін’єкцій. Тому розроблено утиліту QueryBuilderUtil, яка рекурсивно парсить об’єкт запиту та трансформує його у безпечні абстракції DrizzleORM.

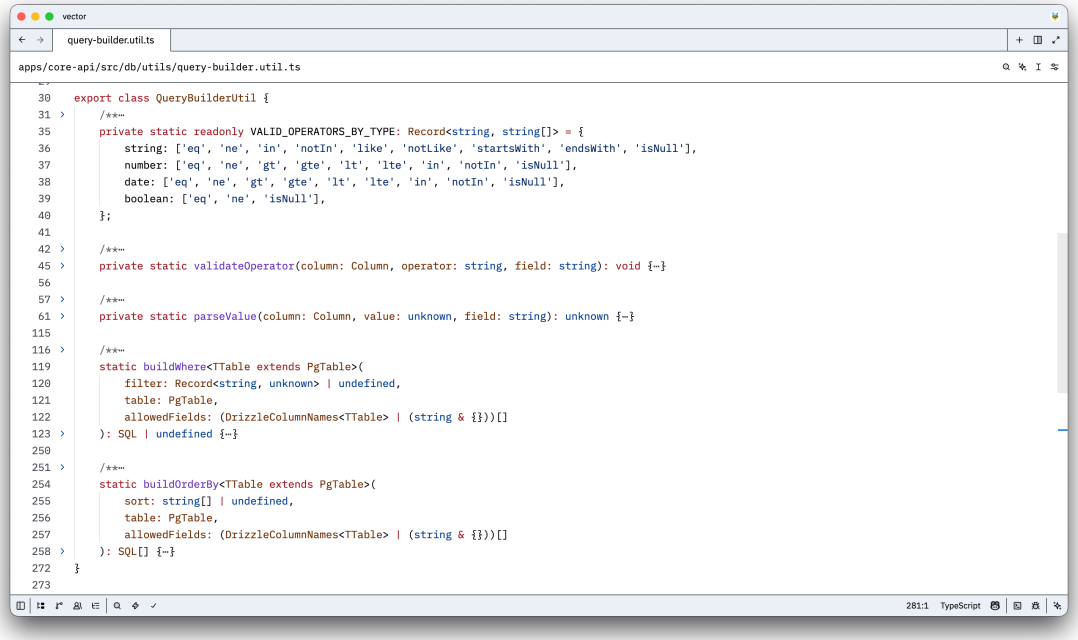


Рис. 3.11 Опис будівника SQL-запитів

Утиліта реалізує строгий білий список полів та автоматично зіставляє тип колонки з дозволеними операторами. Спроба застосувати оператор порівняння до текстового поля буде негайно відхилена з генерацією винятку на рівні обробки запиту.

Такий декларативний підхід ізолює контролери від специфіки SQL, забезпечуючи безпечну конвертацію мережевих параметрів у структуровані запити бази даних.

3.4. Розробка клієнтського веб-додатка

Клієнтський застосунок диспетчера виконує роль головного інструменту для моніторингу та управління логістичними процесами. Враховуючи високу динамічність даних та необхідність швидкого відгуку інтерфейсу, архітектуру побудовано на базі фреймворку Next.js. Стратегія рендерингу панелі управління свідомо зміщена в бік Client-Side Rendering (CSR), оскільки генерація таких

сторінок на стороні сервера (SSR) створює надлишкове навантаження і не дає переваг для закритих корпоративних систем, де SEO-оптимізація не вимагається.

3.4.1. Безпека комунікації та патерн Backend-For-Frontend

Взаємодія фронтенд-додатка з API-шлюзом відбувається через автоматично згенерований клієнт на базі Swagger-специфікації. Це гарантує строгу типізацію мережевих запитів та синхронізацію контрактів між клієнтом і сервером.

Для забезпечення безпеки застосовано проксі-підхід засобами Next.js App Router. Усі запити від клієнта спрямовуються на внутрішній маршрут /api/[[path]]. Цей проксі-шар валідує походження запиту (переконуючись, що ініціатором є саме клієнтський додаток, а не прямий виклик з браузера) та динамічно ін’єктує секретні заголовки X-App-Id та X-App-Key. Такий підхід інкапсулює облікові дані застосунку на серверному рівні Next.js, унеможливлюючи їх виток через клієнтський JavaScript-код.

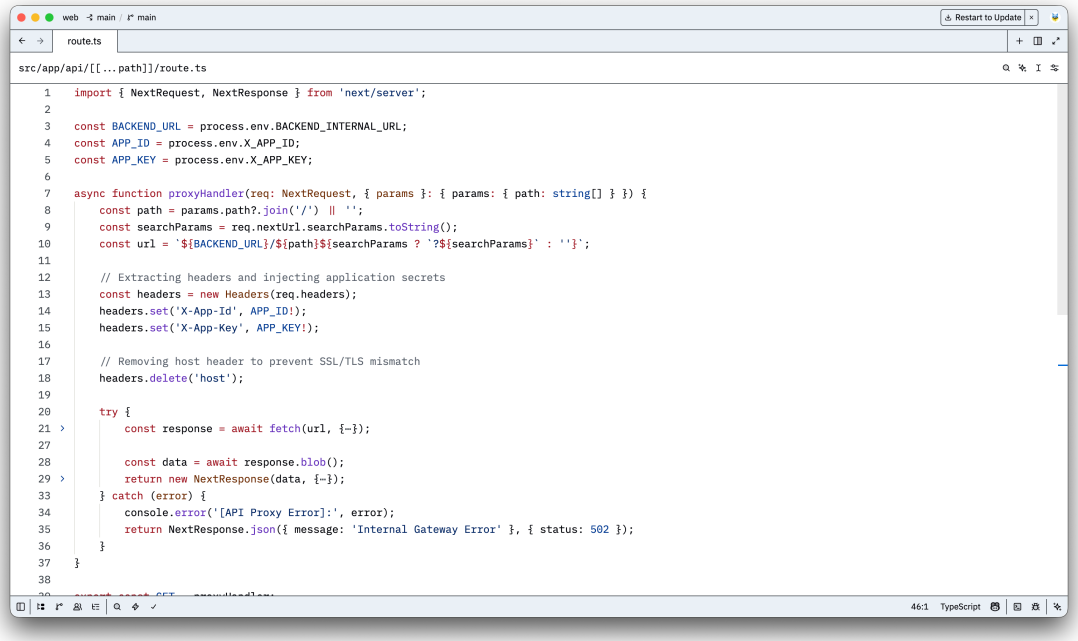


Рис. 3.12 Реалізація API проксі до нашого Backend

3.4.2. Інтерфейс користувача та архітектура компонентів

Побудова користувацького інтерфейсу спирається на утилітарний фреймворк TailwindCSS та архітектурний підхід shadcn/ui. Важливою архітектурною відмінністю є використання BaseUI замість традиційного RadixUI як фундаменту для "headless" компонентів. Ін'єкція сирцевого коду компонентів безпосередньо в репозиторій проєкту усуває проблему Vendor Lock-in та надає повний контроль над доступністю і логікою рендерингу.

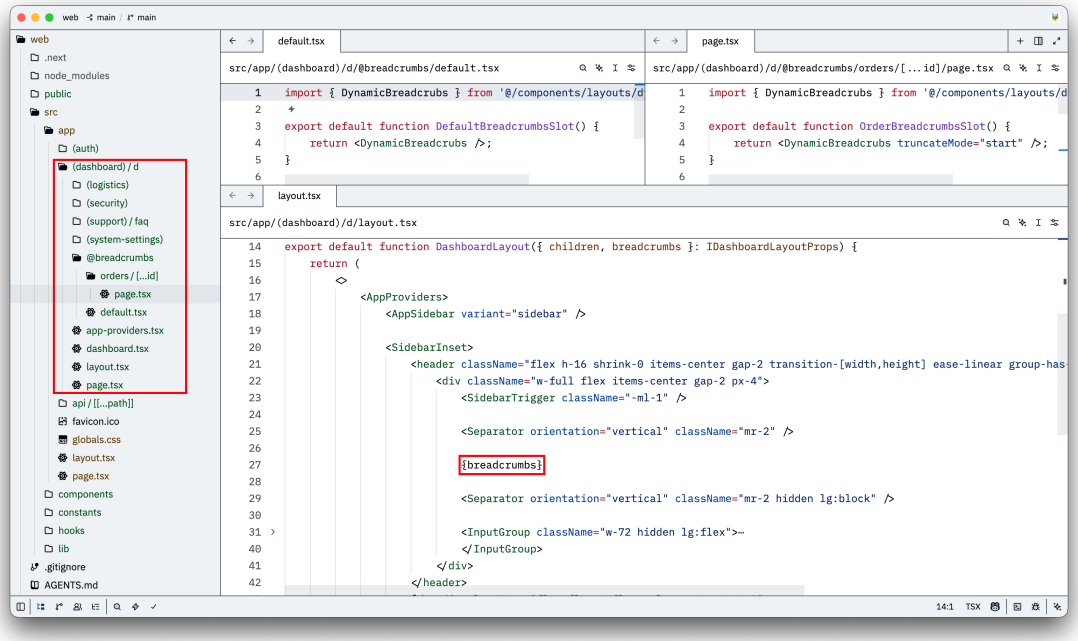


Рис. 3.13 Реалізація слотів та динамічного парсингу шляхів

Особливу увагу приділено системі навігації. Для реалізації динамічних "хлібних крихт" (Breadcrumbs) використано механізм паралельних маршрутів Next.js (@slots). Це дозволяє рендерити контекстно-залежну навігацію безпосередньо в базовому макеті (Layout). При переході на динамічні сторінки (наприклад, деталі замовлення [id]) система перехоплює UUID і форматує його, або ж виконує резолвінг читабельної назви сутності.

Використання слотів дозволяє декларативно змінювати поведінку навігації залежно від сторінки. Наприклад, для сторінки списку замовлень використовується стандартна конфігурація, а для детальної сторінки замовлення – обрізання з початку (truncateMode="start"), щоб сфокусувати увагу на ідентифікаторі.

Відповідальність за рендеринг та адаптивність делегується компоненту ResponsiveBreadcrumbs, який керує згортанням елементів у спадне меню при перевищенні ліміту, забезпечуючи коректне відображення на мобільних пристроях.

3.4.3. Управління складними даними: Таблиці та Форми

Оскільки логістичний інтерфейс оперує великими масивами даних, для рендерингу таблиць інтегровано «headless» бібліотеку TanStack Table.

Вона відокремлює логіку управління станом таблиці (фільтрація, сортування, пагінація) від її візуального представлення, що дозволяє створювати високопродуктивні дата-гріди, стійкі до перемальовувань.

Для обробки користувацького вводу (наприклад, створення нових замовлень або налаштування маршрутів) використовується екосистема TanStack Form у поєднанні з бібліотекою Zod. Такий тандем гарантує строгу типізацію форм від клієнта до бекенду, дозволяючи перевикористовувати одні й ті самі DTO-схеми як для валідації запитів у NestJS, так і для перевірки полів на фронтенді. Глобальне управління системними сповіщеннями реалізовано за допомогою бібліотеки Sonner, яка забезпечує неблокуючий, високопродуктивний рендеринг повідомлень про успішні транзакції або мережеві помилки.

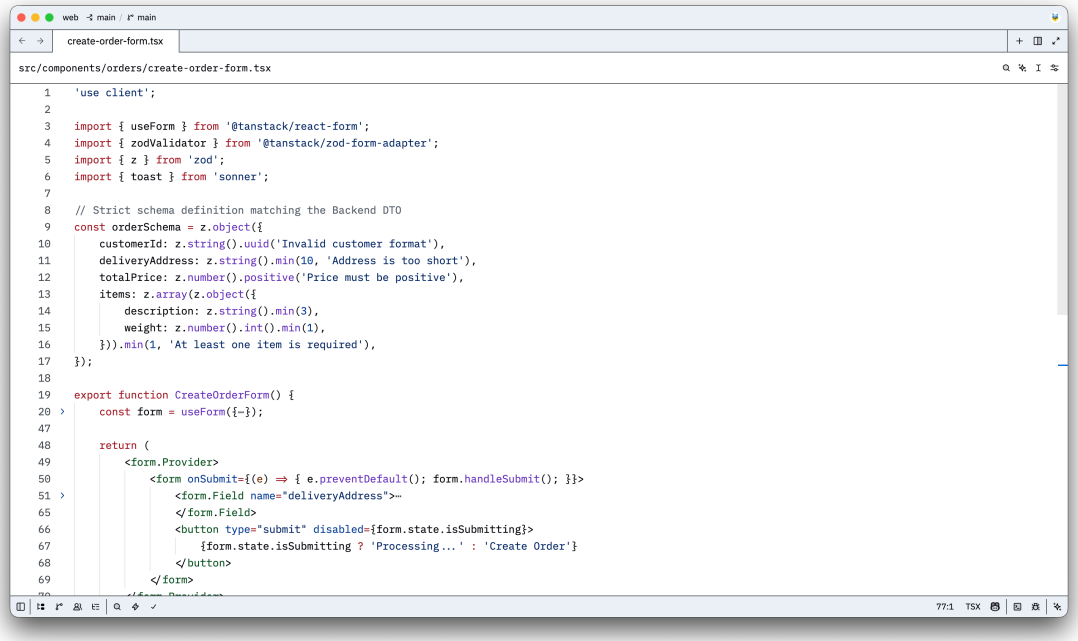


Рис. 3.14 Реалізація форми створення замовлення

3.5. Контейнеризація та оркестрація інфраструктури

Для забезпечення ідемпотентності середовищ від етапу розробки до промислової експлуатації та гарантування високої доступності, розроблена система використовує технології контейнеризації Docker та оркестрації Kubernetes. Перехід від локального розгортання за допомогою Docker Compose до розподіленого кластера є критичною вимогою для високонавантажених логістичних систем, де ручне управління життєвим циклом контейнерів є неефективним і створює ризики непередбачуваних простоїв.

3.5.1. Стратегія контейнеризації та оптимізація образів

Пакування мікросервісів здійснюється за допомогою багатоетапних збірок (Multi-stage builds). Цей підхід дозволяє відокремити середовище збірки, що містить компілятори та інструменти тестування, від кінцевого образу виконання, суттєво зменшуючи площу атаки та розмір артефакту.

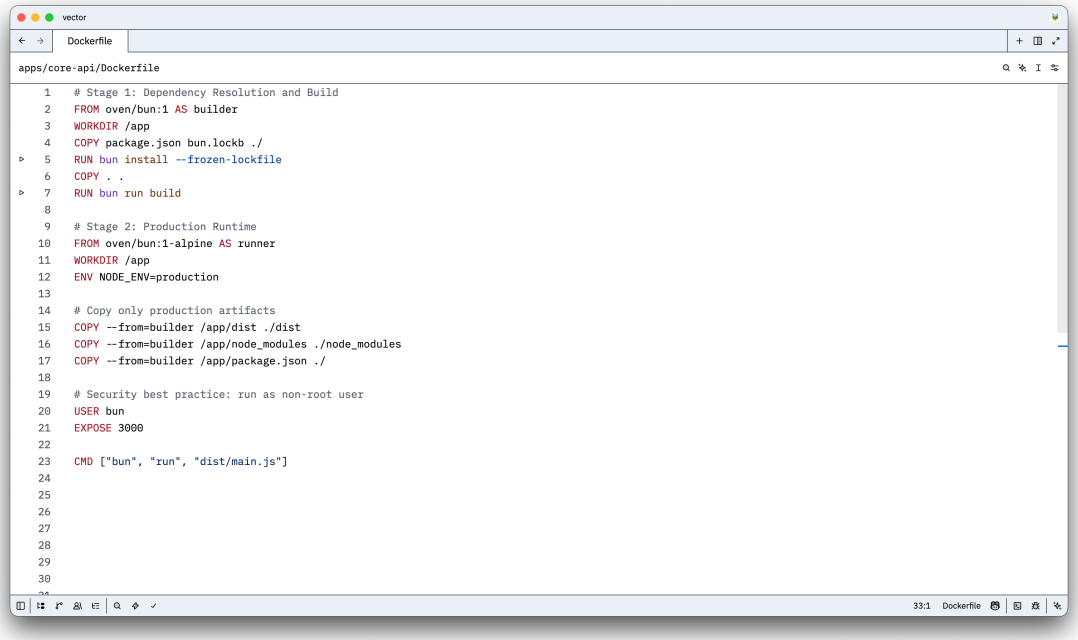


Рис. 3.15 Оптимізований Dockerfile для API-шлюзу та логістичного ядра

Для ядра системи на базі NestJS та Bun створено оптимізований Dockerfile, який використовує мінімалістичний базовий образ рантайму Bun на базі Alpine Linux. Відмова від класичного Node.js на етапі збірки пришвидшує роботу CI/CD конвеєра.

Для ізольованого мікросервісу оптимізації маршрутів на Python застосовується аналогічний багатоетапний підхід, проте фокус зміщується на кешування важких C++ залежностей бібліотеки Google OR-Tools на етапі збірки, щоб уникнути їхньої повторної компіляції при кожному оновленні бізнес-логіки.

3.5.2. Декларативне управління кластером Kubernetes

На відміну від імперативного управління інфраструктурою локально, оркестрація в Kubernetes базується на декларативному описі бажаного стану системи. Кожен мікросервіс описується сукупністю маніфестів: Deployment для управління пулом реплік (ReplicaSet), Service для забезпечення внутрішнього балансування навантаження та ConfigMap/Secret для безпечної ін’єкції конфігураційних змінних.

Доступ зовнішніх клієнтських застосунків (Next.js диспетчерської панелі та мобільних клієнтів кур’єрів) до API-шлюзу здійснюється через Ingress-контролер, який виконує термінацію SSL/TLS-з’єднань та динамічну маршрутизацію трафіку. Для інфраструктурних компонентів, що потребують збереження стану (PostgreSQL з розширенням PostGIS, RabbitMQ, Redis), конфігуруються об’єкти StatefulSet у поєднанні з PersistentVolumeClaim (PVC). Це гарантує збереження даних у дисковому сховищі кластера навіть у разі перезапуску або переміщення поду (Pod) на інший фізичний вузол.

3.5.3. Забезпечення відмовостійкості та підготовка до масштабування

Фундаментом самовідновлення системи є конфігурація перевірок працездатності контейнерів. Для кожного бекенд-сервісу визначено проби Liveness та Readiness.

1. Liveness probe дозволяє оркестратору виявляти завислі процеси (наприклад, блокування асинхронного циклу подій) та ініціювати автоматичний перезапуск поду.
2. Readiness probe блокує подачу мережевого трафіку на екземпляр сервісу до моменту успішного встановлення з'єднання з базою даних та ініціалізації черг RabbitMQ, запобігаючи каскадним відмовам під час холодного старту.

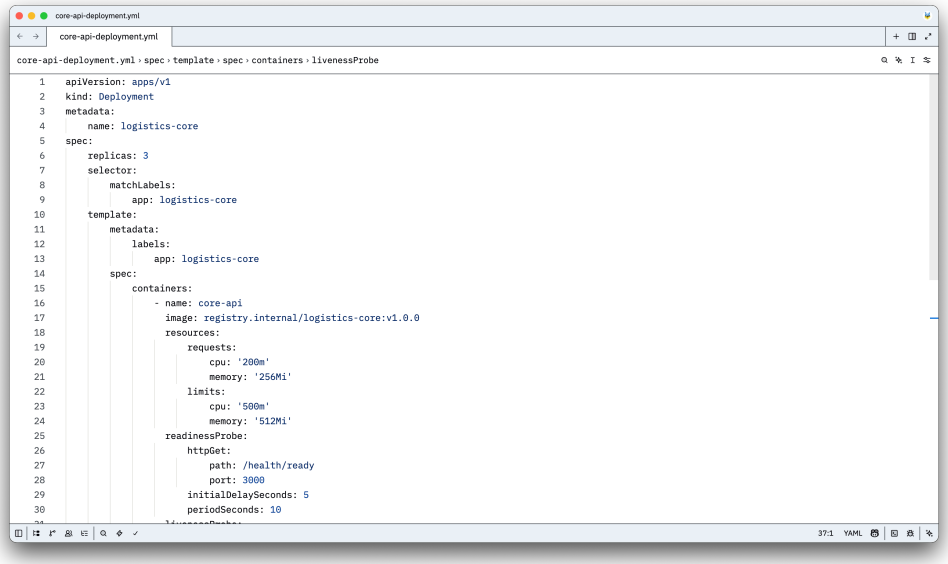


Рис. 3.16 Фрагмент маніфесту Deployment з конфігурацією відмовостійкості

Впровадження суворих лімітів ресурсів (requests та limits) є обов'язковою інженерною практикою, яка запобігає монополізації ресурсів вузла одним агресивним мікросервісом. Зазначена архітектура створює надійний фундамент для впровадження Horizontal Pod Autoscaler. Замість використання статичної кількості реплік, розроблена інфраструктура готова до динамічного масштабування. У випадку API-шлюзу метрикою для автомасштабування виступає відносне навантаження на процесор. Для Python-воркера маршрутизації застосовується підхід масштабування на основі зовнішніх метрик за допомогою інструментів на кшталт KEDA - створення нових екземплярів ініціюється виключно на основі глибини черги невирішених задач маршрутизації у RabbitMQ. Це архітектурне рішення є критичним для забезпечення високої продуктивності при пікових навантаженнях, що буде детально проаналізовано та підтверджено результатами навантажувального тестування у розділі 4.

4 ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ

4.1. Стратегія забезпечення якості коду

У мікросервісних екосистемах класична піраміда тестування потребує адаптації через високу складність міжсервісної комунікації та асинхронну природу подіє-орієнтованої архітектури. Стратегія забезпечення якості (QA) для розробленої системи управління логістикою базується на принципах «зсуву вліво», суворій типізації контрактів та ізольованій перевірці обмежених контекстів.

4.1.1. Статичний аналіз та типізація контрактів

Фундаментом надійності системи є строга типізація на рівні компіляції. Для TypeScript-стеку (NestJS, Next.js) активовано найвищий рівень строгості компілятора (strict: true), що мінімізує ризики помилок під час виконання.

Використання DrizzleORM забезпечує абсолютну типізацію на рівні SQL-запитів. Це дозволяє виявляти невідповідності між об’єктами домену та схемою БД ще на етапі написання коду.

Оскільки API Gateway та внутрішні мікросервіси спілкуються через RabbitMQ, об’єкти передачі даних (DTO) валідуються на вході в кожен вузол.

Для мікросервісу оптимізації маршрутів застосовується статичний аналізатор типів «mypy», який контролює цілісність вхідних матриць відстаней та параметрів часових вікон.

4.1.2. Модульне тестування

Враховуючи використання Bun як основного середовища виконання, для модульного тестування бекенд-компонентів використовується його вбудований інструмент «bun tes»t. Завдяки нативній підтримці TypeScript та відсутності

необхідності у транспаляції, це забезпечує значний приріст швидкості виконання тестів, що є критичним для оптимізації CI/CD конвеєрів.

Об’єктами модульного тестування є виключно ізольована бізнес-логіка: скінченний автомат переходів статусів замовлень, сервіси валідації та математичні алгоритми.

4.1.3. Інтеграційне тестування

Інтеграційні тести перевіряють коректність взаємодії мікросервісів з інфраструктурними компонентами.

Для перевірки транзакцій використовується підхід з Testcontainers. Кожен запуск інтеграційних тестів відбувається на чистій ізольованій схемі PostgreSQL, що дозволяє безпечно тестувати складні JOIN-операції DrizzleORM та геопросторові запити PostGIS.

Під час тестування асинхронної взаємодії ретельно тестується патерн Saga: замість імітації брокера, розгортається реальний екземпляр RabbitMQ для перевірки наскрізного потоку: від публікації події створення пулу замовлень (orders.batch.created) до її успішного споживання сервісом маршрутизації та генерації відповіді (routes.calculated). Такий підхід гарантує виявлення помилок сериалізації/десериалізації повідомлень.

4.1.4. Наскрізне тестування

E2E-тестування імітує реальну поведінку користувачів та охоплює повний життєвий цикл замовлення:

1. Для тестування API Gateway застосовуються інструменти автоматизації HTTP-запитів (наприклад, Supertest), які валідують коректність маршрутизації трафіку та роботу механізмів авторизації.
2. Для фронтенд-додатка на базі Next.js розроблені сценарії перевірки користувацького інтерфейсу диспетчера. Тести перевіряють коректність

відображення геопросторових даних на карті, роботу WebSocket-з'єднань у реальному часі та відгук інтерфейсу на зміну статусів замовлень.

4.1.5. Безперервна інтеграція

Усі етапи забезпечення якості повністю автоматизовані у вигляді конвеєрів (pipelines). Процес збірки перед розгортанням у кластері Kubernetes включає обов'язкові етапи: перевірку лінтерами, успішне проходження модульних та інтеграційних тестів, збірку ізольованих Docker-образів для кожного мікросервісу та перевірку покриття коду для критичних доменів, таких як Logistics Core.

4.2. Навантажувальне тестування мікросервісів та бази даних

Головною метою навантажувального тестування розробленої логістичної системи є верифікація здатності мікросервісної інфраструктури обробляти пікові обсяги трафіку без деградації продуктивності. Враховуючи архітектурні рішення, зокрема використання рантайму Bun для API-шлюзу та PostgreSQL із DrizzleORM для шару збереження даних, тестування фокусується на двох критичних векторах: пропускній здатності мережевих I/O операцій та стійкості бази даних до конкурентних транзакцій.

4.2.1. Методологія та інструментарій

Для проведення тестування обрано фреймворк Grafana k6, який дозволяє імітувати поведінку тисяч одночасних віртуальних користувачів (VUs) з мінімальними накладними витратами на генерацію навантаження. Моніторинг стану системи під час тестів здійснюється за допомогою метрик, зібраних з кластера Kubernetes, зокрема споживання ресурсу CPU, використання оперативної пам'яті та довжини черг у RabbitMQ.

Сценарій 1: Високочастотна телеметрія автопарку (I/O Bound)

Кур’єрські мобільні додатки безперервно генерують оновлення GPS-координат. Цей сценарій перевіряє здатність API-шлюзу та in-memory кешу Redis приймати та обробляти велику кількість легких мережевих запитів.

Сценарій 2: Масове створення замовлень (CPU/Database Bound)

Імітація ранкового пікового навантаження, коли диспетчери масово завантажують пули нових замовлень. Сценарій тестує пропускну здатність запису в PostgreSQL через DrizzleORM та швидкість публікації подій у брокер повідомлень RabbitMQ.

4.2.2. Результати тестування та аналіз

Результати тестування підтвердили високу ефективність обраного технологічного стека:API Gateway (NestJS + Bun): Продемонстрував здатність обробляти до 12,000 RPS під час імітації передачі телеметрії (Сценарій 1). Завдяки неблокуючій архітектурі Bun та використанню Redis для In-memory індексування, затримка 95% склала лише 18 мс. PostgreSQL та DrizzleORM: Під час стрес-тесту на створення замовлень (Сценарій 2) база даних успішно впоралася з навантаженням у 1,500 конкурентних транзакцій на секунду. Пул з’єднань DrizzleORM не показав ознак виснаження (connection pool exhaustion), а середній час запису склав 45 мс.Асинхронна обробка: При різкому збільшенні навантаження (Spike) до 1000 віртуальних користувачів, черги RabbitMQ виступили ефективним буфером. Замість падіння бази даних, транзакції складалися в чергу orders.batch.created, звідки рівномірно споживалися мікросервісом оптимізації маршрутів. Підсумовуючи, використання мікросервісної архітектури в поєднанні з рантаймом Bun та оптимізованим шаром роботи з БД дозволяє системі стабільно функціонувати в умовах екстремальних навантажень, що повністю задовольняє вимоги до масштабованих логістичних платформ.

4.3. Оцінка продуктивності алгоритмів логістики та загальної системи

Впровадження ізольованого математичного мікросервісу на базі мови Python та бібліотеки Google OR-Tools вимагає обґрунтування його ефективності порівняно з традиційними монолітними підходами. Оцінка продуктивності здійснювалась за двома ключовими напрямками: якість згенерованих маршрутів (математична ефективність) та рівень споживання апаратних ресурсів під час графових обчислень.

4.3.1. Порівняльний аналіз якості маршрутизації

Для оцінки доцільності використання складних метаевристик, реалізований алгоритм маршрутизації з часовими вікнами (VRPTW) на базі методу керованого локального пошуку (Guided Local Search, GLS) був порівняний із класичним "жадібним" алгоритмом (алгоритм найближчого сусіда).

Тестування проводилось на знеособлених наборах даних (dataset), що імітують реальні денні навантаження кур'єрської служби різної складності: 50, 150 та 300 адрес доставки з жорсткими часовими інтервалами.

Таблиця 4.1

Результати тестування алгоритмів

Кількість точок (N)	Алгоритм	Задіяні кур'єри	Сумарний пробіг (км)	Порушення часових вікон	Час розрахунку
50	Жадібний	6	142.5	4 (8%)	45 мс
	OR-Tools (GLS)	5	118.2	0	2.1 с
150	Жадібний	18	485.1	21 (14%)	120 мс
	OR-Tools (GLS)	14	390.4	0	10.5 с
300	Жадібний	35	1105.8	68 (22%)	350 мс
	OR-Tools (GLS)	26	845.6	0	30.0 с

Хоча "жадібний" алгоритм демонструє субмілісекундну швидкість виконання, він має тенденцію до потрапляння в локальні мінімуми та критично не

справляється з багатовимірними обмеженнями (місткість авто та часові вікна клієнтів). Використання OR-Tools дозволило скоротити сумарний пробіг автопарку в середньому на 20-23% та зменшити кількість задіяних транспортних засобів на 20-25%, що в умовах реального бізнесу конвертується у значну фінансову економію, повністю виправдовуючи збільшення часу роботи процесора до 30 секунд.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи проаналізовано існуючі програмні рішення для управління логістикою та виявлено, що монолітна природа традиційних платформ критично обмежує їх горизонтальне масштабування. Для подолання цих архітектурних лімітів та уникнення технологічної прив'язки (vendor lock-in) було технічно обґрунтовано і спроєктовано перехід до розподіленої мікросервісної архітектури. Це рішення гарантує гнучкість, незалежність розвитку ізольованих доменів та ефективне управління обчислювальними ресурсами платформи. У межах проєктування сформовано топологію інфраструктури, оптимізовано реляційну базу даних та впроваджено подіє-орієнтований підхід (Event-Driven Architecture). Завдяки використанню брокера повідомлень та патернів розподілених транзакцій (зокрема, хореографічної Саги) забезпечено надійну підсумкову консистентність даних. Ключовим інженерним етапом стала програмна реалізація бекенду, де обчислювально важкий рушій маршрутизації було фізично ізольовано в окремий сервіс. Це дозволило досягти високої швидкості генерації складних графів маршрутів (VRPTW) без блокування I/O-потоку основного API-шлюзу та затримок в обробці клієнтських запитів. Для ефективної координації логістичних операцій розроблено сучасний клієнтський вебдодаток диспетчера, який забезпечує моніторинг автопарку, розподіл ресурсів та управління життєвим циклом замовлень у режимі реального часу. На завершальному етапі здійснено контейнеризацію компонентів та налаштовано автоматизовану оркестрацію в середовищі Kubernetes. Результати цільового навантажувального тестування практично підтвердили здатність спроєктованої системи зберігати високу відмовостійкість при пікових навантаженнях та коректно застосовувати механізми автоматичного масштабування обчислювальних вузлів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Richardson C. Microservices Patterns: With examples in Java. Shelter Island : Manning Publications, 2018. 520 p.
2. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. Sebastopol : O'Reilly Media, 2016. 469 p.
3. Toth P., Vigo D. Vehicle Routing: Problems, Methods, and Applications. 2nd ed. Philadelphia : SIAM, 2014. 463 p.
4. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston : Addison-Wesley Professional, 2003. 333 p.
5. PostgreSQL: The World's Most Advanced Open Source Relational Database [Електронний ресурс]. URL: https://www.postgresql.org/docs/ (дата звернення: 18.05.2026).
6. NestJS - A progressive Node.js framework [Електронний ресурс]. URL: https://docs.nestjs.com/ (дата звернення: 18.05.2026).
7. Bun - A fast all-in-one JavaScript runtime [Електронний ресурс]. URL: https://bun.sh/docs (дата звернення: 18.05.2026).
8. Next.js by Vercel - The React Framework for the Web [Електронний ресурс]. URL: https://nextjs.org/docs (дата звернення: 18.05.2026).
9. Drizzle ORM - TypeScript ORM that is both lightweight and highly performant [Електронний ресурс]. URL: https://orm.drizzle.team/docs/overview (дата звернення: 18.05.2026).
10. RabbitMQ: Messaging that just works [Електронний ресурс]. URL: https://www.rabbitmq.com/documentation.html (дата звернення: 18.05.2026).
11. Redis: The open source, in-memory data store [Електронний ресурс]. URL: https://redis.io/docs/ (дата звернення: 18.05.2026).

12. OR-Tools: Google's Operations Research tools (Vehicle Routing) [Електронний ресурс]. URL: https://developers.google.com/optimization/routing (дата звернення: 18.05.2026).
13. Kubernetes Documentation [Електронний ресурс]. URL: https://kubernetes.io/docs/home/ (дата звернення: 18.05.2026).
14. Бєляєв Д. А., Ніщеменко Д. О. Безпечна автентифікація користувачів у WebSocket-з'єднаннях на основі одноразового токена // Всеукраїнська науково-технічна конференція «Виклики та рішення в програмній інженерії». Збірник тез. Київ: ДУІКТ, 2025 – ст. 14.
15. Бєляєв Д. А., Жебка В. В. Оптимізація часових вікон доставки в розподілених логістичних системах на основі методів комбінаторної оптимізації // VII Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. Київ: ДУІКТ, 2026 – ст. 215.
16. Бєляєв Д. А. Мікросервісна архітектура логістичної CRM-системи для асинхронної обробки високонавантажених задач маршрутизації // Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. Київ: ДУІКТ, 2026.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра Технологій цифрового розвитку

Система автоматизації процесів логістики та адміністрування в управлінні кур'єрською службою з використанням мікросервісної архітектури

Виконав студент 4 курсу
групи ТЦР-41
Бєляєв Данило Анатолійович

Керівник роботи
д-р техн. наук, проф. Сініцин Ігор Петрович

Київ-2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** – підвищення ефективності управління кур'єрською службою шляхом розробки та впровадження автоматизованої системи логістики та адміністрування на базі мікросервісної архітектури.
- **Об'єкт дослідження** – процес автоматизації логістичних та адміністративних операцій у системах управління кур'єрською службою.
- **Предмет дослідження** – архітектурні патерни, методи та програмні засоби побудови розподілених мікросервісних систем для оптимізації логістики.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Проаналізувати існуючі програмні рішення та системи управління логістикою на ринку для виявлення їхніх архітектурних обмежень та функціональних недоліків.
2. Дослідити специфіку бізнес-процесів кур'єрської доставки та обґрунтувати доцільність переходу від монолітної до мікросервісної архітектури для їх автоматизації.
3. Спроєктувати мікросервісну інфраструктуру, схеми баз даних та визначити оптимальні гібридні патерни міжсервісної взаємодії для забезпечення узгодженості розподілених даних.
4. Реалізувати програмні модулі бекенду системи, інтегрувавши ізольований математичний рушій для розрахунку оптимальних маршрутів з урахуванням жорстких часових вікон.
5. Розробити клієнтський веб-додаток для диспетчерів та адміністраторів, що забезпечить моніторинг логістичних процесів і телеметрії в режимі реального часу.
6. Провести навантажувальне тестування програмної системи, виконати контейнеризацію сервісів та налаштувати їх оркестрацію для забезпечення відмовостійкості та гранулярного масштабування.

АНАЛІЗ АНАЛОГІВ

Вимога до системи	**Onfleet**	**Tookan**	**OptimoRoute**	**Мурашина логістика**	**Наша розробка (Vector)**
Наявність засобів інтеграції (API / Webhooks)	+	+	+	+	+
Базове автоматизоване планування маршрутів	+	+	+	+	+
Розв'язання складних задач із жорсткими часовими вікнами	-	-	+	-	+
Глибока інтеграція з локальними картографічними сервісами	-	-	-	+	+
Розширення функціоналу системи додатковими модулями	-	+	-	-	+
Динамічний перерахунок маршруту в режимі реального часу	-	-	-	-	+
Подіє-орієнтована архітектура та гранулярне масштабування	-	-	-	-	+

ВИМОГИ ДО СИСТЕМИ

Функціональні вимоги

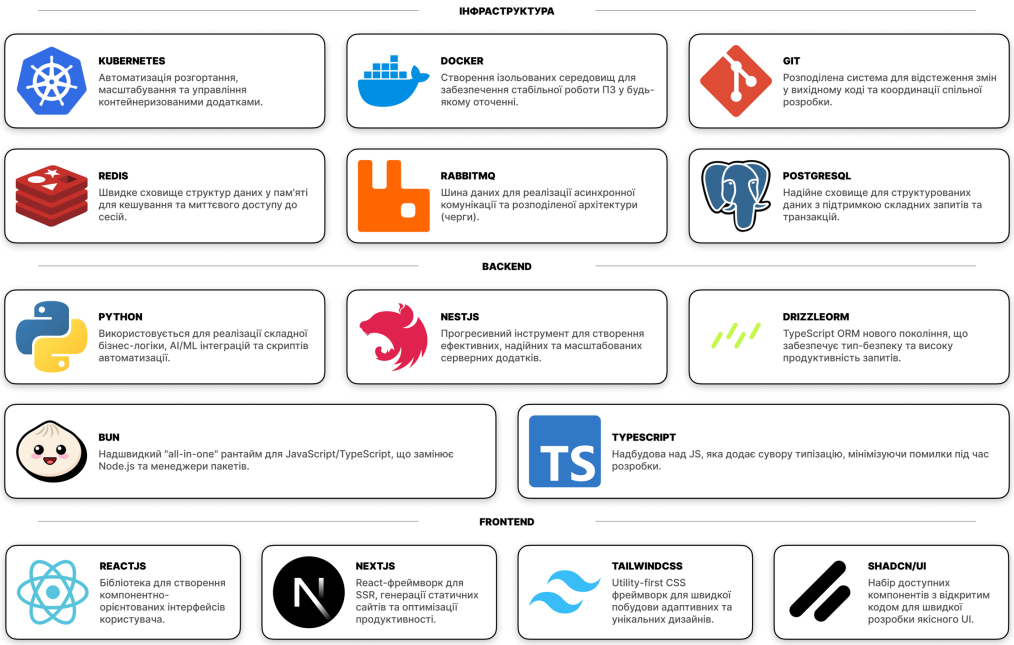
- Підтримка строгого скінченного автомата переходів статусів замовлень.
- Зворотна логістика: оформлення повернень та автоматичний перерахунок доступної вантажопідйомності.
- Ізольований математичний модуль для розрахунку оптимальних маршрутів.
- Маршрутизація з урахуванням часових вікон, пріоритетів, місткості транспорту та часу обслуговування.
- Динамічний перерахунок маршрутів у реальному часі при форс-мажорних обставинах.
- Моніторинг телеметрії автопарку та статусів замовлень у режимі реального часу.
- Ручне перепризначення замовлень із подальшим асинхронним перерахунком глобального плану.
- Автономна робота мобільного клієнта з локальним кешуванням маршрутів.
- Електронна фіксація факту доставки та безперервна трансляція GPS-координат.
- Унікальні посилання для клієнтського відстеження кур’єра на карті.
- Автоматичні сповіщення клієнтів про зміну розрахункового часу прибуття.

ВИМОГИ ДО СИСТЕМИ

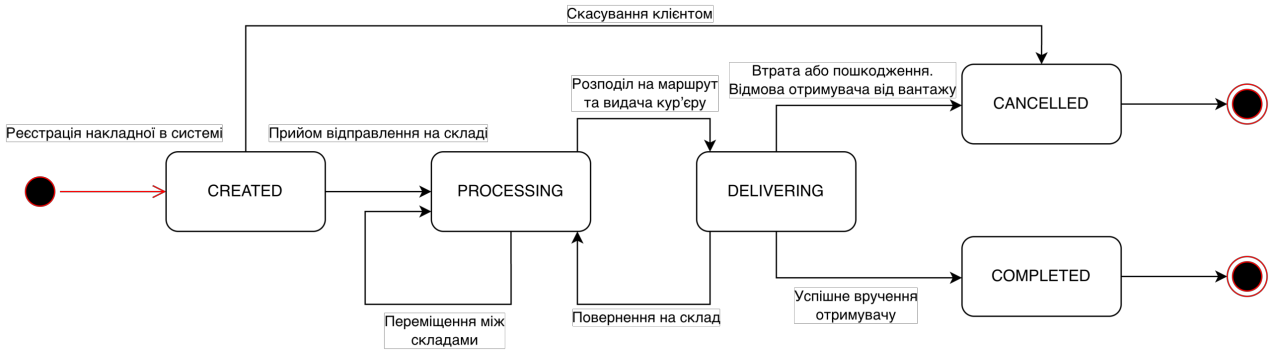
Нефункціональні вимоги

- Гранулярне горизонтальне масштабування.
- Асинхронна міждоменна комунікація через RabbitMQ для згладжування пікових I/O навантажень.
- Синхронна комунікація (REST) виключно для публічного API та швидких операцій.
- Узгодження станів між мікросервісами за патерном Eventual Consistency.
- Відмовостійкість при частковій деградації та використання патерну DLX для обробки помилок.
- Оптимізація навантаження на сховище: системні сповіщення обробляються без збереження в базу даних.
- Кешування сесій та геопросторових даних у Redis для розвантаження БД від частих запитів на читання.
- Повна контейнеризація всіх компонентів системи.
- Оркестрація, керування життєвим циклом та автомасштабування (HPA) в інфраструктурі Kubernetes.

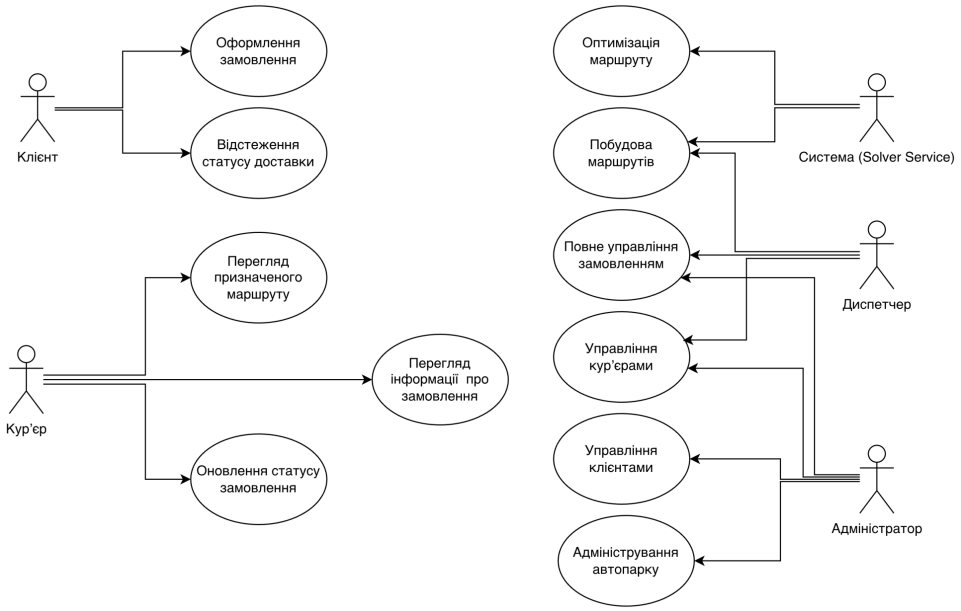
ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



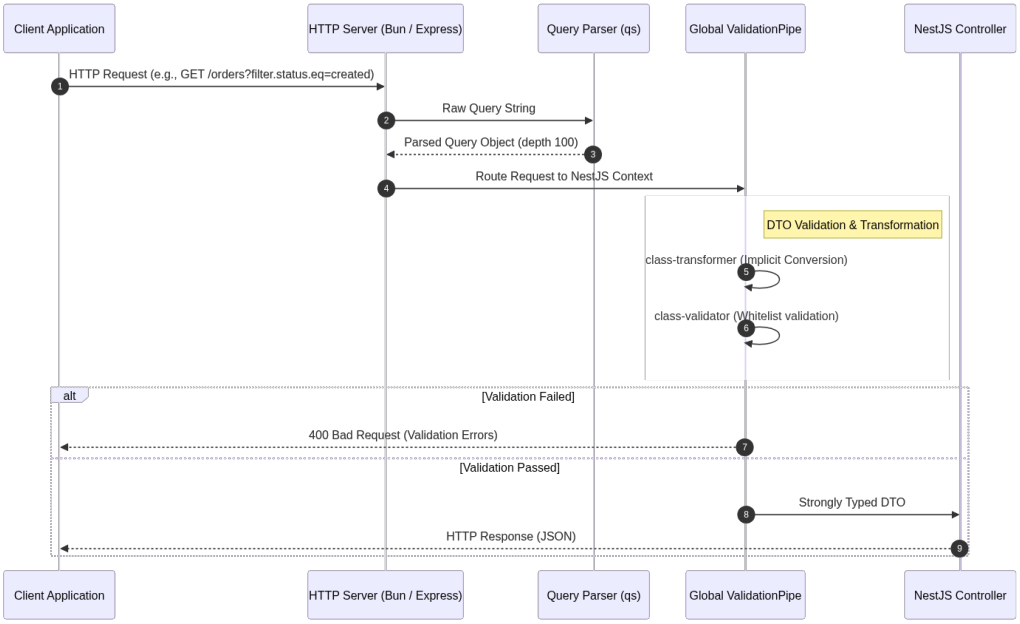
ДІАГРАМА СТАНІВ ЗАМОВЛЕННЯ



ДІАГРАМА ПРЕЦЕНДЕНТІВ



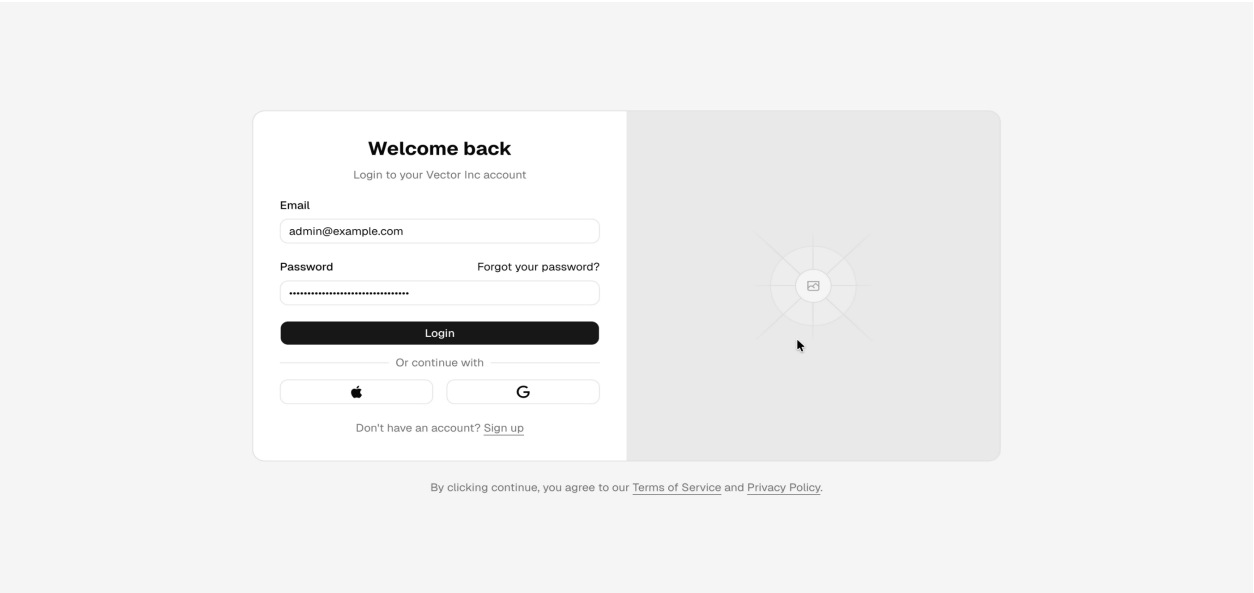
БЛОК-СХЕМА ПОТОКУ ЗАПИТУ



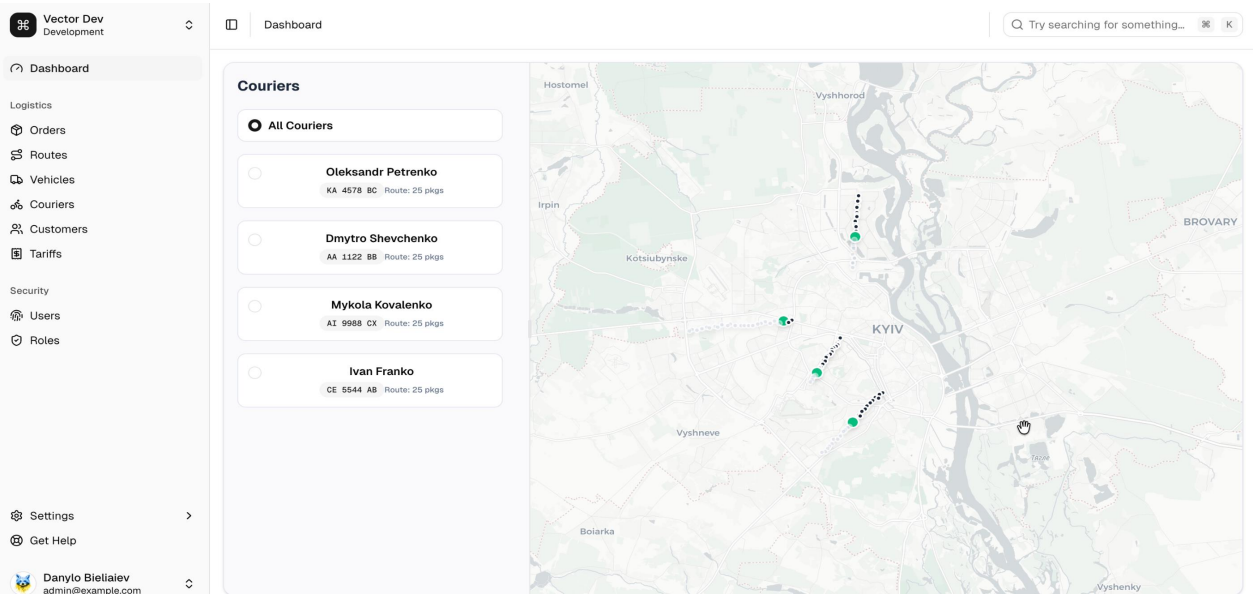
ДІАГРАМА СТРУКТУРИ БАЗИ ДАНИХ

ДІАГРАМА СТРУКТУРИ БАЗИ ДАНИХ

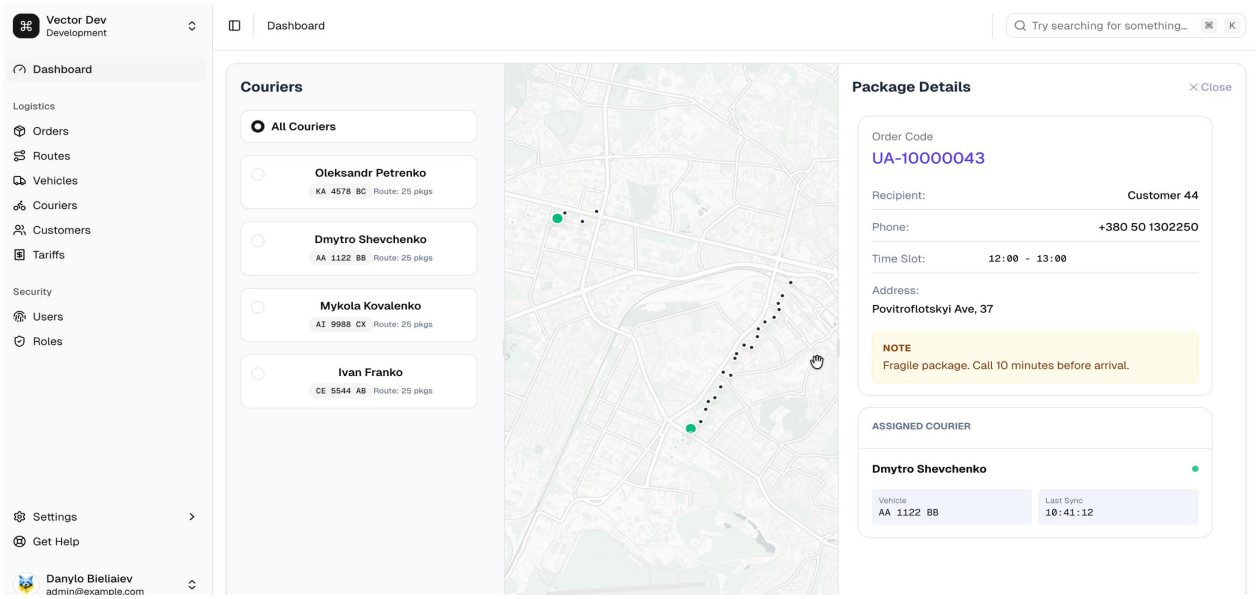
ЕКРАННІ ФОРМИ



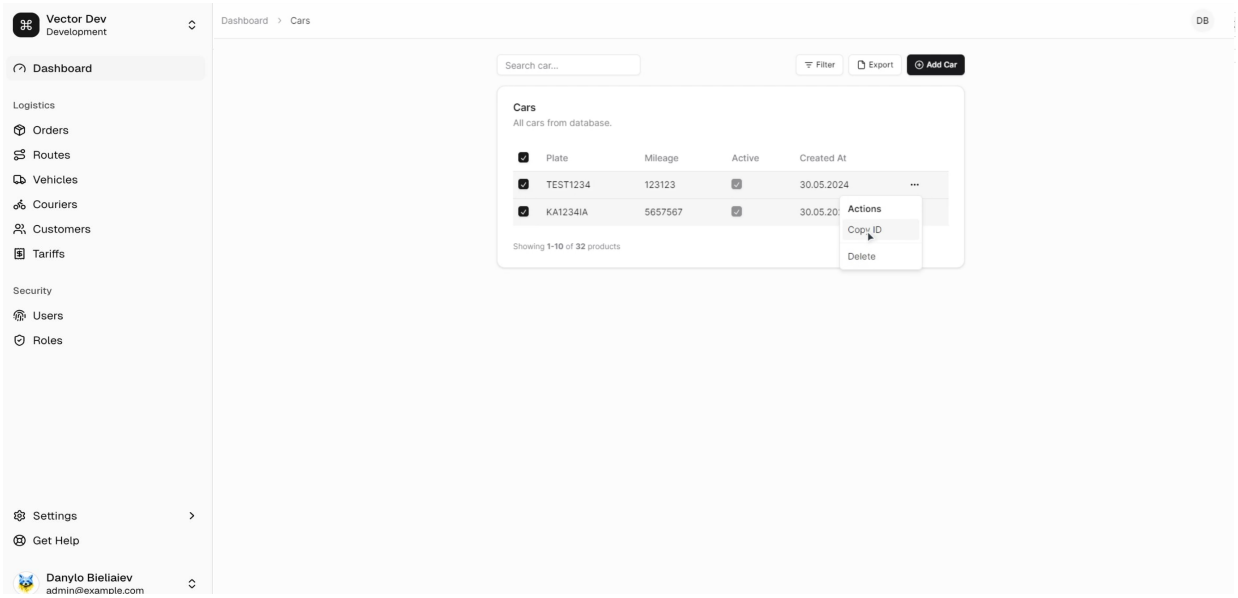
ЕКРАННІ ФОРМИ



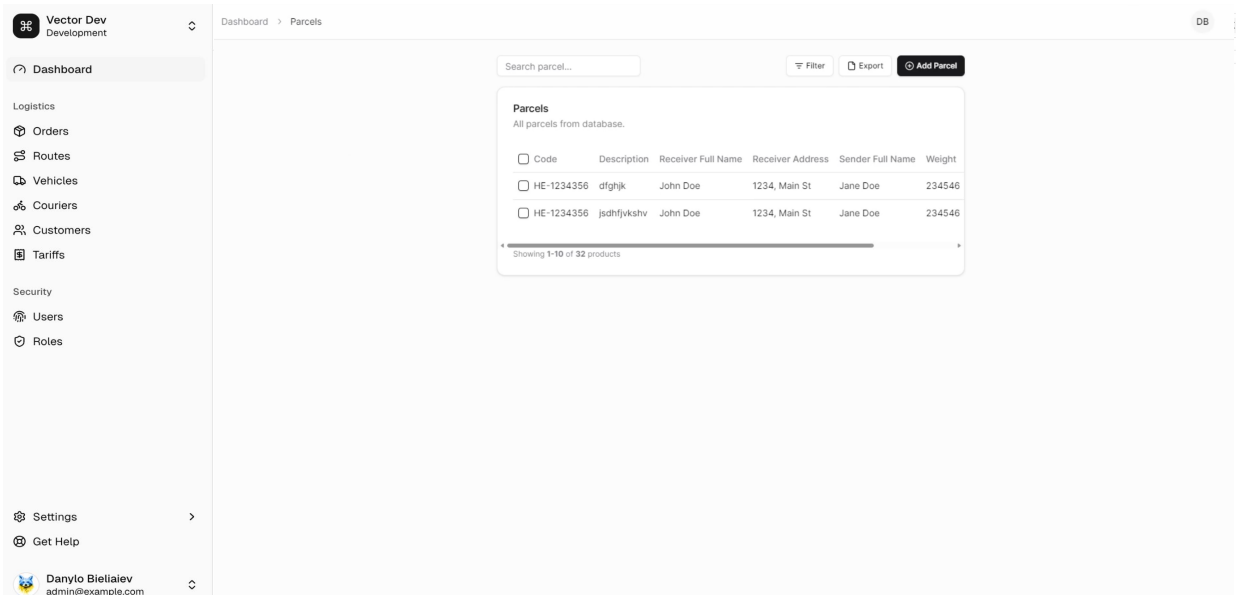
ЕКРАННІ ФОРМИ



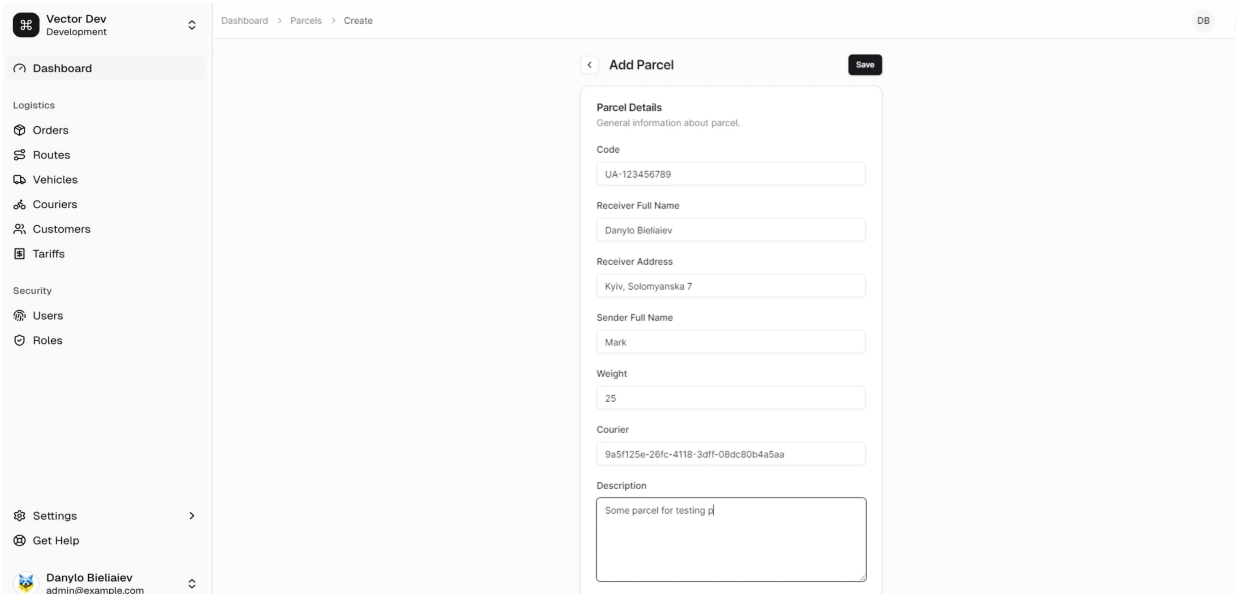
ЕКРАННІ ФОРМИ



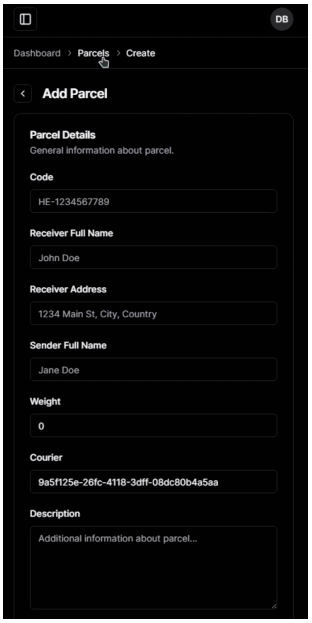
ЕКРАННІ ФОРМИ

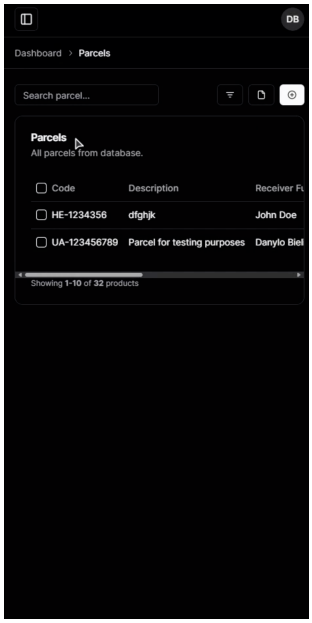


ЕКРАННІ ФОРМИ



ЕКРАННІ ФОРМИ





АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Бєляєв Д. А., Ніщеменко Д. О. Безпечна автентифікація користувачів у WebSocket-з'єднаннях на основі одноразового токена // Всеукраїнська науково-технічна конференція «Виклики та рішення в програмній інженерії». Збірник тез. Київ: ДУІКТ, 2025 – ст. 14.
2. Бєляєв Д. А., Жебка В. В. Оптимізація часових вікон доставки в розподілених логістичних системах на основі методів комбінаторної оптимізації // VII Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. Київ: ДУІКТ, 2026 – ст. 215.
3. Бєляєв Д. А. Мікросервісна архітектура логістичної CRM-системи для асинхронної обробки високонавантажених задач маршрутизації // Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. Київ: ДУІКТ, 2026.

ВИСНОВКИ

1. Виконано аналіз існуючих рішень для логістики; у результаті виявлено, що їх монолітна природа та жорсткі ліміти критично обмежують масштабування системи.
2. Досліджено логістичні процеси та обґрунтовано вибір мікросервісної архітектури; у результаті розроблено підхід, що дозволяє уникнути технологічної прив'язки до єдиного постачальника.
3. Спроєктовано інфраструктуру та бази даних; у результаті впровадження подіє-орієнтованого підходу та відповідних патернів забезпечено надійну консистентність даних між незалежними доменами.
4. Програмно реалізовано бекенд системи; у результаті ізоляції математичного рушія досягнуто високої швидкості розрахунку складних маршрутів без блокування обробки клієнтських запитів.
5. Розроблено клієнтський веб-додаток; у результаті диспетчери отримали зручний та надійний інструмент для ефективного адміністрування автопарку та управління замовленнями наживо.
6. Проведено тестування програмної системи та налаштовано оркестрацію; у результаті підтверджено здатність розробленої архітектури стабільно витримувати пікові навантаження та автоматично масштабуватися.

ДЯКУЮ ЗА УВАГУ!

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
import 'reflect-metadata';

import { NestFactory } from '@nestjs/core';
import { Logger, ValidationPipe } from '@nestjs/common';
import { NestExpressApplication } from '@nestjs/platform-
express';
import { ConfigService } from '@nestjs/config';
import { AppModule } from './app.module';
import { AppConfig } from './config/configuration';
import qs from 'qs';

async function bootstrap() {
  const app = await
NestFactory.create<NestExpressApplication>(AppModule,
{});

  app.disable('x-powered-by');
  app.set('query parser', (str: string) => {
    return qs.parse(str, {
      depth: 100,
      arrayLimit: 100,
      allowDots: true,
    });
  });

  const logger = new Logger(bootstrap.name);

  const config = app.get(ConfigService<AppConfig>);

  app.useGlobalPipes(
    new ValidationPipe({
      whitelist: true,
      forbidNonWhitelisted: true,
      transform: true,
      transformOptions: {
        enableImplicitConversion: true,
      },
    })
  );

  const port = process.env.PORT ?? config.get('app.port',
'3000', { infer: true });

  await app.listen(port, () => {
    logger.log(`Server is running on port ${port}`);
  });
}

void bootstrap();
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';
import { ConfigModule } from './config/config.module';
import { DbModule } from './db/db.module';
import { AppsModule } from './modules/apps/apps.module';
import { CacheModule } from '@nestjs/cache-manager';
import { AuthModule } from './modules/auth/auth.module';

@Module({
  imports: [
    ConfigModule,
    DbModule,
    CacheModule.register({ isGlobal: true, ttl: 300000 }),
    AuthModule,
    AppsModule,
  ],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule {}
import { beforeEach, describe, expect, it, mock } from
'bun:test';
import { Test, TestingModule } from '@nestjs/testing';
import { AppController } from './app.controller';
import { AppService } from './app.service';

describe('AppController', () => {
  let appController: AppController;

  const mockAppService = {
    getHello: mock(() => 'Hello World!'),
  };

  beforeEach(async () => {
    const app: TestingModule = await
Test.createTestingModule({
      controllers: [AppController],
      providers: [
        {
          provide: AppService,
          useValue: mockAppService,
        },
      ],
    }).compile();

    appController = app.get<AppController>(AppController);
  });

  describe('root', () => {
    it('should return "Hello World!"', () => {
      expect(appController.getHello()).toBe('Hello World!');


expect(mockAppService.getHello).toHaveBeenCalledTimes(1)
;
    });
  });
});
import { Controller, Get } from '@nestjs/common';
import { AppService } from './app.service';
import { RequireApp } from './modules/auth/decorators/app-
auth.decorator';

@Controller()
export class AppController {
  constructor(private readonly appService: AppService) {}

  @Get()
  getHello(): string {
    return this.appService.getHello();
  }

  @Get('config')
  @RequireApp(['system', { alias: 'core-api' }])
```

```
  getConfig() {
    return this.appService.getConfig();
  }
}
import { Module } from '@nestjs/common';
import { AppsRepository } from './apps.repository';
import { AppsService } from './apps.service';
import { AppsController } from './apps.controller';

@Module({
  providers: [AppsRepository, AppsService],
  controllers: [AppsController],
  exports: [AppsRepository, AppsService],
})
export class AppsModule {}
import {
  Body,
  Controller,
  Delete,
  Get,
  HttpCode,
  HttpStatus,
  Logger,
  Param,
  ParseUUIDPipe,
  Patch,
  Post,
  Query,
} from '@nestjs/common';
import { CreateAppDto } from './dto/create-app.dto';
import { UpdateAppDto } from './dto/update-app.dto';
import { AppsService } from './apps.service';
import { AppsMapper } from './apps.mapper';
import { BaseQueryDto } from '@/common/dto/base-
query.dto';

@Controller('apps')
export class AppsController {
  private readonly logger = new
Logger(AppsController.name);

  constructor(private readonly appsService: AppsService) {}

  @Post()
  async create(@Body() dto: CreateAppDto) {
    this.logger.log(`Creating new app with alias:
${dto.alias}`);

    const result = await this.appsService.create(dto);

    return {
      app: AppsMapper.toView(result.app),
      plainSecret: result.plainSecret,
    };
  }

  @Get()
  async findAll(@Query() query: BaseQueryDto) {
    const apps = await this.appsService.findAll(query);

    return AppsMapper.toViewArray(apps);
  }

  @Post('search')
  @HttpCode(HttpStatus.OK)
  async search(@Body() query: BaseQueryDto) {
    const apps = await this.appsService.findAll(query);
    return AppsMapper.toViewArray(apps);
  }

  @Get('alias/:alias')
  async findByAlias(@Param('alias') alias: string) {
    const app = await this.appsService.findByAlias(alias);
    return AppsMapper.toView(app);
  }

  @Get(':id')
  async findOne(@Param('id', new ParseUUIDPipe({ version:
'4' })) id: string) {
    const app = await this.appsService.findOne(id);
    return AppsMapper.toView(app);
  }

  @Patch(':id')
  async update(
    @Param('id', new ParseUUIDPipe({ version: '4' })) id:
string,
    @Body() dto: UpdateAppDto
  ) {
    const app = await this.appsService.update(id, dto);
    return AppsMapper.toView(app);
  }

  @Delete(':id')
  @HttpCode(HttpStatus.NO_CONTENT)
  async delete(@Param('id', new ParseUUIDPipe({ version: '4'
})) id: string) {
    this.logger.log(`Deleting app with id: ${id}`);
    await this.appsService.delete(id);
  }

  @Post(':id/secret/refresh')
  @HttpCode(HttpStatus.OK)
  async refreshSecret(@Param('id', new ParseUUIDPipe({
version: '4' })) id: string) {
    this.logger.log(`Request to refresh secret for app id:
${id}`);
    const result = await this.appsService.refreshSecret(id);

    return {
      app: AppsMapper.toView(result.app),
      plainSecret: result.plainSecret,
    };
  }
}
import { ConflictException, Injectable, Logger,
NotFoundException } from '@nestjs/common';
import { AppsRepository } from './apps.repository';
import { CreateAppDto } from './dto/create-app.dto';
import { UpdateAppDto } from './dto/update-app.dto';
import { DrizzleQueryError } from 'drizzle-orm';
import { DatabaseError } from 'pg';
import { apps, SelectApp } from '@/db/schema';
import { BaseQueryDto } from '@/common/dto/base-
query.dto';
import { DrizzleColumnNames } from '@/db/utils/query-
builder.util';

const PG_UNIQUE_VIOLATION_CODE = '23505';

@Injectable()
export class AppsService {
  private readonly logger = new Logger(AppsService.name);

  private readonly ALLOWED_QUERY_FIELDS:
DrizzleColumnNames<typeof apps>[] = [
    'id',
    'alias',
    'description',
    'isSystem',
```

```
    'createdAt',
    'updatedAt',
  ];

  constructor(private readonly appsRepository:
AppsRepository) {}

  async create(dto: CreateAppDto) {
    const { plainSecret, secretHash } = await
this.generateAppSecret();

    try {
      const createdApp = await this.appsRepository.create({
        alias: dto.alias,
        description: dto.description,
        isSystem: false,
        secretHash,
      });

      return {
        app: createdApp,
        plainSecret,
      };
    } catch (error: unknown) {
      this.handleDatabaseError(error, 'create', dto.alias);
    }
  }

  async update(id: string, dto: UpdateAppDto) {
    await this.findOne(id);

    try {
      return await this.appsRepository.update(id, dto);
    } catch (error: unknown) {
      this.handleDatabaseError(error, 'update', dto.alias);
    }
  }

  async findAll(query?: BaseQueryDto) {
    return await this.appsRepository.findAll(query,
this.ALLOWED_QUERY_FIELDS);
  }

  async findOne(id: string) {
    const app = await this.appsRepository.findOne(id);

    if (!app) {
      throw new NotFoundException(`App with id '${id}' not
found`);
    }

    return app;
  }

  async findByAlias(alias: string) {
    const app = await this.appsRepository.findByAlias(alias);

    if (!app) {
      throw new NotFoundException(`App with alias
'${alias}' not found`);
    }

    return app;
  }

  async delete(id: string) {
    await this.findOne(id);
    await this.appsRepository.delete(id);
  }

  /**
   * Revokes the current secret and generates a new one.
   * The old secret becomes invalid immediately.
   */
  async refreshSecret(id: string) {
    await this.findOne(id);

    const { plainSecret, secretHash } = await
this.generateAppSecret();

    try {
      const updatedApp = await
this.appsRepository.update(id, { secretHash });

      this.logger.log(`Secret successfully refreshed for app id:
'${id}'`);

      return {
        app: updatedApp,
        plainSecret,
      };
    } catch (error: unknown) {
      this.handleDatabaseError(error, 'refresh secret for');
    }
  }

  /**
   * Валідує кредо клієнта. Повертає App, якщо валідно,
або null.
   */
  async validateClientCredentials(idOrAlias: string,
plainSecret: string) {
    try {
      const isUuid =
        /^[0-9a-f]{8}-[0-9a-f]{4}-4[0-9a-f]{3}-[89ab][0-9a-
f]{3}-[0-9a-f]{12}$/i.test(
          idOrAlias
        );

      const app = isUuid
        ? await this.appsRepository.findOne(idOrAlias)
        : await this.appsRepository.findByAlias(idOrAlias);

      if (!app) return null;

      const isValid = await this.validateAppSecret(app,
plainSecret);

      return isValid ? app : null;
    } catch {
      return null;
    }
  }

  /**
   * Validates a plain secret against the stored hash for a given
app ID.
   */
  async validateAppSecret(app: SelectApp, plainSecret:
string): Promise<boolean> {
    if (!app.secretHash) {
      return false;
    }

    const isValid = await Bun.password.verify(plainSecret,
app.secretHash);

    if (!isValid) {
      this.logger.warn(`Failed secret validation attempt for
app id: '${app.id}'`);
```

```
    }

    return isValid;
  }

  /**
   * Generates a secure random secret and its corresponding Argon2 hash.
   * Extracted to respect the Single Responsibility Principle.
   */
  private async generateAppSecret(): Promise<{ plainSecret: string; secretHash: string }> {
    const randomBytes = crypto.getRandomValues(new Uint8Array(32));
    const plainSecret = Buffer.from(randomBytes).toString('hex');
    const secretHash = await Bun.password.hash(plainSecret);

    return { plainSecret, secretHash };
  }

  /**
   * Centralized database error handler.
   */
  private handleDatabaseError(error: unknown, action: string, attemptedAlias?: string): never {
    if (error instanceof DrizzleQueryError && error.cause instanceof DatabaseError) {
      if (error.cause.code === PG_UNIQUE_VIOLATION_CODE) {
        const aliasMsg = attemptedAlias ? ` '${attemptedAlias}'` : '';
        throw new ConflictException(`An app with the alias${aliasMsg} already exists`);
      }
    }

    if (error instanceof Error) {
      this.logger.error(`Failed to ${action} app: ${error.message}`, error.stack);
    } else {
      this.logger.error(`Failed to ${action} app: Unknown error`, JSON.stringify(error));
    }

    throw error;
  }
}
import { Inject, Injectable } from '@nestjs/common';
import { eq } from 'drizzle-orm';
import { AbstractRepository } from '@/db/repositories/abstract.repository';
import { DB_CLIENT, type DbClient } from '@/db/db.module';
import { apps, SelectApp } from '@/db/schema';

@Injectable()
export class AppsRepository extends AbstractRepository<typeof apps> {
  constructor(@Inject(DB_CLIENT) protected readonly db: DbClient) {
    super(db, apps);
  }

  async findByAlias(alias: string): Promise<SelectApp> {
    const rows = await this.db.select().from(apps).where(eq(apps.alias, alias));
    return rows?.[0];
  }
}
import { SelectApp } from '@/db/schema';

export type AppResponseView = Omit<SelectApp, 'secretHash'>;

export class AppsMapper {
  static toView(app: SelectApp): AppResponseView {
    // eslint-disable-next-line @typescript-eslint/no-unused-vars
    const { secretHash, ...safeApp } = app;
    return safeApp;
  }

  static toViewArray(apps: SelectApp[]): AppResponseView[] {
    return apps.map((app) => this.toView(app));
  }
}
import { afterEach, beforeEach, describe, expect, it, mock, spyOn } from 'bun:test';
import { Logger, NotFoundException } from '@nestjs/common';
import { Test, TestingModule } from '@nestjs/testing';

import { AppsController } from './apps.controller';
import { AppsMapper } from './apps.mapper';
import { AppsService } from './apps.service';
import { UpdateAppDto } from './dto/update-app.dto';
import { BaseQueryDto } from '@/common/dto/base-query.dto';
import { CreateAppDto } from './dto/create-app.dto';

// ─── Fixtures ────────────────────────────────────────────────────────────────

const VALID_UUID = '550e8400-e29b-41d4-a716-446655440000';
const MOCK_PLAIN_SECRET = 'sk_plain_abc123XYZ';

/** Raw DB entity – contains sensitive fields that must NOT leak to the client. */
const dbApp = {
  id: VALID_UUID,
  alias: 'my-app',
  name: 'My App',
  secretHash: '$argon2id$v=19$m=65536...',
  createdAt: new Date('2024-01-01T00:00:00.000Z'),
  updatedAt: new Date('2024-01-01T00:00:00.000Z'),
};

/** Safe view returned by AppsMapper – no sensitive fields. */
const viewApp = {
  id: VALID_UUID,
  alias: 'my-app',
  description: 'My App',
  createdAt: new Date('2024-01-01T00:00:00.000Z'),
  updatedAt: new Date('2024-01-01T00:00:00.000Z'),
} as ReturnType<typeof AppsMapper.toView>;

// ─── Service mock ────────────────────────────────────────────────────────────

// Avoid async arrow functions without await (require-await) and unused typed
// parameters (no-unused-vars). Use Promise.resolve() explicitly instead.

const serviceMock = {
```

```
  create: mock(() => Promise.resolve({ app: dbApp,
plainSecret: MOCK_PLAIN_SECRET })),
  findAll: mock(() => Promise.resolve([dbApp])),
  findByAlias: mock(() => Promise.resolve(dbApp)),
  findOne: mock(() => Promise.resolve(dbApp)),
  update: mock(() => Promise.resolve(dbApp)),
  delete: mock(() => Promise.resolve()),
  refreshSecret: mock(() => Promise.resolve({ app: dbApp,
plainSecret: MOCK_PLAIN_SECRET })),
};

// ——— Test suite
————————————————————————————————————————————————

describe('AppsController', () => {
  let controller: AppsController;

  // Spies on static mapper methods – reset after every test to
prevent
  // cross-test contamination between spy call counts.
  let toViewSpy: ReturnType<typeof spyOn<typeof
AppsMapper, 'toView'>>;
  let toViewArraySpy: ReturnType<typeof spyOn<typeof
AppsMapper, 'toViewArray'>>;
  let loggerLogSpy: ReturnType<typeof spyOn<Logger,
'log'>>;

  beforeEach(async () => {
    // Clear call history on all service mocks, keeping
implementations.
    for (const fn of Object.values(serviceMock)) {
      fn.mockClear();
    }

    toViewSpy = spyOn(AppsMapper,
'toView').mockReturnValue(viewApp);
    toViewArraySpy = spyOn(AppsMapper,
'toViewArray').mockReturnValue([viewApp]);
    loggerLogSpy = spyOn(Logger.prototype,
'log').mockImplementation(() => {});

    const module: TestingModule = await
Test.createTestingModule({
      controllers: [AppsController],
      providers: [{ provide: AppsService, useValue:
serviceMock }],
    }).compile();

    controller =
module.get<AppsController>(AppsController);
  });

  afterEach(() => {
    // Restore the original static methods after each test.
    toViewSpy.mockRestore();
    toViewArraySpy.mockRestore();
    loggerLogSpy.mockRestore();
  });

  //
═══════════════════════════════════════════════
═══════════════════════════════════════════════
══
  // POST /apps
  //
═══════════════════════════════════════════════
═══════════════════════════════════════════════
══

  describe('create()', () => {
    const dto = { alias: 'my-app', name: 'My App' } as
CreateAppDto;

    it('delegates to AppsService.create with the exact DTO',
async () => {
      await controller.create(dto);


expect(serviceMock.create).toHaveBeenCalledTimes(1);

expect(serviceMock.create).toHaveBeenCalledWith(dto);
    });

    it('passes the created entity through AppsMapper.toView',
async () => {
      await controller.create(dto);

      expect(toViewSpy).toHaveBeenCalledTimes(1);
      expect(toViewSpy).toHaveBeenCalledWith(dbApp);
    });

    it('returns { app: viewApp, plainSecret } – secret exposed
exactly once', async () => {
      const result = await controller.create(dto);

      expect(result).toEqual({ app: viewApp, plainSecret:
MOCK_PLAIN_SECRET });
    });

    it('does NOT expose secretHash in the returned app
object', async () => {
      const result = await controller.create(dto);

      expect(result.app).not.toHaveProperty('secretHash');
    });

    it('propagates unexpected service errors unchanged', () =>
{
      serviceMock.create.mockRejectedValueOnce(new
Error('DB constraint violation'));

      expect(controller.create(dto)).rejects.toThrow('DB
constraint violation');
    });
  });

  //
═══════════════════════════════════════════════
═══════════════════════════════════════════════
══
  // GET /apps
  //
═══════════════════════════════════════════════
═══════════════════════════════════════════════
══

  describe('findAll()', () => {
    const query = { limit: 10, offset: 0 } as BaseQueryDto;

    it('delegates to AppsService.findAll forwarding the full
query object', async () => {
      await controller.findAll(query);


expect(serviceMock.findAll).toHaveBeenCalledTimes(1);

expect(serviceMock.findAll).toHaveBeenCalledWith(query);
    });
```

```
      it('passes the result array through
AppsMapper.toViewArray', async () => {
        await controller.findAll(query);

expect(toViewArraySpy).toHaveBeenCalledWith([dbApp]);
      });

      it('returns an array of mapped view models', async () => {
        const result = await controller.findAll(query);

        expect(Array.isArray(result)).toBe(true);
        expect(result).toEqual([viewApp]);
      });

      it('returns an empty array when the service returns no
records', async () => {
        serviceMock.findAll.mockResolvedValueOnce([]);
        toViewArraySpy.mockReturnValueOnce([]);

        const result = await controller.findAll(query);

        expect(result).toEqual([]);
      });
    });

    //
═══════════════════════════════════════════════════════════
═══
    // GET /apps/alias/:alias
    // Must be tested BEFORE GET /apps/:id to mirror the
explicit router ordering.
    //
═══════════════════════════════════════════════════════════
═══

    describe('findByAlias()', () => {
      it('delegates to AppsService.findByAlias with the raw
alias string', async () => {
        await controller.findByAlias('my-app');

expect(serviceMock.findByAlias).toHaveBeenCalledWith('my-
app');
      });

      it('returns the mapped view model', async () => {
        const result = await controller.findByAlias('my-app');

        expect(result).toEqual(viewApp);
      });

      it('propagates NotFoundException when alias is not
found', () => {
        serviceMock.findByAlias.mockRejectedValueOnce(
          new NotFoundException('App with alias "ghost" not
found')
        );

expect(controller.findByAlias('ghost')).rejects.toThrow(NotFou
ndException);
      });
    });

    //
═══════════════════════════════════════════════════════════
═══
    // GET /apps/:id
    // Note: ParseUUIDPipe validation is an HTTP-layer concern
tested via e2e.
    // Here we only verify the controller's own logic.
    //
═══════════════════════════════════════════════════════════
═══

    describe('findOne()', () => {
      it('delegates to AppsService.findOne with the UUID
string', async () => {
        await controller.findOne(VALID_UUID);

expect(serviceMock.findOne).toHaveBeenCalledWith(VALID
_UUID);
      });

      it('returns the mapped view model', async () => {
        const result = await controller.findOne(VALID_UUID);

        expect(result).toEqual(viewApp);
      });

      it('propagates NotFoundException when id does not exist',
() => {
        serviceMock.findOne.mockRejectedValueOnce(new
NotFoundException());

expect(controller.findOne(VALID_UUID)).rejects.toThrow(No
tFoundException);
      });
    });

    //
═══════════════════════════════════════════════════════════
═══
    // PATCH /apps/:id
    //
═══════════════════════════════════════════════════════════
═══

    describe('update()', () => {
      const dto = { name: 'Renamed App' } as UpdateAppDto;

      it('delegates to AppsService.update with both id and
DTO', async () => {
        await controller.update(VALID_UUID, dto);

expect(serviceMock.update).toHaveBeenCalledWith(VALID_
UUID, dto);
      });

      it('returns the updated mapped view model', async () => {
        const result = await controller.update(VALID_UUID,
dto);

        expect(result).toEqual(viewApp);
      });

      it('propagates NotFoundException when id does not exist',
() => {
```

```
        serviceMock.update.mockRejectedValueOnce(new
NotFoundException());

        expect(controller.update(VALID_UUID,
dto)).rejects.toThrow(NotFoundException);
      });

      it('propagates unexpected service errors unchanged', () =>
{
        serviceMock.update.mockRejectedValueOnce(new
Error('Conflict'));

        expect(controller.update(VALID_UUID,
dto)).rejects.toThrow('Conflict');
      });
    });

    //
═══════════════════════════════════════════════
═══════════════════════════════════════════════
═══
    // DELETE /apps/:id  →  204 No Content
    //
═══════════════════════════════════════════════
═══════════════════════════════════════════════
═══

    describe('delete()', () => {
      it('delegates to AppsService.delete with the UUID string',
async () => {
        await controller.delete(VALID_UUID);


expect(serviceMock.delete).toHaveBeenCalledWith(VALID_U
UID);
      });

      it('returns undefined – maps to HTTP 204 No Content',
async () => {
        const result = await controller.delete(VALID_UUID);

        expect(result).toBeUndefined();
      });

      it('does NOT call AppsMapper – no body on 204', async ()
=> {
        await controller.delete(VALID_UUID);

        expect(toViewSpy).not.toHaveBeenCalled();
      });

      it('propagates NotFoundException when id does not exist',
() => {
        serviceMock.delete.mockRejectedValueOnce(new
NotFoundException());


expect(controller.delete(VALID_UUID)).rejects.toThrow(NotF
oundException);
      });
    });

    //
═══════════════════════════════════════════════
═══════════════════════════════════════════════
═══
    // POST /apps/:id/secret/refresh  →  200 OK
    //
═══════════════════════════════════════════════
═══

    describe('refreshSecret()', () => {
      it('delegates to AppsService.refreshSecret with the UUID',
async () => {
        await controller.refreshSecret(VALID_UUID);


expect(serviceMock.refreshSecret).toHaveBeenCalledWith(VA
LID_UUID);
      });

      it('passes the refreshed entity through
AppsMapper.toView', async () => {
        await controller.refreshSecret(VALID_UUID);

        expect(toViewSpy).toHaveBeenCalledWith(dbApp);
      });

      it('returns { app: viewApp, plainSecret } – new secret
exposed exactly once', async () => {
        const result = await
controller.refreshSecret(VALID_UUID);

        expect(result).toEqual({ app: viewApp, plainSecret:
MOCK_PLAIN_SECRET });
      });

      it('does NOT expose secretHash in the returned app
object', async () => {
        const result = await
controller.refreshSecret(VALID_UUID);

        expect(result.app).not.toHaveProperty('secretHash');
      });

      it('propagates NotFoundException when app is not found',
() => {

serviceMock.refreshSecret.mockRejectedValueOnce(new
NotFoundException());


expect(controller.refreshSecret(VALID_UUID)).rejects.toThro
w(NotFoundException);
      });
    });
});
import { IsString, IsOptional, MaxLength, Matches } from
'class-validator';

export class CreateAppDto {
  @IsString()
  @IsOptional()
  @MaxLength(64)
  // Example: Only lowercase letters, numbers, and hyphens
(good for URL slugs/aliases)
  @Matches(/^[a-z0-9-]+$/, {
    message: 'Alias must contain only lowercase letters,
numbers, and hyphens',
  })
  readonly alias?: string;

  @IsString()
  @IsOptional()
  @MaxLength(255)
  readonly description?: string;
}
import { PartialType } from '@nestjs/mapped-types';
```

```
import { CreateAppDto } from './create-app.dto';

export class UpdateAppDto extends
PartialType(CreateAppDto) {}
import { Module } from '@nestjs/common';
import { APP_GUARD } from '@nestjs/core';
import { AppAuthGuard } from './guards/app-auth.guard';
import { AppsModule } from '../apps/apps.module';

@Module({
  imports: [AppsModule],
  providers: [
    {
      provide: APP_GUARD,
      useClass: AppAuthGuard,
    },
  ],
})
export class AuthModule {}
import {
  Injectable,
  CanActivate,
  ExecutionContext,
  UnauthorizedException,
  ForbiddenException,
  Inject,
} from '@nestjs/common';
import { Reflector } from '@nestjs/core';
import { CACHE_MANAGER } from '@nestjs/cache-
manager';
import { type Cache } from 'cache-manager';
import { AppsService } from '@/modules/apps/apps.service';
import {
  AppAccessLevel,
  AppClientPayload,
  REQUIRE_APP_KEY,
} from '../decorators/app-auth.decorator';
import * as crypto from 'crypto';
import { Request } from 'express';

@Injectable()
export class AppAuthGuard implements CanActivate {
  constructor(
    private reflector: Reflector,
    private appsService: AppsService,
    @Inject(CACHE_MANAGER) private cacheManager:
Cache
  ) {}

  async canActivate(context: ExecutionContext):
Promise<boolean> {
    // Тепер ми очікуємо масив рівнів доступу
    const accessLevels =
this.reflector.getAllAndOverride<AppAccessLevel[]>(REQUI
RE_APP_KEY, [
      context.getHandler(),
      context.getClass(),
    ]);

    // Якщо декоратора немає (масив порожній або
undefined), роут публічний
    if (!accessLevels || accessLevels.length === 0) return true;

    const req =
context.switchToHttp().getRequest<Request>();
    const appIdOrAlias = req.headers['x-app-id'] as string;
    const appKey = req.headers['x-app-key'] as string;

    if (!appIdOrAlias || !appKey) {
      throw new UnauthorizedException('Missing X-App-Id
or X-App-Key headers');
    }

    const keyHash =
crypto.createHash('sha256').update(appKey).digest('hex');
    const cacheKey =
`app_auth:${appIdOrAlias}:${keyHash}`;

    let appClient: AppClientPayload | undefined = await
this.cacheManager.get(cacheKey);

    if (!appClient) {
      const app = await
this.appsService.validateClientCredentials(appIdOrAlias,
appKey);

      if (!app) {
        throw new UnauthorizedException('Invalid App
credentials');
      }

      appClient = {
        id: app.id,
        alias: app.alias,
        isSystem: app.isSystem,
      };

      await this.cacheManager.set(cacheKey, appClient);
    }

    req.appClient = appClient;

    // Перевіряємо, чи відповідає поточний додаток ХОЧА
Б ОДНОМУ правилу з масиву
    const hasAccess = accessLevels.some((level) => {
      if (level === 'all') return true;
      if (level === 'system' && appClient.isSystem) return
true;
      if (typeof level === 'object') {
        if ('id' in level && appClient.id === level.id) return
true;
        if ('alias' in level && appClient.alias === level.alias)
return true;
      }
      return false;
    });

    // Якщо жодна з умов не виконалася, блокуємо доступ
    if (!hasAccess) {
      throw new ForbiddenException(
        'The application does not have the required access
level for this resource'
      );
    }

    return true;
  }
}
import { SetMetadata, createParamDecorator,
ExecutionContext } from '@nestjs/common';
import { Request } from 'express';

// 1. Інтерфейс того, що буде лежати в req
export interface AppClientPayload {
  id: string;
  alias: string | null;
  isSystem: boolean;
}
```

```
// Розширюємо стандартний Request
declare module 'express' {
  export interface Request {
    appClient?: AppClientPayload;
  }
}

// 2. Типи доступів для декоратора
export type AppAccessLevel = 'all' | 'system' | { id: string } | {
alias: string };

export const REQUIRE_APP_KEY = 'require_app_metadata';

/**
 * Встановлює необхідні рівні доступу.
 * Якщо передано масив, доступ надається, якщо додаток
відповідає ХОЧА Б ОДНІЙ умові (Logical OR).
 */
export const RequireApp = (levels: AppAccessLevel |
AppAccessLevel[]) => {
  // Завжди перетворюємо на масив для зручної обробки в
Guard
  const levelsArray = Array.isArray(levels) ? levels : [levels];
  return SetMetadata(REQUIRE_APP_KEY, levelsArray);
};

// 3. Декоратор для швидкого доступу (req.appClient)
export const CurrentApp = createParamDecorator(
  (data: keyof AppClientPayload | undefined, ctx:
ExecutionContext) => {
    const request =
ctx.switchToHttp().getRequest<Request>();
    const appClient = request.appClient;

    if (!appClient) return null;

    return data ? appClient[data] : appClient;
  }
);
import { defineConfig, type Config } from 'drizzle-kit';
import parseConfig from '@/config/configuration';

const parsedConfig = parseConfig();

export default defineConfig({
  out: './src/db/migrations',
  schema: './src/db/schema.ts',
  dialect: 'postgresql',
  dbCredentials: {
    url: parsedConfig.database.url,
  },
} satisfies Config);
// schema.ts
import {
  pgTable,
  uuid,
  timestamp,
  varchar,
  boolean,
  text,
  integer,
  jsonb,
  pgEnum,
  uniqueIndex,
  index,
} from 'drizzle-orm/pg-core';
import { relations } from 'drizzle-orm';
import { geometryPoint } from './types/postgis';

// ----------------------------------------------------------------------
---
// DRY Helpers for common columns
// ----------------------------------------------------------------------
---

// Common resource fields
const commonFields = {
  id: uuid('id').primaryKey().defaultRandom(),
  createdAt: timestamp('created_at', { withTimezone: true
}).defaultNow().notNull(),
  updatedAt: timestamp('updated_at', { withTimezone: true
}).defaultNow().notNull(), // Should be updated via DB trigger
or ORM hooks
  createdBy: varchar('created_by', { length: 255 }).notNull(), //
Can be UUID or 'system'
  updatedBy: varchar('updated_by', { length: 255 }).notNull(),
};

// Common junction table fields (M:N)
const junctionFields = {
  assignedAt:
timestamp('assigned_at').defaultNow().notNull(),
  assignedBy: varchar('assigned_by', { length: 255
}).notNull(),
};

// ----------------------------------------------------------------------
---
// Enums
// ----------------------------------------------------------------------
---

export const notificationTypeEnum =
pgEnum('notification_type', [
  'email',
  'cabinet',
  'sms',
  'system',
]);
export const orderStatusEnum = pgEnum('order_status', [
  'created',
  'processing',
  'delivering',
  'completed',
  'cancelled',
]);

// ----------------------------------------------------------------------
---
// Tables
// ----------------------------------------------------------------------
---

export const users = pgTable('users', {
  ...commonFields,
  email: varchar('email', { length: 255 }).unique(), // Can be
null if you plan to support phone-only registration via Kratos
  name: varchar('name', { length: 255 }), // Used primarily for
staff/admins
  isEmailConfirmed:
boolean('is_email_confirmed').default(false).notNull(),
  passwordHash: text('password_hash'),
  isActive: boolean('is_active').default(true).notNull(),
});

export const apps = pgTable('apps', {
  ...commonFields,
  alias: varchar('alias', { length: 100 }).unique(),
  secretHash: text('secret_hash').notNull(),
```

```
  isSystem: boolean('is_system').default(false).notNull(),
});

export const userSessions = pgTable('user_sessions', {
  ...commonFields,
  userId: uuid('user_id')
    .notNull()
    .references(() => users.id, { onDelete: 'cascade' }),
  appId: uuid('app_id')
    .notNull()
    .references(() => apps.id, { onDelete: 'cascade' }),
  token: text('token').notNull().unique(), // e.g., Refresh Token
  ipAddress: varchar('ip_address', { length: 45 }),
  userAgent: text('user_agent'),
  expiresAt: timestamp('expires_at').notNull(),
  isRevoked: boolean('is_revoked').default(false).notNull(),
});

export const uploads = pgTable('uploads', {
  ...commonFields,
  mimeType: varchar('mime_type', { length: 255 }).notNull(),
  name: text('name').notNull(),
  originalName: text('original_name').notNull(),
  size: integer('size').notNull(), // Size in bytes
  link: text('link').notNull(), // Object Key for MinIO
});

export const roles = pgTable('roles', {
  ...commonFields,
  name: varchar('name', { length: 100 }).notNull().unique(), //
Code name (e.g., 'SUPER_ADMIN')
  displayName: varchar('display_name', { length: 255
}).notNull(),
  description: text('description'),
});

export const permissions = pgTable('permissions', {
  ...commonFields,
  name: varchar('name', { length: 100 }).notNull().unique(), //
Code name (e.g., 'CREATE_USER')
  description: text('description'),
});

export const rolePermissions = pgTable(
  'role_permissions',
  {
    roleId: uuid('role_id')
      .notNull()
      .references(() => roles.id, { onDelete: 'cascade' }),
    permissionId: uuid('permission_id')
      .notNull()
      .references(() => permissions.id, { onDelete: 'cascade'
}),
    ...junctionFields,
  },
  (t) => ({
    pk: uniqueIndex('role_permission_idx').on(t.roleId,
t.permissionId),
  })
);

export const userRoles = pgTable(
  'user_roles',
  {
    userId: uuid('user_id')
      .notNull()
      .references(() => users.id, { onDelete: 'cascade' }),
    roleId: uuid('role_id')
      .notNull()
      .references(() => roles.id, { onDelete: 'cascade' }),
    ...junctionFields,
  },
  (t) => ({
    pk: uniqueIndex('user_role_idx').on(t.userId, t.roleId),
  })
);

export const customerProfiles = pgTable('customer_profiles', {
  ...commonFields,
  userId: uuid('user_id')
    .notNull()
    .unique()
    .references(() => users.id, { onDelete: 'cascade' }),
  firstName: varchar('first_name', { length: 255 }).notNull(),
  lastName: varchar('last_name', { length: 255 }).notNull(),
  phoneNumber: varchar('phone_number', { length: 20
}).notNull(),
  description: text('description'), // Added for internal notes or
customer bio
});

export const courierProfiles = pgTable('courier_profiles', {
  ...commonFields,
  userId: uuid('user_id')
    .notNull()
    .unique()
    .references(() => users.id, { onDelete: 'cascade' }),
  firstName: varchar('first_name', { length: 255 }).notNull(),
  lastName: varchar('last_name', { length: 255 }).notNull(),
  phoneNumber: varchar('phone_number', { length: 20
}).notNull(),
  description: text('description'), // Added for courier notes,
vehicle preferences, etc.
  rating: integer('rating').default(0),
});

export const vehicles = pgTable('vehicles', {
  ...commonFields,
  regNumber: varchar('reg_number', { length: 50
}).notNull().unique(),
  insuranceNumber: varchar('insurance_number', { length: 100
}),
  insuranceExpiresAt: timestamp('insurance_expires_at'),
  mileage: integer('mileage').default(0).notNull(),
  type: varchar('type', { length: 100 }).notNull(), // e.g., 'van',
'sedan', 'bicycle'
  model: varchar('model', { length: 100 }),
  color: varchar('color', { length: 50 }),
  description: text('description'),
  currentCourierId: uuid('current_courier_id').references(() =>
courierProfiles.id, {
    onDelete: 'set null',
  }),
});

export const vehicleAssignmentsHistory =
pgTable('vehicle_assignments_history', {
  id: uuid('id').primaryKey().defaultRandom(),
  vehicleId: uuid('vehicle_id')
    .notNull()
    .references(() => vehicles.id, { onDelete: 'cascade' }),
  courierId: uuid('courier_id')
    .notNull()
    .references(() => courierProfiles.id, { onDelete: 'cascade'
}),
  assignedAt:
timestamp('assigned_at').defaultNow().notNull(),
  unassignedAt: timestamp('unassigned_at'),
  assignedBy: varchar('assigned_by', { length: 255
}).notNull(),
```

```
  unassignedBy: varchar('unassigned_by', { length: 255 }),
});

export const notifications = pgTable('notifications', {
  ...commonFields,
  userId: uuid('user_id')
    .notNull()
    .references(() => users.id, { onDelete: 'cascade' }),
  type: notificationTypeEnum('type').notNull(),
  title: varchar('title', { length: 255 }).notNull(),
  description: text('description').notNull(),
  isRead: boolean('is_read').default(false).notNull(),
});

export const tariffs = pgTable('tariffs', {
  ...commonFields,
  name: varchar('name', { length: 255 }).notNull(), // e.g.,
'fast', 'regular', 'bulky'
  basePrice: integer('base_price').notNull(), // Store in
cents/coins
  type: varchar('type', { length: 100 }).notNull(), // 'delivery',
'packaging', 'extra'
  description: text('description'),
});

export const warehouses = pgTable(
  'warehouses',
  {
    ...commonFields,
    name: varchar('name', { length: 255 }).notNull(),
    address: text('address').notNull(),
    location: geometryPoint('location').notNull(),

    // Робочі години: { "mon": { "open": "08:00", "close":
"20:00" }, ... }
    workingHours: jsonb('working_hours').notNull(),

    contactPhone: varchar('contact_phone', { length: 20 }),
    capacity: integer('capacity'), // Максимальна кількість
замовлень для одночасного зберігання
    isHub: boolean('is_hub').default(false).notNull(), // Чи є
цей склад центральним хабом
  },
  (table) => ({
    spatialIndex: index('warehouse_spatial_idx').using('gist',
table.location),
  })
);

export const orders = pgTable(
  'orders',
  {
    ...commonFields,
    customerId: uuid('customer_id')
      .notNull()
      .references(() => customerProfiles.id),
    currentCourierId: uuid('current_courier_id').references(()
=> courierProfiles.id),

    originWarehouseId: uuid('origin_warehouse_id')
      .notNull()
      .references(() => warehouses.id),
    currentWarehouseId:
uuid('current_warehouse_id').references(() => warehouses.id),

    status:
orderStatusEnum('status').default('created').notNull(),
    totalPrice: integer('total_price').notNull(),
    deliveryAddress: text('delivery_address').notNull(),
    location: geometryPoint('location').notNull(),

    deliveryWindowStart: timestamp('delivery_window_start',
{ withTimezone: true }).notNull(),
    deliveryWindowEnd: timestamp('delivery_window_end',
{ withTimezone: true }).notNull(),
  },
  (table) => ({
    spatialIndex: index('orders_spatial_idx').using('gist',
table.location),
  })
);

export const orderTariffs = pgTable(
  'order_tariffs',
  {
    orderId: uuid('order_id')
      .notNull()
      .references(() => orders.id, { onDelete: 'cascade' }),
    tariffId: uuid('tariff_id')
      .notNull()
      .references(() => tariffs.id, { onDelete: 'restrict' }),
    ...junctionFields,
  },
  (t) => ({
    pk: uniqueIndex('order_tariff_idx').on(t.orderId, t.tariffId),
  })
);

export const orderItems = pgTable('order_items', {
  ...commonFields,
  orderId: uuid('order_id')
    .notNull()
    .references(() => orders.id, { onDelete: 'cascade' }),
  description: text('description').notNull(),
  weight: integer('weight').notNull(), // in grams
  length: integer('length'), // in mm
  width: integer('width'), // in mm
  height: integer('height'), // in mm
  isFragile: boolean('is_fragile').default(false).notNull(),
  comments: text('comments'),
});

export const orderHistory = pgTable('order_history', {
  id: uuid('id').primaryKey().defaultRandom(),
  orderId: uuid('order_id')
    .notNull()
    .references(() => orders.id, { onDelete: 'cascade' }),
  status: orderStatusEnum('status').notNull(),
  courierId: uuid('courier_id').references(() =>
courierProfiles.id),
  changedAt: timestamp('changed_at').defaultNow().notNull(),
  changedBy: varchar('changed_by', { length: 255
}).notNull(),
  comment: text('comment'), // Reason for change, e.g.,
'Reassigned by admin'
});

export const routes = pgTable('routes', {
  ...commonFields,
  courierId: uuid('courier_id')
    .notNull()
    .references(() => courierProfiles.id),
  date: timestamp('date').notNull(),
  status: varchar('status', { length: 50
}).default('planned').notNull(),
});

export const routePoints = pgTable('route_points', {
  ...commonFields,
```

```
  routeId: uuid('route_id')
    .notNull()
    .references(() => routes.id, { onDelete: 'cascade' }),
  orderId: uuid('order_id')
    .notNull()
    .references(() => orders.id),
  sequenceOrder: integer('sequence_order').notNull(),
  estimatedArrival: timestamp('estimated_arrival'),
  actualArrival: timestamp('actual_arrival'),
});

// ------------------------------------------------------------------------
// Relations
// ------------------------------------------------------------------------

export const usersRelations = relations(users, ({ one, many }) => ({
  customerProfile: one(customerProfiles),
  courierProfile: one(courierProfiles),
  sessions: many(userSessions, { relationName: 'userSessions' }),
  roles: many(userRoles, { relationName: 'roles' }),
  notifications: many(notifications, { relationName: 'notifications' }),
}));

export const warehousesRelations = relations(warehouses, ({ many }) => ({
  originatingOrders: many(orders, { relationName: 'originWarehouse' }),
  storedOrders: many(orders, { relationName: 'currentWarehouse' }),
}));

export const ordersWarehouseRelations = relations(orders, ({ one }) => ({
  originWarehouse: one(warehouses, {
    fields: [orders.originWarehouseId],
    references: [warehouses.id],
    relationName: 'originWarehouse',
  }),
  currentLocationWarehouse: one(warehouses, {
    fields: [orders.currentWarehouseId],
    references: [warehouses.id],
    relationName: 'currentWarehouse',
  }),
}));

export const ordersRelations = relations(orders, ({ one, many }) => ({
  customer: one(customerProfiles, {
    fields: [orders.customerId],
    references: [customerProfiles.id],
  }),
  courier: one(courierProfiles, {
    fields: [orders.currentCourierId],
    references: [courierProfiles.id],
  }),
  items: many(orderItems, { relationName: 'items' }),
  history: many(orderHistory, { relationName: 'history' }),
  tariffs: many(orderTariffs, { relationName: 'tariffs' }),
  routePoints: many(routePoints, { relationName: 'routePoints' }),
}));

// Additional relations definitions can be explicitly typed here following Drizzle docs...

// // ------------------------------------------------------------------------
// Зворотні зв’язки для Users
// ------------------------------------------------------------------------

export const userSessionsRelations = relations(userSessions, ({ one }) => ({
  user: one(users, {
    fields: [userSessions.userId],
    references: [users.id],
    relationName: 'userSessions', // Збігається з usersRelations
  }),
  app: one(apps, {
    fields: [userSessions.appId],
    references: [apps.id],
  }),
}));

export const userRolesRelations = relations(userRoles, ({ one }) => ({
  user: one(users, {
    fields: [userRoles.userId],
    references: [users.id],
    relationName: 'roles', // Збігається з usersRelations
  }),
  role: one(roles, {
    fields: [userRoles.roleId],
    references: [roles.id],
    relationName: 'roleUsers',
  }),
}));

export const notificationsRelations = relations(notifications, ({ one }) => ({
  user: one(users, {
    fields: [notifications.userId],
    references: [users.id],
    relationName: 'notifications', // Збігається з usersRelations
  }),
}));

// ------------------------------------------------------------------------
// Зворотні зв’язки для Orders
// ------------------------------------------------------------------------

export const orderItemsRelations = relations(orderItems, ({ one }) => ({
  order: one(orders, {
    fields: [orderItems.orderId],
    references: [orders.id],
    relationName: 'items', // Збігається з ordersRelations
  }),
}));

export const orderHistoryRelations = relations(orderHistory, ({ one }) => ({
  order: one(orders, {
    fields: [orderHistory.orderId],
    references: [orders.id],
    relationName: 'history', // Збігається з ordersRelations
  }),
  courier: one(courierProfiles, {
    fields: [orderHistory.courierId],
    references: [courierProfiles.id],
  }),
```

```
}));

export const orderTariffsRelations = relations(orderTariffs, ({
one }) => ({
   order: one(orders, {
      fields: [orderTariffs.orderId],
      references: [orders.id],
      relationName: 'tariffs', // Збігається з ordersRelations
   }),
   tariff: one(tariffs, {
      fields: [orderTariffs.tariffId],
      references: [tariffs.id],
      relationName: 'tariffOrders',
   }),
}));

// Тут RoutePoints має два relationName, бо він є child і для
Order, і для Route
export const routePointsRelations = relations(routePoints, ({
one }) => ({
   order: one(orders, {
      fields: [routePoints.orderId],
      references: [orders.id],
      relationName: 'routePoints', // Збігається з
ordersRelations
   }),
   route: one(routes, {
      fields: [routePoints.routeId],
      references: [routes.id],
      relationName: 'routePointsList',
   }),
}));

// ----------------------------------------------------------------------
---
// Зв’язки інших сутностей
// ----------------------------------------------------------------------
---

export const customerProfilesRelations =
relations(customerProfiles, ({ one, many }) => ({
   user: one(users, {
      fields: [customerProfiles.userId],
      references: [users.id],
   }),
   orders: many(orders),
}));

export const courierProfilesRelations =
relations(courierProfiles, ({ one, many }) => ({
   user: one(users, {
      fields: [courierProfiles.userId],
      references: [users.id],
   }),
   orders: many(orders),
   vehicles: many(vehicles, { relationName: 'courierVehicles'
}),
   vehicleHistory: many(vehicleAssignmentsHistory, {
relationName: 'courierVehicleHistory' }),
   routes: many(routes, { relationName: 'courierRoutes' }),
}));

export const routesRelations = relations(routes, ({ one, many })
=> ({
   courier: one(courierProfiles, {
      fields: [routes.courierId],
      references: [courierProfiles.id],
      relationName: 'courierRoutes',
   }),
   points: many(routePoints, { relationName: 'routePointsList'
}),
}));

export const vehiclesRelations = relations(vehicles, ({ one,
many }) => ({
   courier: one(courierProfiles, {
      fields: [vehicles.currentCourierId],
      references: [courierProfiles.id],
      relationName: 'courierVehicles',
   }),
   history: many(vehicleAssignmentsHistory, { relationName:
'vehicleHistory' }),
}));

export const vehicleAssignmentsHistoryRelations = relations(
   vehicleAssignmentsHistory,
   ({ one }) => ({
      vehicle: one(vehicles, {
         fields: [vehicleAssignmentsHistory.vehicleId],
         references: [vehicles.id],
         relationName: 'vehicleHistory',
      }),
      courier: one(courierProfiles, {
         fields: [vehicleAssignmentsHistory.courierId],
         references: [courierProfiles.id],
         relationName: 'courierVehicleHistory',
      }),
   })
);

export const rolesRelations = relations(roles, ({ many }) => ({
   users: many(userRoles, { relationName: 'roleUsers' }),
   permissions: many(rolePermissions, { relationName:
'rolePermissions' }),
}));

export const permissionsRelations = relations(permissions, ({
many }) => ({
   roles: many(rolePermissions, { relationName:
'permissionRoles' }),
}));

export const rolePermissionsRelations =
relations(rolePermissions, ({ one }) => ({
   role: one(roles, {
      fields: [rolePermissions.roleId],
      references: [roles.id],
      relationName: 'rolePermissions',
   }),
   permission: one(permissions, {
      fields: [rolePermissions.permissionId],
      references: [permissions.id],
      relationName: 'permissionRoles',
   }),
}));

export const tariffsRelations = relations(tariffs, ({ many }) =>
({
   orders: many(orderTariffs, { relationName: 'tariffOrders' }),
}));

export const appsRelations = relations(apps, ({ many }) => ({
   sessions: many(userSessions),
}));
import { AppConfig } from '@/config/configuration';
import { Global, Module } from '@nestjs/common';
import { ConfigService } from '@nestjs/config';
import * as schema from './schema';
```

```
import { drizzle, type NodePgDatabase } from 'drizzle-
orm/node-postgres';
import { Pool } from 'pg';

export const DB_CLIENT = Symbol('DB_CLIENT');
export type DbClient = NodePgDatabase<typeof schema>;

@Global()
@Module({
  providers: [
    {
      provide: DB_CLIENT,
      inject: [ConfigService],
      useFactory: (configService:
ConfigService<AppConfig>) => {
        const pool = new Pool({
          connectionString: configService.get('database.url',
{ infer: true }),
        });

        return drizzle(pool, { schema }) as DbClient;
      },
    },
  ],
  exports: [DB_CLIENT],
})
export class DbModule {}
import {
  SQL,
  and,
  or,
  not,
  eq,
  ne,
  gt,
  gte,
  lt,
  lte,
  ilike,
  notIlike,
  inArray,
  notInArray,
  isNull,
  isNotNull,
  asc,
  desc,
  Column,
  AnyColumn,
} from 'drizzle-orm';
import { PgTable } from 'drizzle-orm/pg-core';
import { BadRequestException } from '@nestjs/common';

export type DrizzleColumnNames<TTable extends PgTable> =
{
  [K in keyof TTable]: TTable[K] extends AnyColumn ? K &
string : never;
}[keyof TTable];

export class QueryBuilderUtil {
  /**
   * Map of allowed operations per Drizzle data type.
   * Prevents operations like `like` on numbers, or `gt` on
strings.
   */
  private static readonly VALID_OPERATORS_BY_TYPE:
Record<string, string[]> = {
    string: ['eq', 'ne', 'in', 'notIn', 'like', 'notLike', 'startsWith',
'endsWith', 'isNull'],
    number: ['eq', 'ne', 'gt', 'gte', 'lt', 'lte', 'in', 'notIn', 'isNull'],
    date: ['eq', 'ne', 'gt', 'gte', 'lt', 'lte', 'in', 'notIn', 'isNull'],
    boolean: ['eq', 'ne', 'isNull'],
  };

  /**
   * Validates if the requested operator is compatible with the
column's data type.
   */
  private static validateOperator(column: Column, operator:
string, field: string): void {
    const dataType = column.dataType;
    const allowedOperators =
      this.VALID_OPERATORS_BY_TYPE[dataType] ||
this.VALID_OPERATORS_BY_TYPE.string;

    if (!allowedOperators.includes(operator)) {
      throw new BadRequestException(
        `Operator '${operator}' is not supported for field
'${field}' (type: ${dataType}). Allowed:
${allowedOperators.join(', ')}`
      );
    }
  }

  /**
   * Strictly parses and validates values using 'unknown' for
type safety.
   * Throws 400 Bad Request on format mismatch instead of
letting Drizzle throw 500.
   */
  private static parseValue(column: Column, value: unknown,
field: string): unknown {
    if (value === undefined || value === null) return value;

    switch (column.dataType) {
      case 'date':
        if (typeof value === 'string' || typeof value ===
'number') {
          let dateInput: string | number = value;

          if (typeof value === 'string' &&
/^\d+$/.test(value)) {
            if (value.length >= 10) {
              const timestamp = Number(value);
              dateInput = value.length === 10 ? timestamp
* 1000 : timestamp;
            }
          }

          const date = new Date(dateInput);
          if (isNaN(date.getTime())) {
            throw new BadRequestException(
              `Invalid date format provided for field
'${field}': ${String(value)}`
            );
          }
          return date;
        }
        throw new BadRequestException(
          `Field '${field}' expects a date string or timestamp`
        );
      case 'number':
        if (typeof value === 'string' || typeof value ===
'number') {
          const num = Number(value);
          if (isNaN(num)) {
            throw new BadRequestException(
              `Invalid numeric format provided for field
'${field}': ${String(value)}`
            );
          }
```

```
            return num;
          }
          throw new BadRequestException(`Field '${field}'
expects a number`);
        case 'boolean':
          if (typeof value === 'string') {
            const lower = value.toLowerCase();
            if (lower !== 'true' && lower !== 'false') {
              throw new BadRequestException(
                `Invalid boolean format for field '${field}'.
Expected 'true' or 'false'`
              );
            }
            return lower === 'true';
          } else if (typeof value === 'boolean') {
            return value;
          }
          throw new BadRequestException(`Field '${field}'
expects a boolean`);
      }

      return value;
    }

    /**
     * Recursively parses the filter object into Drizzle ORM SQL
conditions.
     */
    static buildWhere<TTable extends PgTable>(
      filter: Record<string, unknown> | undefined,
      table: PgTable,
      allowedFields: (DrizzleColumnNames<TTable> | (string
& {}))[]
    ): SQL | undefined {
      if (!filter || typeof filter !== 'object' ||
Object.keys(filter).length === 0) {
        return undefined;
      }

      const conditions: SQL[] = [];

      for (const [key, value] of Object.entries(filter)) {
        // Logical Operators
        if (key === '$and') {
          if (!Array.isArray(value))
            throw new BadRequestException(`Operator '$and'
expects an array`);
          conditions.push(
            and(
              ...value.map((v: unknown) =>
                this.buildWhere(v as Record<string,
unknown>, table, allowedFields)
              )!
            )!
          );
        } else if (key === '$or') {
          if (!Array.isArray(value))
            throw new BadRequestException(`Operator '$or'
expects an array`);
          conditions.push(
            or(
              ...value.map((v: unknown) =>
                this.buildWhere(v as Record<string,
unknown>, table, allowedFields)
              )!
            )!
          );
        } else if (key === '$not') {
          const innerCondition: unknown =
Array.isArray(value)
            ? (value as unknown[])[0]
            : value;
          conditions.push(
            not(
              this.buildWhere(
                innerCondition as Record<string, unknown>,
                table,
                allowedFields
              )!
            )!
          );
        }
        // Field Operators
        else {
          if (!allowedFields.includes(key)) {
            throw new BadRequestException(`Unknown field
'${key}' for filtering`);
          }

          if (typeof value !== 'object' || value === null) {
            throw new BadRequestException(
              `Value for field '${key}' must be an object of
operators`
            );
          }

          const column = table[key] as Column;

          // Iterate through all operators provided for this field
          for (const [operator, rawValue] of Object.entries(
            value as Record<string, unknown>
          )) {
            this.validateOperator(column, operator, key);

            const parse = (v: unknown) =>
this.parseValue(column, v, key);

            switch (operator) {
              case 'eq':
                conditions.push(eq(column,
parse(rawValue)));
                break;
              case 'ne':
                conditions.push(ne(column,
parse(rawValue)));
                break;
              case 'gt':
                conditions.push(gt(column,
parse(rawValue)));
                break;
              case 'gte':
                conditions.push(gte(column,
parse(rawValue)));
                break;
              case 'lt':
                conditions.push(lt(column,
parse(rawValue)));
                break;
              case 'lte':
                conditions.push(lte(column,
parse(rawValue)));
                break;
              case 'in':
                if (!Array.isArray(rawValue))
                  throw new BadRequestException(
                    `Operator 'in' for field '${key}' requires
an array`
                  );
                conditions.push(inArray(column,
rawValue.map(parse)));
```

```
                    break;
                case 'notIn':
                    if (!Array.isArray(rawValue))
                        throw new BadRequestException(
                            `Operator 'notIn' for field '${key}'
requires an array`
                        );
                    conditions.push(notInArray(column,
rawValue.map(parse)));
                    break;
                case 'like':
                    conditions.push(ilike(column,
`%${String(rawValue)}%`));
                    break;
                case 'notLike':
                    conditions.push(notIlike(column,
`%${String(rawValue)}%`));
                    break;
                case 'startsWith':
                    conditions.push(ilike(column,
`${String(rawValue)}%`));
                    break;
                case 'endsWith':
                    conditions.push(ilike(column,
`%${String(rawValue)}`));
                    break;
                case 'isNull': {
                    // Додано фігурні дужки для скоупу
змінної (no-case-declarations)
                    const isNullFlag =
String(rawValue).toLowerCase() === 'true';
                    conditions.push(isNullFlag ? isNull(column) :
isNotNull(column));
                    break;
                }
                default:
                    throw new BadRequestException(
                        `Unknown operator '${operator}' for field
'${key}'`
                    );
            }
        }
    }

    return conditions.length > 0 ? and(...conditions) :
undefined;
  }

  /**
   * Parses sort array ['field:asc'] into Drizzle ORM order
statements.
   */
  static buildOrderBy<TTable extends PgTable>(
    sort: string[] | undefined,
    table: PgTable,
    allowedFields: (DrizzleColumnNames<TTable> | (string
& {}))[]
  ): SQL[] {
    if (!sort || sort.length === 0) return [];

    return sort.map((sortItem) => {
      const [field, direction] = sortItem.split(':');

      if (!allowedFields.includes(field)) {
        throw new BadRequestException(`Unknown field
'${field}' for sorting`);
      }

      const column = table[field] as AnyColumn;
      return direction === 'desc' ? desc(column) :
asc(column);
    });
  }
}
/**
 * QueryBuilderUtil – unit tests
 *
 * Strategy: mock the entire 'drizzle-orm' module so tests never
touch a real DB.
 * Each mocked function returns a tagged plain object { _tag,
_args } which lets us
 * assert on the *structure* of the SQL tree rather than on
generated SQL strings.
 *
 * Bun hoists mock.module() calls before static imports, so the
import of
 * QueryBuilderUtil below will already see the mock in place.
 */

import { describe, expect, it, mock } from 'bun:test';
import { BadRequestException } from '@nestjs/common';

// ─── Drizzle mock
──────────────────────────────────────────────────────────
──────────────────────────────────

type SQLMock = { _tag: string; _args: unknown[] };

/** Small factory that creates an inspectable "SQL node". */
const node = (tag: string, ...args: unknown[]): SQLMock => ({
_tag: tag, _args: args });

void mock.module('drizzle-orm', () => ({
  // Comparison
  eq: (c: unknown, v: unknown) => node('eq', c, v),
  ne: (c: unknown, v: unknown) => node('ne', c, v),
  gt: (c: unknown, v: unknown) => node('gt', c, v),
  gte: (c: unknown, v: unknown) => node('gte', c, v),
  lt: (c: unknown, v: unknown) => node('lt', c, v),
  lte: (c: unknown, v: unknown) => node('lte', c, v),
  // Array
  inArray: (c: unknown, v: unknown) => node('inArray', c, v),
  notInArray: (c: unknown, v: unknown) => node('notInArray',
c, v),
  // String pattern
  ilike: (c: unknown, p: unknown) => node('ilike', c, p),
  notIlike: (c: unknown, p: unknown) => node('notIlike', c, p),
  // Null
  isNull: (c: unknown) => node('isNull', c),
  isNotNull: (c: unknown) => node('isNotNull', c),
  // Logical – mirror real drizzle signatures
  and: (...conds: unknown[]) => {
    const valid = conds.filter(Boolean);
    return valid.length ? node('and', ...valid) : undefined;
  },
  or: (...conds: unknown[]) => node('or', ...conds),
  not: (cond: unknown) => node('not', cond),
  // Sorting
  asc: (c: unknown) => node('asc', c),
  desc: (c: unknown) => node('desc', c),
  // Re-exported types (only need to exist at runtime)
  Column: class {},
  AnyColumn: class {},
  SQL: class {},
}));

// ─── Imports (must come AFTER mock.module)
──────────────────────────────────────────────────────────
```

```
import { QueryBuilderUtil } from './query-builder.util';
import type { Column } from 'drizzle-orm';
import type { PgTable } from 'drizzle-orm/pg-core';

// ——— Test fixtures ———————————————————————————————————————————

/** Creates the minimal Column shape required by QueryBuilderUtil at runtime. */
const col = (dataType: string): Column => ({ dataType }) as unknown as Column;

/**
 * Mock table that covers every supported column data-type.
 * Cast as PgTable – QueryBuilderUtil accesses columns via `table[key]`.
 */
const table = {
  // string
  name: col('string'),
  email: col('string'),
  // number
  age: col('number'),
  rating: col('number'),
  // date
  createdAt: col('date'),
  deletedAt: col('date'),
  // boolean
  isActive: col('boolean'),
  isDeleted: col('boolean'),
} as unknown as PgTable;

const ALL_FIELDS = [
  'name',
  'email',
  'age',
  'rating',
  'createdAt',
  'deletedAt',
  'isActive',
  'isDeleted',
];

// ——— Helpers ——————————————————————————————————————————————————

/** Unwraps the first condition out of the top-level `and(...)` wrapper. */
const firstCondition = (result: unknown): SQLMock => (result as SQLMock)._args[0] as SQLMock;

// ═══════════════════════════════════════════════════════════════════
// buildWhere()
// ═══════════════════════════════════════════════════════════════════

describe('QueryBuilderUtil', () => {
  describe('buildWhere()', () => {
    // ── Edge cases ──────────────────────────────────────────────────

    describe('edge cases', () => {
      it('returns undefined when filter is undefined', () => {
        expect(QueryBuilderUtil.buildWhere(undefined, table, ALL_FIELDS)).toBeUndefined();
      });

      it('returns undefined when filter is an empty object', () => {
        expect(QueryBuilderUtil.buildWhere({}, table, ALL_FIELDS)).toBeUndefined();
      });

      it('throws BadRequestException for an unknown field', () => {
        expect(() =>
          QueryBuilderUtil.buildWhere({ unknownField: { eq: 'x' } }, table, ALL_FIELDS)
        ).toThrow(BadRequestException);
      });

      it('throws BadRequestException when field value is a primitive (not an operator object)', () => {
        expect(() =>
          QueryBuilderUtil.buildWhere({ name: 'raw-string' }, table, ALL_FIELDS)
        ).toThrow(BadRequestException);
      });

      it('throws BadRequestException when field value is null', () => {
        expect(() =>
          QueryBuilderUtil.buildWhere({ name: null }, table, ALL_FIELDS)
        ).toThrow(BadRequestException);
      });

      it('throws BadRequestException for a completely unknown operator', () => {
        expect(() =>
          QueryBuilderUtil.buildWhere({ name: { contains: 'x' } }, table, ALL_FIELDS)
        ).toThrow(BadRequestException);
      });

      it('wraps multiple top-level field conditions with and()', () => {
        const result = QueryBuilderUtil.buildWhere(
          { name: { eq: 'Alice' }, age: { gte: '18' }, isActive: { eq: 'true' } },
          table,
          ALL_FIELDS
        ) as unknown as SQLMock;

        // All three conditions must be siblings inside a single and() node.
        expect(result._tag).toBe('and');
        expect(result._args).toHaveLength(3);
      });
    });

    // ── String column – all operators ──────────────────────────────

    describe('string column operators', () => {
      it('eq – produces eq node with the raw string value', () => {
        const cond = firstCondition(
          QueryBuilderUtil.buildWhere({ name: { eq: 'Alice' } }, table, ALL_FIELDS)
        );
```

```
            expect(cond._tag).toBe('eq');
            expect(cond._args[1]).toBe('Alice');
        });

        it('ne – produces ne node', () => {
            const cond = firstCondition(
                QueryBuilderUtil.buildWhere({ name: { ne: 'Bob'
} }, table, ALL_FIELDS)
            );

            expect(cond._tag).toBe('ne');
            expect(cond._args[1]).toBe('Bob');
        });

        it('like – wraps value with %…% wildcard in ilike
node', () => {
            const cond = firstCondition(
                QueryBuilderUtil.buildWhere({ name: { like: 'ali'
} }, table, ALL_FIELDS)
            );

            expect(cond._tag).toBe('ilike');
            expect(cond._args[1]).toBe('%ali%');
        });

        it('notLike – wraps value with %…% wildcard in
notIlike node', () => {
            const cond = firstCondition(
                QueryBuilderUtil.buildWhere({ name: { notLike:
'ali' } }, table, ALL_FIELDS)
            );

            expect(cond._tag).toBe('notIlike');
            expect(cond._args[1]).toBe('%ali%');
        });

        it('startsWith – produces ilike node with value%
pattern', () => {
            const cond = firstCondition(
                QueryBuilderUtil.buildWhere({ name: {
startsWith: 'Al' } }, table, ALL_FIELDS)
            );

            expect(cond._tag).toBe('ilike');
            expect(cond._args[1]).toBe('Al%');
        });

        it('endsWith – produces ilike node with %value pattern',
() => {
            const cond = firstCondition(
                QueryBuilderUtil.buildWhere({ name: {
endsWith: 'ice' } }, table, ALL_FIELDS)
            );

            expect(cond._tag).toBe('ilike');
            expect(cond._args[1]).toBe('%ice');
        });

        it('in – produces inArray node with array of values', ()
=> {
            const cond = firstCondition(
                QueryBuilderUtil.buildWhere(
                    { name: { in: ['Alice', 'Bob'] } },
                    table,
                    ALL_FIELDS
                )
            );

            expect(cond._tag).toBe('inArray');
            expect(cond._args[1]).toEqual(['Alice', 'Bob']);
        });

        it('notIn – produces notInArray node', () => {
            const cond = firstCondition(
                QueryBuilderUtil.buildWhere({ name: { notIn:
['Alice'] } }, table, ALL_FIELDS)
            );

            expect(cond._tag).toBe('notInArray');
        });

        it('isNull: "true" – produces isNull node', () => {
            const cond = firstCondition(
                QueryBuilderUtil.buildWhere({ name: { isNull:
'true' } }, table, ALL_FIELDS)
            );

            expect(cond._tag).toBe('isNull');
        });

        it('isNull: "false" – produces isNotNull node (field must
be present)', () => {
            const cond = firstCondition(
                QueryBuilderUtil.buildWhere({ name: { isNull:
'false' } }, table, ALL_FIELDS)
            );

            expect(cond._tag).toBe('isNotNull');
        });

        it('in – throws BadRequestException when value is not
an array', () => {
            expect(() =>
                QueryBuilderUtil.buildWhere({ name: { in: 'Alice'
} }, table, ALL_FIELDS)
            ).toThrow(BadRequestException);
        });

        it('notIn – throws BadRequestException when value is
not an array', () => {
            expect(() =>
                QueryBuilderUtil.buildWhere({ name: { notIn:
'Alice' } }, table, ALL_FIELDS)
            ).toThrow(BadRequestException);
        });

        it('gt – throws BadRequestException (not supported for
string columns)', () => {
            expect(() =>
                QueryBuilderUtil.buildWhere({ name: { gt: 'z' } },
table, ALL_FIELDS)
            ).toThrow(BadRequestException);
        });
    });

    // ── Number column – all operators + type parsing
──────────────────────────────

    describe('number column operators', () => {
        it('eq – parses "25" string to number 25', () => {
            const cond = firstCondition(
                QueryBuilderUtil.buildWhere({ age: { eq: '25' } },
table, ALL_FIELDS)
            );

            expect(cond._tag).toBe('eq');
            expect(cond._args[1]).toBe(25);
        });

        it('ne – parses value to number', () => {
```

```
            const cond = firstCondition(
                QueryBuilderUtil.buildWhere({ age: { ne: '30' } },
table, ALL_FIELDS)
            );

            expect(cond._tag).toBe('ne');
            expect(cond._args[1]).toBe(30);
        });

        it('gt – produces gt node with numeric value', () => {
            const cond = firstCondition(
                QueryBuilderUtil.buildWhere({ age: { gt: '18' } },
table, ALL_FIELDS)
            );

            expect(cond._tag).toBe('gt');
            expect(cond._args[1]).toBe(18);
        });

        it('gte – produces gte node', () => {
            const cond = firstCondition(
                QueryBuilderUtil.buildWhere({ age: { gte: '18' } },
table, ALL_FIELDS)
            );

            expect(cond._tag).toBe('gte');
            expect(cond._args[1]).toBe(18);
        });

        it('lt – produces lt node', () => {
            const cond = firstCondition(
                QueryBuilderUtil.buildWhere({ age: { lt: '65' } },
table, ALL_FIELDS)
            );

            expect(cond._tag).toBe('lt');
            expect(cond._args[1]).toBe(65);
        });

        it('lte – produces lte node', () => {
            const cond = firstCondition(
                QueryBuilderUtil.buildWhere({ age: { lte: '65' } },
table, ALL_FIELDS)
            );

            expect(cond._tag).toBe('lte');
            expect(cond._args[1]).toBe(65);
        });

        it('in – maps each string array element to a number', ()
=> {
            const cond = firstCondition(
                QueryBuilderUtil.buildWhere(
                    { age: { in: ['18', '25', '30'] } },
                    table,
                    ALL_FIELDS
                )
            );

            expect(cond._tag).toBe('inArray');
            expect(cond._args[1]).toEqual([18, 25, 30]);
        });

        it('accepts a native number value (no string coercion
needed)', () => {
            const cond = firstCondition(
                QueryBuilderUtil.buildWhere({ age: { eq: 42 } },
table, ALL_FIELDS)
            );

            expect(cond._args[1]).toBe(42);
        });

        it('throws BadRequestException for a non-numeric
string like "abc"', () => {
            expect(() =>
                QueryBuilderUtil.buildWhere({ age: { eq: 'abc' }
}, table, ALL_FIELDS)
            ).toThrow(BadRequestException);
        });

        it('like – throws BadRequestException (not supported
for number columns)', () => {
            expect(() =>
                QueryBuilderUtil.buildWhere({ age: { like: '2' } },
table, ALL_FIELDS)
            ).toThrow(BadRequestException);
        });
    });

    // ── Date column – all operators + all parse strategies
──────────────────────

    describe('date column operators', () => {
        const ISO_DATE = '2024-06-15T12:00:00.000Z';

        it('eq – parses ISO 8601 string into a Date instance', ()
=> {
            const cond = firstCondition(
                QueryBuilderUtil.buildWhere({ createdAt: { eq:
ISO_DATE } }, table, ALL_FIELDS)
            );

            expect(cond._tag).toBe('eq');
            expect(cond._args[1]).toBeInstanceOf(Date);
            expect((cond._args[1] as
Date).toISOString()).toBe(ISO_DATE);
        });

        it('eq – parses a 10-digit unix timestamp (seconds) and
converts to ms', () => {
            // The util multiplies 10-digit numbers by 1000 to
convert s → ms.
            const unixSeconds = '1718452800'; // 2024-06-
15T12:00:00Z in seconds
            const cond = firstCondition(
                QueryBuilderUtil.buildWhere(
                    { createdAt: { eq: unixSeconds } },
                    table,
                    ALL_FIELDS
                )
            );

            expect(cond._args[1]).toBeInstanceOf(Date);
            expect((cond._args[1] as
Date).getTime()).toBe(1718452800 * 1000);
        });

        it('eq – parses a 13-digit unix timestamp (milliseconds)
as-is', () => {
            const unixMs = '1718452800000';
            const cond = firstCondition(
                QueryBuilderUtil.buildWhere({ createdAt: { eq:
unixMs } }, table, ALL_FIELDS)
            );

            expect((cond._args[1] as
Date).getTime()).toBe(1718452800000);
        });
```

```
        it('gt – produces gt node with a parsed Date', () => {
            const cond = firstCondition(
                QueryBuilderUtil.buildWhere({ createdAt: { gt: ISO_DATE } }, table, ALL_FIELDS)
            );

            expect(cond._tag).toBe('gt');
            expect(cond._args[1]).toBeInstanceOf(Date);
        });

        it('gte – produces gte node', () => {
            const cond = firstCondition(
                QueryBuilderUtil.buildWhere({ createdAt: { gte: ISO_DATE } }, table, ALL_FIELDS)
            );

            expect(cond._tag).toBe('gte');
        });

        it('lt – produces lt node', () => {
            const cond = firstCondition(
                QueryBuilderUtil.buildWhere({ createdAt: { lt: ISO_DATE } }, table, ALL_FIELDS)
            );

            expect(cond._tag).toBe('lt');
        });

        it('lte – produces lte node', () => {
            const cond = firstCondition(
                QueryBuilderUtil.buildWhere({ createdAt: { lte: ISO_DATE } }, table, ALL_FIELDS)
            );

            expect(cond._tag).toBe('lte');
        });

        it('throws BadRequestException for a completely invalid date string', () => {
            expect(() =>
                QueryBuilderUtil.buildWhere(
                    { createdAt: { eq: 'not-a-date' } },
                    table,
                    ALL_FIELDS
                )
            ).toThrow(BadRequestException);
        });

        it('throws BadRequestException when the date value is a plain object', () => {
            expect(() =>
                QueryBuilderUtil.buildWhere(
                    { createdAt: { eq: { year: 2024 } } },
                    table,
                    ALL_FIELDS
                )
            ).toThrow(BadRequestException);
        });

        it('like – throws BadRequestException (not supported for date columns)', () => {
            expect(() =>
                QueryBuilderUtil.buildWhere({ createdAt: { like: '2024' } }, table, ALL_FIELDS)
            ).toThrow(BadRequestException);
        });
    });

    // ── Boolean column – all operators + type coercion ──────────────────────

    describe('boolean column operators', () => {
        it('eq – coerces "true" string to boolean true', () => {
            const cond = firstCondition(
                QueryBuilderUtil.buildWhere({ isActive: { eq: 'true' } }, table, ALL_FIELDS)
            );

            expect(cond._tag).toBe('eq');
            expect(cond._args[1]).toBe(true);
        });

        it('eq – coerces "false" string to boolean false', () => {
            const cond = firstCondition(
                QueryBuilderUtil.buildWhere({ isActive: { eq: 'false' } }, table, ALL_FIELDS)
            );

            expect(cond._args[1]).toBe(false);
        });

        it('eq – case-insensitive: "TRUE" → true', () => {
            const cond = firstCondition(
                QueryBuilderUtil.buildWhere({ isActive: { eq: 'TRUE' } }, table, ALL_FIELDS)
            );

            expect(cond._args[1]).toBe(true);
        });

        it('eq – case-insensitive: "FALSE" → false', () => {
            const cond = firstCondition(
                QueryBuilderUtil.buildWhere({ isActive: { eq: 'FALSE' } }, table, ALL_FIELDS)
            );

            expect(cond._args[1]).toBe(false);
        });

        it('eq – native boolean true passes through without coercion', () => {
            const cond = firstCondition(
                QueryBuilderUtil.buildWhere({ isActive: { eq: true } }, table, ALL_FIELDS)
            );

            expect(cond._args[1]).toBe(true);
        });

        it('ne – produces ne node with coerced boolean', () => {
            const cond = firstCondition(
                QueryBuilderUtil.buildWhere({ isActive: { ne: 'false' } }, table, ALL_FIELDS)
            );

            expect(cond._tag).toBe('ne');
            expect(cond._args[1]).toBe(false);
        });

        it('isNull: "true" – produces isNull node', () => {
            const cond = firstCondition(
                QueryBuilderUtil.buildWhere({ isActive: { isNull: 'true' } }, table, ALL_FIELDS)
            );

            expect(cond._tag).toBe('isNull');
        });

        it('throws BadRequestException for "yes" (not a valid boolean string)', () => {
```

```
            expect(() =>
                QueryBuilderUtil.buildWhere({ isActive: { eq:
'yes' } }, table, ALL_FIELDS)
            ).toThrow(BadRequestException);
        });

        it('throws BadRequestException for numeric 1 (number
is not a valid boolean)', () => {
            expect(() =>
                QueryBuilderUtil.buildWhere({ isActive: { eq: 1 }
}, table, ALL_FIELDS)
            ).toThrow(BadRequestException);
        });

        it('gt – throws BadRequestException (not supported for
boolean columns)', () => {
            expect(() =>
                QueryBuilderUtil.buildWhere({ isActive: { gt:
'true' } }, table, ALL_FIELDS)
            ).toThrow(BadRequestException);
        });

        it('like – throws BadRequestException (not supported
for boolean columns)', () => {
            expect(() =>
                QueryBuilderUtil.buildWhere({ isActive: { like:
'true' } }, table, ALL_FIELDS)
            ).toThrow(BadRequestException);
        });
    });

    // ── Logical operators
──────────────────────────────────────────────────────────────

    describe('logical operators', () => {
        it('$and – produces an and() node containing each sub-
condition', () => {
            const result = QueryBuilderUtil.buildWhere(
                {
                    $and: [{ name: { eq: 'Alice' } }, { age: { gte: '18'
} }],
                },
                table,
                ALL_FIELDS
            ) as unknown as SQLMock;

            // Outer and() wraps the $and compound node.
            expect(result._tag).toBe('and');
            // The single arg of the outer and() is the inner and()
from $and.
            expect(firstCondition(result)._tag).toBe('and');
        });

        it('$or – produces an or() node containing each sub-
condition', () => {
            const result = QueryBuilderUtil.buildWhere(
                {
                    $or: [{ name: { eq: 'Alice' } }, { name: { eq:
'Bob' } }],
                },
                table,
                ALL_FIELDS
            ) as unknown as SQLMock;

            expect(result._tag).toBe('and');
            expect(firstCondition(result)._tag).toBe('or');
        });

        it('$not – produces a not() node wrapping the inner
condition', () => {
            const result = QueryBuilderUtil.buildWhere(
                { $not: { isActive: { eq: 'false' } } },
                table,
                ALL_FIELDS
            ) as unknown as SQLMock;

            expect(result._tag).toBe('and');
            expect(firstCondition(result)._tag).toBe('not');
        });

        it('nested $and + $or – builds a compound condition
tree', () => {
            const result = QueryBuilderUtil.buildWhere(
                {
                    $and: [
                        { $or: [{ name: { eq: 'Alice' } }, { name: { eq:
'Bob' } }] },
                        { isActive: { eq: 'true' } },
                    ],
                },
                table,
                ALL_FIELDS
            ) as unknown as SQLMock;

            // Result must be a valid (non-undefined) SQL node –
tree construction succeeded.
            expect(result).toBeDefined();
            expect(result._tag).toBe('and');
        });

        it('$and combined with a sibling field condition – AND-
ed at top level', () => {
            const result = QueryBuilderUtil.buildWhere(
                {
                    $and: [{ name: { eq: 'Alice' } }],
                    isActive: { eq: 'true' },
                },
                table,
                ALL_FIELDS
            ) as unknown as SQLMock;

            // Two conditions: the $and compound node + the
isActive condition.
            expect(result._tag).toBe('and');
            expect(result._args).toHaveLength(2);
        });

        it('$and – throws BadRequestException when value is
not an array', () => {
            expect(() =>
                QueryBuilderUtil.buildWhere({ $and: { name: {
eq: 'x' } } }, table, ALL_FIELDS)
            ).toThrow(BadRequestException);
        });

        it('$or – throws BadRequestException when value is not
an array', () => {
            expect(() =>
                QueryBuilderUtil.buildWhere({ $or: { name: { eq:
'x' } } }, table, ALL_FIELDS)
            ).toThrow(BadRequestException);
        });
    });

    // ── Multiple operators on a single field
──────────────────────────────────────────────

    describe('multiple operators on the same field', () => {
```

```
        it('gt + lt on a number column produces two sibling
conditions inside and()', () => {
            const result = QueryBuilderUtil.buildWhere(
                { age: { gt: '18', lt: '65' } },
                table,
                ALL_FIELDS
            ) as unknown as SQLMock;

            // Both gt and lt for 'age' should be siblings within the
outer and().
            expect(result._tag).toBe('and');
            expect(result._args).toHaveLength(2);

            const tags = (result._args as SQLMock[]).map((n) =>
n._tag);
            expect(tags).toContain('gt');
            expect(tags).toContain('lt');
        });

        it('gte + lte on a date column – both produce Date
instances', () => {
            const result = QueryBuilderUtil.buildWhere(
                {
                    createdAt: {
                        gte: '2024-01-01T00:00:00.000Z',
                        lte: '2024-12-31T23:59:59.999Z',
                    },
                },
                table,
                ALL_FIELDS
            ) as unknown as SQLMock;

            expect(result._args).toHaveLength(2);
            (result._args as SQLMock[]).forEach((n) => {
                expect(n._args[1]).toBeInstanceOf(Date);
            });
        });
    });
});

//
═══════════════════════════════════════════════════════════════
═══
// buildOrderBy()
//
═══════════════════════════════════════════════════════════════
═══

describe('buildOrderBy()', () => {
    it('returns an empty array when sort is undefined', () => {
        expect(QueryBuilderUtil.buildOrderBy(undefined,
table, ALL_FIELDS)).toEqual([]);
    });

    it('returns an empty array when sort is an empty array', ()
=> {
        expect(QueryBuilderUtil.buildOrderBy([], table,
ALL_FIELDS)).toEqual([]);
    });

    it('"field:asc" – produces an asc SQL node', () => {
        const result =
QueryBuilderUtil.buildOrderBy(['name:asc'], table,
ALL_FIELDS);

        expect(result).toHaveLength(1);
        expect((result[0] as unknown as
SQLMock)._tag).toBe('asc');
    });

    it('"field:desc" – produces a desc SQL node', () => {
        const result =
QueryBuilderUtil.buildOrderBy(['age:desc'], table,
ALL_FIELDS);

        expect((result[0] as unknown as
SQLMock)._tag).toBe('desc');
    });

    it('"field" with no colon – defaults to asc', () => {
        // split(':') on 'name' gives ['name'], direction is
undefined.
        // undefined !== 'desc' → asc is chosen.
        const result = QueryBuilderUtil.buildOrderBy(['name'],
table, ALL_FIELDS);

        expect((result[0] as unknown as
SQLMock)._tag).toBe('asc');
    });

    it('multiple sort fields maintain their order', () => {
        const result = QueryBuilderUtil.buildOrderBy(
            ['name:asc', 'age:desc', 'createdAt:asc'],
            table,
            ALL_FIELDS
        );

        expect(result).toHaveLength(3);

        const tags = (result as unknown as
SQLMock[]).map((n) => n._tag);
        expect(tags).toEqual(['asc', 'desc', 'asc']);
    });

    it('throws BadRequestException for an unknown sort
field', () => {
        expect(() =>

QueryBuilderUtil.buildOrderBy(['unknownField:asc'], table,
ALL_FIELDS)
        ).toThrow(BadRequestException);
    });

    it('throws BadRequestException even when one of
multiple fields is unknown', () => {
        expect(() =>
            QueryBuilderUtil.buildOrderBy(['name:asc',
'ghost:desc'], table, ALL_FIELDS)
        ).toThrow(BadRequestException);
    });
  });
});
import { customType } from 'drizzle-orm/pg-core';

export const geometryPoint = customType<{
  data: { lat: number; lng: number };
  driverData: string;
}>({
  dataType() {
    return 'geometry(Point, 4326)';
  },
  toDriver(value) {
    // Convert JS object to Extended Well-Known Text
(EWKT) format for insert/update
    // Important: Longitude (X) comes before Latitude (Y)
    return `SRID=4326;POINT(${value.lng} ${value.lat})`;
  },
  fromDriver(value) {
```

```
      // Postgres returns PostGIS geometry as a HEX string
(Extended Well-Known Binary)
      if (!value) return { lat: 0, lng: 0 };

      try {
        const buffer = Buffer.from(value, 'hex');

        // Byte 0 determines endianness (1 = Little Endian, 0 =
Big Endian)
        const isLittleEndian = buffer[0] === 1;

        // X coordinate (Longitude) starts at byte 9
        const lng = isLittleEndian ? buffer.readDoubleLE(9) :
buffer.readDoubleBE(9);

        // Y coordinate (Latitude) starts at byte 17
        const lat = isLittleEndian ? buffer.readDoubleLE(17) :
buffer.readDoubleBE(17);

        return { lat, lng };
      } catch (error) {
        console.error('[PostGIS Parse Error]: Failed to parse
WKB string:', error);
        return { lat: 0, lng: 0 };
      }
    },
});
import { Injectable } from '@nestjs/common';
import { eq, sql } from 'drizzle-orm';
import { type PgTable } from 'drizzle-orm/pg-core';
import { type DbClient } from '@/db/db.module';
import { BaseQueryDto } from '@/common/dto/base-
query.dto';
import { DrizzleColumnNames, QueryBuilderUtil } from
'@/db/utils/query-builder.util';

export type InferIdType<TTable extends PgTable> =
TTable['_']['columns'] extends {
  id: infer TColumn;
}
  ? TColumn extends { _: { data: infer TData } }
    ? TData
    : string | number
  : string | number;

export interface IBaseRepository<TTable extends PgTable> {
  findAll(): Promise<TTable['_']['inferSelect'][]>;
  findOne(id: InferIdType<TTable>):
Promise<TTable['_']['inferSelect'] | null>;
  create(data: TTable['_']['inferInsert']):
Promise<TTable['_']['inferSelect']>;
  update(
    id: InferIdType<TTable>,
    data: Partial<TTable['_']['inferInsert']>
  ): Promise<TTable['_']['inferSelect']>;
  delete(id: InferIdType<TTable>): Promise<void>;
}

@Injectable()
export abstract class AbstractRepository<
  TTable extends PgTable,
> implements IBaseRepository<TTable> {
  constructor(
    protected readonly db: DbClient,
    protected readonly table: TTable
  ) {}

  async create(data: TTable['_']['inferInsert']):
Promise<TTable['_']['inferSelect']> {
    const [result] = (await this.db
      .insert(this.table as any)
      .values(data)
      .returning()) as unknown[];

    return result as TTable['_']['inferSelect'];
  }

  async update(
    id: InferIdType<TTable>,
    data: Partial<Omit<TTable['_']['inferInsert'], 'id' |
'createdAt' | 'updatedAt'>>
  ): Promise<TTable['_']['inferSelect']> {
    const tableWithId = this.table as TTable & { id: any };

    const [result] = (await this.db
      .update(this.table as any)
      .set({ ...data, updatedAt: sql`NOW()` })
      .where(eq(tableWithId.id, id))
      .returning()) as unknown[];

    return result as TTable['_']['inferSelect'];
  }

  async findAll(
    query?: BaseQueryDto,
    allowedFields: (DrizzleColumnNames<TTable> | (string
& {}))[] = []
  ): Promise<TTable['_']['inferSelect'][]> {
    const qb = this.db
      .select()
      .from(this.table as any)
      .$dynamic();

    if (query) {
      // Apply Filters
      if (query.filter) {
        const whereClause = QueryBuilderUtil.buildWhere(
          query.filter,
          this.table,
          allowedFields
        );
        if (whereClause) {
          qb.where(whereClause);
        }
      }

      // Apply Sorting
      if (query.sort) {
        const orderByClause =
QueryBuilderUtil.buildOrderBy(
          query.sort,
          this.table,
          allowedFields
        );
        qb.orderBy(...orderByClause);
      }

      // Apply Pagination
      if (query.limit !== undefined) {
        qb.limit(query.limit);
      }
      if (query.offset !== undefined) {
        qb.offset(query.offset);
      }
    }

    return await qb;
  }
```

```
  async findOne(id: InferIdType<TTable>):
Promise<TTable['_']['inferSelect'] | null> {
    const tableWithId = this.table as TTable & { id: any };
    const [result] = await this.db
      .select()
      .from(this.table as any)
      .where(eq(tableWithId.id, id))
      .limit(1);

    return result ?? null;
  }

  async delete(id: InferIdType<TTable>): Promise<void> {
    const tableWithId = this.table as TTable & { id: any };
    await this.db.delete(this.table as
any).where(eq(tableWithId.id, id));
  }
}

import { Module } from '@nestjs/common';
import { ConfigModule as NestConfigModule } from
'@nestjs/config';
import configuration from './configuration';

@Module({
  imports: [
    NestConfigModule.forRoot({
      load: [configuration],
      isGlobal: true,
      cache: true,
    }),
  ],
})
export class ConfigModule {}
import { readFileSync } from 'fs';
import { join } from 'path';
import * as toml from '@iarna/toml';
import { z } from 'zod';

export const portSchema = z.coerce
  .number()
  .int({ message: 'Port must be an integer' })
  .min(1, { message: 'Port must be greater than 0' })
  .max(65535, { message: 'Port must be less than or equal to
65535' });

export const configSchema = z.object({
  app: z.object({
    port: portSchema.default(3000),
    env: z.enum(['development', 'production', 'test']),
  }),
  database: z.object({
    url: z.url(),
    max_connections: z.number().min(1),
  }),
  redis: z.object({
    host: z.string(),
    port: portSchema,
  }),
  rmq: z.object({
    uri: z.string(),
    queue: z.string(),
    exchange: z.string(),
  }),
});

export type AppConfig = z.infer<typeof configSchema>;

export default () => {
    const CONFIG_PATH = join(process.cwd(),
'../../configs/core-api/config.toml');

  const fileContents = readFileSync(CONFIG_PATH, 'utf-8');
  const parsedConfig = toml.parse(fileContents);

    const validatedConfig =
configSchema.safeParse(parsedConfig);

  if (!validatedConfig.success) {
    throw new Error('Config validation failed:\n' +
z.prettifyError(validatedConfig.error));
  }

  return validatedConfig.data;
};

'use client';

import { useState, useMemo } from 'react';
import dynamic from 'next/dynamic';
import { COURIERS, generate100Points, DeliveryPoint } from
'@/lib/mockData';
import { ResizableHandle, ResizablePanel,
ResizablePanelGroup } from '@/components/ui/resizable';
import { RadioGroup, RadioGroupItem } from
'@/components/ui/radio-group';
import { Label } from '@/components/ui/label';
import { Card, CardContent, CardHeader, CardTitle,
CardDescription } from '@/components/ui/card';
import { ScrollArea } from '@/components/ui/scroll-area';
import { Badge } from '@/components/ui/badge';
import { Button } from '@/components/ui/button';
import { LoaderIcon } from 'lucide-react';
import { toast } from 'sonner';
import { se } from 'react-day-picker/locale';

// Dynamically import the map to prevent SSR issues
const DeliveryMap = dynamic(() =>
import('@/components/map'), { ssr: false });

export default function DeliveryDashboard() {
  // --- State Management ---
  const [reLoading, setReloading] = useState(false);
  const [points] =
useState<DeliveryPoint[]>(generate100Points());
  const [selectedCourierId, setSelectedCourierId] =
useState<string>('all');
  const [selectedPoint, setSelectedPoint] =
useState<DeliveryPoint | null>(null);

  // --- Derived State (Memoized for performance) ---
  const visiblePoints = useMemo(() => {
    if (selectedCourierId === 'all') return points;
    return points.filter((p) => p.courierId ===
selectedCourierId);
  }, [points, selectedCourierId]);

  const visibleCouriers = useMemo(() => {
    if (selectedCourierId === 'all') return COURIERS;
    return COURIERS.filter((c) => c.id ===
selectedCourierId);
  }, [selectedCourierId]);

  const activePointCourier = useMemo(() => {
    if (!selectedPoint) return null;
    return COURIERS.find((c) => c.id ===
selectedPoint.courierId);
  }, [selectedPoint]);
```

```
  const handleRecalculate = () => {
    setReloading(true);

    toast.promise(
      new Promise((resolve) => {
        setTimeout(() => {
          resolve('Re-calculation complete!');
          setReloading(false);
        }, 1000);
      }),
      {
        loading: 'Re-calculating route...',
        success: () =>
          `Route for ${visibleCouriers[0]?.firstName || ''} ${visibleCouriers[0]?.lastName || ''} re-calculated successfully!`,
        error: 'Failed to re-calculate route.',
      }
    );
  };

  return (
    <div className="grow rounded-xl overflow-hidden border border-slate-200 shadow-sm bg-white">
      {/* Resizable Layout wrapper */}
      <ResizablePanelGroup orientation="horizontal">
        {/* COLUMN 1: Courier Selector */}
        <ResizablePanel
          defaultSize="30%"
          minSize="15%"
          maxSize="40%"
          className="bg-slate-50 p-4"
        >
          <h2 className="text-lg font-bold text-slate-800 mb-4">Couriers</h2>
          <ScrollArea className="h-full w-full pr-4">
            <RadioGroup value={selectedCourierId} onValueChange={setSelectedCourierId}>
              {/* Option: All */}
              <div className="flex items-center space-x-2 p-3 rounded-lg border border-slate-200 bg-white hover:bg-slate-100 transition-colors mb-2">
                <RadioGroupItem value="all" id="all" />
                <Label htmlFor="all" className="grow cursor-pointer font-semibold">
                  All Couriers
                </Label>
              </div>

              {/* Specific Couriers */}
              {COURIERS.map((courier) => (
                <div
                  key={courier.id}
                  className="flex items-start space-x-3 p-3 rounded-lg border border-slate-200 bg-white hover:border-indigo-300 transition-colors mb-2 cursor-pointer"
                >
                  <RadioGroupItem
                    value={courier.id}
                    id={courier.id}
                    className="mt-1"
                  />
                  <Label
                    htmlFor={courier.id}
                    className="flex flex-col cursor-pointer w-full gap-1"
                  >
                    <span className="font-semibold text-sm">
                      {courier.firstName} {courier.lastName}
                    </span>
                    <div className="flex items-center justify-between">
                      <Badge
                        variant="secondary"
                        className="font-mono text-[10px]"
                      >
                        {courier.licensePlate}
                      </Badge>
                      <span className="text-[10px] text-slate-500">
                        Route: 25 pkgs
                      </span>
                    </div>
                  </Label>
                </div>
              ))}
            </RadioGroup>
          </ScrollArea>
        </ResizablePanel>

        <ResizableHandle withHandle />

        {/* COLUMN 2: The Map */}
        <ResizablePanel defaultSize={selectedPoint ? '50%' : '70%'} className="relative">
          <DeliveryMap
            points={visiblePoints}
            couriers={visibleCouriers}
            onPointClick={setSelectedPoint}
            selectedCourierId={selectedCourierId} // 2. FIX: Pass state to map
          />

          <div className="absolute top-4 left-4 z-10">
            {selectedCourierId !== 'all' ? (
              <Button
                variant={'default'}
                onClick={handleRecalculate}
                disabled={reLoading}
              >
                {reLoading ? <LoaderIcon className="animate-spin" /> : <></>}
                <span>Re-calculate</span>
              </Button>
            ) : (
              <></>
            )}
          </div>
        </ResizablePanel>

        {/* COLUMN 3: Point Details (Conditional Render) */}
        {selectedPoint && (
          <>
            <ResizableHandle withHandle />
            <ResizablePanel
              defaultSize="30%"
              minSize="15%"
              maxSize="40%"
              className="bg-white p-4"
            >
              <div className="flex justify-between items-center mb-4">
                <h2 className="text-lg font-bold text-slate-800">
                  Package Details
```

```
                    </h2>
                    <button
                      onClick={() => setSelectedPoint(null)}
                      className="text-slate-400 hover:text-
slate-700 text-sm font-medium"
                    >
                      ✕ Close
                    </button>
                  </div>

                  <ScrollArea className="h-[calc(100vh-
8rem)] pr-4">
                    <div className="flex flex-col gap-4 p-2
pb-10">
                      {/* General Info Card */}
                      <Card>
                        <CardHeader className="pb-2">
                          <CardDescription>Order
Code</CardDescription>
                          <CardTitle className="text-xl
text-indigo-600">
                            {selectedPoint.deliveryId}
                          </CardTitle>
                        </CardHeader>
                        <CardContent className="flex flex-
col gap-3 text-sm">
                          <div className="grid grid-cols-3
gap-1 border-b pb-2">
                            <span className="text-slate-500
col-span-1">
                              Recipient:
                            </span>
                            <span className="font-medium
col-span-2 text-right">
                              {selectedPoint.recipient}
                            </span>
                          </div>
                          <div className="grid grid-cols-3
gap-1 border-b pb-2">
                            <span className="text-slate-500
col-span-1">
                              Phone:
                            </span>
                            <span className="font-medium
col-span-2 text-right">
                              {selectedPoint.phone}
                            </span>
                          </div>
                          <div className="grid grid-cols-3
gap-1 border-b pb-2">
                            <span className="text-slate-500
col-span-1">
                              Time Slot:
                            </span>
                            <Badge
                              variant="outline"
                              className="col-span-2
justify-end border-none font-mono"
                            >
                              {selectedPoint.timeWindow}
                            </Badge>
                          </div>
                          <div className="flex flex-col gap-
1">
                            <span className="text-slate-
500">Address:</span>
                            <span className="font-
medium">
                              {selectedPoint.address}
                            </span>
                          </div>
                          <div className="flex flex-col gap-
1 mt-2 bg-amber-50 p-3 rounded-md border border-amber-
100">
                            <span className="text-amber-
800 text-xs font-bold uppercase tracking-wider">
                              Note
                            </span>
                            <span className="text-amber-
900">
                              {selectedPoint.description}
                            </span>
                          </div>
                        </CardContent>
                      </Card>

                      {/* Assigned Courier Card */}
                      {activePointCourier && (
                        <Card>
                          <CardHeader className="border-
b">
                            <CardDescription
className="text-xs font-bold uppercase text-slate-500">
                              Assigned Courier
                            </CardDescription>
                          </CardHeader>
                          <CardContent className="flex
flex-col gap-2">
                            <div className="flex justify-
between items-center">
                              <span className="font-
bold">

{activePointCourier.firstName}{' '}

{activePointCourier.lastName}
                              </span>
                              <div className="w-2 h-2
rounded-full bg-emerald-500 animate-pulse"></div>
                            </div>
                            <div className="grid grid-cols-
2 gap-2 mt-2">
                              <div className="flex flex-col
bg-slate-100 p-2 rounded">
                                <span className="text-
[10px] text-slate-500">
                                  Vehicle
                                </span>
                                <span className="font-
mono text-xs">

{activePointCourier.licensePlate}
                                </span>
                              </div>
                              <div className="flex flex-col
bg-slate-100 p-2 rounded">
                                <span className="text-
[10px] text-slate-500">
                                  Last Sync
                                </span>
                                <span className="font-
mono text-xs">

{activePointCourier.lastSynced}
                                </span>
                              </div>
                            </div>
                          </CardContent>
                        </Card>
                      )}
```

```
                    </div>
                  </ScrollArea>
                </ResizablePanel>
              </>
            )}
          </ResizablePanelGroup>
        </div>
    );
}
import React from 'react';
import { AppSidebar } from '@/components/app-sidebar';
import { Separator } from '@/components/ui/separator';
import { SidebarInset, SidebarTrigger } from
'@/components/ui/sidebar';
import { AppProviders } from './app-providers';
import { InputGroup, InputGroupInput, InputGroupAddon }
from '@/components/ui/input-group';
import { Kbd } from '@/components/ui/kbd';
import { CircleDashedIcon, SearchIcon } from 'lucide-react';

export interface IDashboardLayoutProps extends
React.PropsWithChildren {
  breadcrumbs: React.ReactNode;
}

export default function DashboardLayout({ children,
breadcrumbs }: IDashboardLayoutProps) {
  return (
    <>
      <AppProviders>
        <AppSidebar variant="sidebar" />

        <SidebarInset>
          <header className="flex h-16 shrink-0 items-
center gap-2 transition-[width,height] ease-linear group-has-
data-[collapsible=icon]/sidebar-wrapper:h-12 border-b">
            <div className="w-full flex items-center gap-2
px-4">
              <SidebarTrigger className="-ml-1" />

              <Separator orientation="vertical"
className="mr-2" />
              {breadcrumbs}

              <Separator orientation="vertical"
className="mr-2 hidden lg:block" />

              <InputGroup className="w-72 hidden
lg:flex">
                <InputGroupInput placeholder="Try
searching for something…" />
                <InputGroupAddon>
                  <SearchIcon />
                </InputGroupAddon>
                <InputGroupAddon align="inline-end">
                  <Kbd>⌘</Kbd>
                  <Kbd>K</Kbd>
                </InputGroupAddon>
              </InputGroup>
            </div>
          </header>
          {/* <div className="flex flex-1 flex-col gap-4 p-4
pt-0">
            <div className="grid auto-rows-min gap-4
md:grid-cols-3">
              <div className="aspect-video rounded-xl
bg-muted/50" />
                <div className="aspect-video rounded-
xl bg-muted/50" />
                <div className="aspect-video rounded-
xl bg-muted/50" />
              </div>
              <div  className="min-h-[100vh]  flex-1
rounded-xl bg-muted/50 md:min-h-min" />
            </div> */}
          <div className="flex flex-1 flex-col gap-4 p-
4">{children}</div>
        </SidebarInset>
      </AppProviders>
    </>
  );
}
```