Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Створення вебплатформи для відстеження замовлень товарів у режимі реального часу

Виконав: студент IV курсу, групи СН-42
спеціальності 122 Комп’ютерні науки
(шифр і назва спеціальності)

(підпис) Луців А. В.
(прізвище та ініціали)

Керівник (підпис) Готович В. А.
(прізвище та ініціали)

Нормоконтроль (підпис) Липак Г.І.
(прізвище та ініціали)

Завідувач кафедри (підпис) Боднарчук І.О.
(прізвище та ініціали)

Рецензент (підпис) Стадник Н.Б.
(прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«__»_____2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)
за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)
Студенту Луціву Арсену Володимировичу
(прізвище, ім'я, по батькові)
1. Тема роботи Створення вебплатформи для відстеження замовлень товарів у режимі реального часу
Керівник роботи Готович Володимир Анатолійович, к.т.н., доц., доцент кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом ректора від «*14*» *травня* 2026 року № *4/9-239*
2. Термін подання студентом завершеної роботи 22 червня 2026 р.
3. Вихідні дані до роботи Літературні та інтернет-джерела з технологій розробки вебтехнологій та архітектури систем, баз даних, мов програмування.
4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1 Аналіз предметної області та постановка завдання. 1.1 Особливості процесу відстеження замовлень товарів. 1.2 Аналітичний огляд існуючих аналогів відстеження замовлень. 1.3 Огляд технологій передавання даних у режимі реального часу. 1.4 Обґрунтування створення вебплатформи для відстеження замовлень товарів у режимі реального часу. 1.5 Висновок до першого розділу. 2 Проєктна частина та архітектурні рішення. 2.1 Клієнт-серверна архітектура та потік опрацювання запиту. 2.2 Діаграма прецедентів вебплатформи
Вебплатформи. 2.3 Проєктування бази даних MySQL. 2.4 Побудова оптимального маршруту та взаємодія компонентів у реальному часі. 2.5 Програмна структура, життєвий цикл та захист Даних. 2.6 Висновок до другого розділу. 3 Практична реалізація та тестування вебплатформи 3.1 Реалізація серверної логіки опрацювання замовлень. 3.2 Реалізація передавання даних у режимі реального часу. 3.3 Реалізація SQL-запитів до бази даних. 3.4 Реалізація клієнтської частини, стилізація та адаптивність. 3.5 Демонстрація роботи вебплатформи. 3.6 Тестування вебплатформи. 3.7 Аналіз результатів роботи. 3.8 Висновок до третього розділу. 4 Безпека життєдіяльності, основи охорони праці. 4.1 Безпека життєдіяльності. Долікарська допомога при ураженні електричним струмом. 4.2 Основи охорони праці. Психофізіологічне розвантаження для працівників. 4.3 Висновок до четвертого розділу. Висновки. Перелік
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Титульна сторінка. 2. Мета кваліфікаційної роботи. 3. Актуальність обраної теми. 4. Аналіз предметної області. 5. Актори програмного комплексу. 6. Використані технології. 7. Структура програмного комплекс. 8. База даних. 9. Вигляд користувацької частини сайту. 10. Вигляд панелі адміністратора. 11. Висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Гурик О. Я. кандидат технічних наук, доцент кафедри МТ		

7. Дата видачі завдання 26 січня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	26.01.2026	
2.	Підбір та опрацювання літературних джерел по темі кваліфікаційної роботи	27.01.2026-16.01.2026	
3.	Виконання дослідження щодо розробки програмного комплексу для складання розкладу занять	17.01.2026-10.05.2026	
	Розроблення функціоналу програмного комплексу		
4.	Оформлення розділу «Аналіз предметної області та постановка завдання»	11.05.2026-17.05.2026	
5.	Оформлення розділу «Проєктна частина та архітектурні рішення»	18.05.2026-24.05.2026	
6.	Оформлення розділу «Практична реалізація та тестування вебплатформи»	25.05.2026-31.05.2026	
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»	01.06.2026-08.06.2026	
8.	Виконання завдання до підрозділу «Основи охорони праці»	01.06.2026-08.06.2026	
9.	Оформлення кваліфікаційної роботи	09.06.2026-11.06.2026	
10.	Нормоконтроль		
11.	Перевірка на плагіат		
12.	Попередній захист кваліфікаційної роботи		
13.	Захист кваліфікаційної роботи	24.06.2026	

Студент _____ Луців А. В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Готович В. А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Створення вебплатформи для відстеження замовлень товарів у режимі реального часу // Кваліфікаційна робота освітнього рівня «Бакалавр» // Луців Арсен Володимирович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СН-42 // Тернопіль, 2026 // С. – 69, рис. – 16, табл. – 10, кресл. – , додат. – 5, бібліогр. – 39.

Ключові слова: вебплатформа, відстеження замовлень, режим реального часу, WebSocket, геолокація, клієнт-серверна архітектура, MySQL, алгоритм Дейкстри.

Кваліфікаційна робота присвячена дослідженню та розробці вебплатформи для відстеження замовлень товарів у режимі реального часу.

В першому розділі кваліфікаційної роботи досліджено специфіку процесів доставлення товарів та проаналізовано існуючі аналоги систем відстеження. Розглянуто технології передавання даних у режимі реального часу. Виявлено недоліки наявних рішень та обґрунтовано доцільність створення нової вебплатформи. Сформульовано постановку задачі на розробку.

В другому розділі кваліфікаційної роботи спроєктовано клієнт-серверну архітектуру вебплатформи. Розроблено схему реляційної бази даних MySQL, діаграму прецедентів та блок-схеми взаємодії компонентів. Застосовано алгоритм Дейкстри для побудови оптимального маршруту кур'єра та подано покрокову структуру його роботи.

В третьому розділі кваліфікаційної роботи описано процес практичної реалізації вебплатформи на основі PHP-фреймворку Laravel, WebSocket-сервера Ratchet та СУБД MySQL. Подано лістинги серверних контролерів, конфігурації бази даних, коду трансляції координат та клієнтської частини з інтегрованою колірною гамою інтерфейсу. Проведено функціональне тестування основних сценаріїв роботи вебплатформи.

ANNOTATION

Development of a web platform for real-time order tracking // Qualification work of the educational level «Bachelor» // Lutsiv Arsen Volodymyrovych // Ternopil Ivan Puluj National Technical University, Computer and Information Systems and Software Engineering Faculty, Computer Sciences Department, group SN-42 // Ternopil, 2026 // P. – 69, fig. – 16, tabl. – 10, chair. – , annexes. – 5, references – 39.

Keywords: web platform, order tracking, real time, WebSocket, geolocation, client-server architecture, MySQL, Dijkstra's algorithm.

The qualification work is dedicated to the research and development of a web platform for real-time order tracking.

The goal of the work is to improve the quality of delivery services by providing customers, couriers, and administrators with instant access to the current status and geographic position of orders.

The first section investigates the specifics of goods delivery processes and analyzes existing tracking solutions. Real-time data transfer technologies are reviewed, the shortcomings of available systems are identified, and the expediency of developing a dedicated web platform is justified.

In the second section of the qualification work, the client-server architecture of the web platform is designed. A diagram of the MySQL relational database, a precedent diagram, and component interaction block diagrams were developed. Dijkstra's algorithm is applied to build the optimal courier route and the step-by-step structure of its work is presented.

The third section describes the practical implementation of the web platform based on the Laravel PHP framework, the Ratchet WebSocket server, and the MySQL DBMS. Server-side controllers, database configuration, coordinate broadcasting code, and the front-end with the integrated color scheme are provided. Functional testing of the main scenarios was performed.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AJAX (англ. Asynchronous JavaScript and XML) – технологія асинхронного обміну даними браузера із сервером.

API (англ. Application Programming Interface) – прикладний програмний інтерфейс.

CSS (англ. Cascading Style Sheets) – каскадні таблиці стилів.

ERD (англ. Entity-Relationship Diagram) – діаграма «сутність–зв’язок».

GPS (англ. Global Positioning System) – глобальна система позиціонування.

HTTP (англ. HyperText Transfer Protocol) – протокол передавання гіпертексту.

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

MVC (англ. Model-View-Controller) – архітектурний шаблон «модель–подання–контролер».

ORM (англ. Object-Relational Mapping) – об’єктно-реляційне відображення.

REST (англ. Representational State Transfer) – архітектурний стиль взаємодії компонентів.

SQL (англ. Structured Query Language) – структурована мова запитів до баз даних.

UML (англ. Unified Modeling Language) – уніфікована мова моделювання.

БД – база даних.

СУБД – система управління базами даних.

ЗМІСТ

ВСТУП 8
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ 10
1.1 Особливості процесу відстеження замовлень товарів 10
1.2 Аналітичний огляд існуючих аналогів відстеження замовлень 13
1.3 Огляд технологій передавання даних у режимі реального часу 15
1.4 Обґрунтування створення вебплатформи для відстеження замовлень товарів у режимі реального часу 17
1.5 Висновок до першого розділу 19
РОЗДІЛ 2. ПРОЄКТНА ЧАСТИНА ТА АРХІТЕКТУРНІ РІШЕННЯ 20
2.1 Клієнт-серверна архітектура та потік опрацювання запиту 20
2.2 Діаграма прецедентів вебплатформи 24
2.3 Проєктування бази даних MySQL 26
2.4 Побудова оптимального маршруту та взаємодія компонентів у реальному часі 31
2.5 Програмна структура, життєвий цикл та захист даних 34
2.6 Висновок до другого розділу 38
РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБПЛАТФОРМИ 40
3.1 Реалізація серверної логіки опрацювання замовлень 40
3.2 Реалізація передавання даних у режимі реального часу 42
3.3 Реалізація SQL-запитів до бази даних 43
3.4 Реалізація клієнтської частини, стилізація та адаптивність 44
3.5 Демонстрація роботи вебплатформи 48
3.6 Тестування вебплатформи 53
3.7 Аналіз результатів роботи 55
3.8 Висновок до третього розділу 56
РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ 58

4.1 Безпека життєдіяльності. Долікарська допомога при ураженні електричним струмом 58
4.2 Основи охорони праці. Психофізіологічне розвантаження для працівників 60
4.3 Висновок до четвертого розділу 63
ВИСНОВКИ 64
ПЕРЕЛІК ДЖЕРЕЛ 66
ДОДАТКИ

ВСТУП

Актуальність теми. Внаслідок стрімкого зростання обсягів електронної комерції та сервісів кур’єрського доставлення кінцевий споживач очікує не лише швидкого отримання товару, але й повної прозорості логістичного процесу [2]. Дослідження споживчої поведінки свідчать, що понад 90 % покупців вважають можливість відстеження замовлення ключовим чинником довіри до продавця, а відсутність актуальної інформації про місцеперебування товару є однією з головних причин звернень до служб підтримки.

Традиційні системи відстеження працюють за принципом періодичного оновлення статусів, коли інформація з’являється із затримкою у години або навіть дні. Водночас сучасні технології двостороннього зв’язку, зокрема протокол WebSocket, дозволяють транслювати зміну статусу та географічні координати кур’єра клієнтові за частки секунди.

Тому задача створення вебплатформи для відстеження замовлень товарів у режимі реального часу є актуальним напрямком сучасних досліджень у галузі розподілених інформаційних систем та інженерії даних.

Мета і задачі дослідження. Метою даної кваліфікаційної роботи освітнього рівня «Бакалавр» є підвищення якості послуг доставлення товарів шляхом розробки вебплатформи, що забезпечує миттєвий доступ клієнтів, кур’єрів та адміністраторів до актуального статусу та географічного положення замовлення. Для досягнення поставленої мети потрібно виконати ряд завдань, зокрема:

1. Проаналізувати стан досліджень у сфері відстеження замовлень та існуючі аналоги програмних рішень.

2. Дослідити технології передавання даних у режимі реального часу та обґрунтувати вибір протоколу WebSocket.

3. Спроєктувати клієнт-серверну архітектуру вебплатформи та структуру реляційної бази даних MySQL.

4. Застосувати алгоритм Дейкстри для побудови оптимального маршруту доставки доставлення.

5. Розробити серверну частину вебплатформи мовою PHP та клієнтську частину засобами JavaScript і CSS з інтегрованою колірною гамою.

6. Провести функціональне тестування вебплатформи на основних сценаріях використання.

Об’єкт дослідження – процес відстеження замовлень товарів у режимі реального часу.

Предмет дослідження – методи та засоби побудови вебплатформи для відстеження замовлень товарів у режимі реального часу на основі клієнт-серверної архітектури.

Практичне значення одержаних результатів полягає у створенні функціональної вебплатформи, яку можна впроваджувати у діяльність служб кур’єрського доставлення, закладів громадського харчування та суб’єктів електронної торгівлі малого та середнього бізнесу для автоматизації інформування клієнтів та диспетчеризації кур’єрів. Розроблена вебплатформа не потребує сплати ліцензійних відрахувань стороннім операторам, розгортається на власній інфраструктурі замовника та може бути розширена додатковими модулями аналітики.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Особливості процесу відстеження замовлень товарів

Процес доставлення товару від продавця до споживача являє собою багатоетапний логістичний ланцюг, який охоплює приймання замовлення, його комплектацію на складі, передавання кур'єрові, транспортування до місця призначення та безпосереднє вручення одержувачеві. На кожному з перелічених етапів замовлення змінює свій стан, і саме фіксація, збереження та оприлюднення цих станів становить сутність процесу відстеження [1]. Чим вищою є дискретизація фіксації станів, тим точнішою є картина руху замовлення, яку бачить кінцевий споживач.

Класична модель відстеження ґрунтується на дискретних подіях: замовлення сканується у заздалегідь визначених контрольних точках (склад відправника, сортувальний центр, проміжний хаб, відділення видачі), після чого відповідний статус фіксується у базі даних логістичного оператора. Така модель має принципову ваду – у проміжках між контрольними точками замовлення фактично «зникає» з поля зору клієнта, оскільки жодна інформація про його фактичне положення не надходить. Для міжміських та міжнародних перевезень, тривалість яких вимірюється днями, така дискретність є прийнятною, проте для кур'єрського доставлення «останньої милі», де час виконання замовлення вимірюється десятками хвилин, дискретна модель не задовольняє очікувань сучасного споживача [2].

Модель реального часу, на противагу дискретній, передбачає неперервне передавання геопозиції кур'єра з його мобільного пристрою на сервер та миттєву ретрансляцію цих даних в браузер клієнта. У такій моделі клієнт спостерігає рух кур'єра на інтерактивній мапі практично без затримки, бачить орієнтовний час прибуття та отримує миттєві сповіщення про зміну статусу. Технологічно реалізація моделі реального часу вимагає вирішення низки задач:

– забезпечення постійного двостороннього каналу зв’язку між сервером та браузером клієнта без циклічного перезавантаження сторінки;

– отримання географічних координат на стороні кур’єра за допомогою GPS-датчика мобільного пристрою та їх періодичне надсилання на сервер;

– візуалізації руху на картографічній основі з плавним переміщенням маркера;

– розмежування прав доступу, оскільки координати кур’єра є чутливими персональними даними і повинні бути доступні лише клієнтові відповідного замовлення та адміністраторові вебплатформи;

– збереження історії переміщень для подальшого аналізу якості доставлення та розв’язання спірних ситуацій.

Окремою особливістю предметної області є життєвий цикл замовлення, який доцільно формалізувати у вигляді скінченного автомата станів. Типовий життєвий цикл охоплює стани «нове», «підтверджене», «комплектується», «передане кур’єрові», «в дорозі», «доставлене» та «скасоване». Переходи між станами є строго регламентованими: наприклад, замовлення не може перейти зі стану «нове» одразу в стан «в дорозі», оминаючи призначення кур’єра. Формалізація життєвого циклу у вигляді довідникової таблиці статусів у базі даних дозволяє гнучко налаштовувати бізнес-процес без зміни програмного коду вебплатформи.

Слід також зазначити, що процес відстеження має різну інформаційну цінність для різних учасників. Для клієнта головним є прогнозованість: коли саме прибуде замовлення і де воно перебуває зараз. Для кур’єра – зручність фіксації виконаних дій мінімальною кількістю натискань. Для адміністратора – цілісна операційна картина: скільки замовлень перебуває у роботі, які кур’єри вільні, де виникають затримки. Відповідно, вебплатформа повинна надавати три різні подання тих самих даних, узгоджені між собою у реальному часі.

Деталізуємо інформаційні потреби кожної категорії користувачів. Клієнт взаємодіє з вебплатформою епізодично та короткочасно – у середньому два-три сеанси на одне замовлення тривалістю кілька хвилин кожен. Його ключові запитання: «на якому етапі моє замовлення», «де зараз кур’єр» і «коли він

прибуде». Звідси випливають вимоги до клієнтського подання: миттєве завантаження без зайвих кроків, велика наочна шкала етапів, мапа з маркером кур’єра у центрі уваги та помітний прогноз часу прибуття. Будь-яка потреба у ручному оновленні сторінки сприймається клієнтом як несправність сервісу.

Кур’єр, навпаки, користується вебплатформою безперервно протягом усієї робочої зміни, переважно зі смартфона, часто однією рукою та в русі. Його сценарій гранично утилітарний: побачити чергове призначене замовлення з адресою, підтвердити приймання одним дотиком, розпочати доставлення, після вручення – зафіксувати завершення. Передавання координат має відбуватися повністю автоматично у фоновому режимі, не відволікаючи кур’єра та не вимагаючи жодних додаткових дій. Звідси вимоги: великі елементи керування, мінімальна кількість екранів, стійкість до короткочасних розривів мобільного зв’язку.

Адміністратор працює з вебплатформою стаціонарно протягом дня та виконує диспетчерську функцію: створює замовлення за зверненнями, добирає вільного кур’єра з урахуванням його поточного положення, контролює перебіг усіх активних доставлень та втручається у виняткових ситуаціях. Для нього критичною є саме сукупна картина: спільна мапа всіх кур’єрів, перелік замовлень з кольоровим кодуванням статусів та зведені показники за день. Урахування описаних відмінностей трьох профілів користування і визначило рольову структуру вебплатформи, формалізовану далі у діаграмі прецедентів другого розділу.

Узагальнюючи, предметна область характеризується трьома визначальними властивостями: подієвістю (стани замовлення змінюються асинхронно та непередбачувано у часі), геопросторовістю (ключова інформація має координатну природу і потребує картографічної візуалізації) та багатосуб’єктністю (одні й ті самі дані одночасно споживаються кількома категоріями користувачів з різними правами). Саме ці властивості визначають архітектурні рішення, прийняті у другому розділі: подієвість зумовлює потребу у каналі проштовхування даних від сервера до браузера, геопросторовість – у

виборі точних числових типів збереження координат та картографічної бібліотеки, а багатосуб’єктність – у рольовій моделі розмежування доступу.

1.2 Аналітичний огляд існуючих аналогів відстеження замовлень

На сьогодні на ринку відомо декілька класів рішень для відстеження замовлень, кожен з яких має власну сферу застосування, переваги та обмеження. Автором проведено порівняльний аналіз найпоширеніших представників кожного класу.

Першим класом є трекінгові сервіси поштово-логістичних операторів, серед яких найвідомішими в Україні є системи відстеження «Нова пошта» та «Укрпошта», а у світовому масштабі – DHL Express, FedEx та UPS. Зазначені системи демонструють високу надійність, масштабованість та опрацьовують мільйони запитів на добу. Клієнт вводить номер товарно-транспортної накладної та отримує хронологію проходження контрольних точок. Проте всі вони працюють виключно за дискретною моделлю: геопозиція транспортного засобу у проміжках між точками сканування не транслюється, а інтервал оновлення статусів може сягати кількох годин. Крім того, ці системи є закритими і не можуть бути адаптовані під потреби окремого продавця: неможливо змінити перелік статусів, додати власну айдентику чи інтегрувати мапу руху кур’єра.

Другим класом є агрегатори відстеження, такі як AfterShip, 17TRACK та Parcel Monitor [30], які консолідують дані сотень перевізників у єдиному інтерфейсі та надають розвинений програмний інтерфейс для інтеграції [31]. Перевагою агрегаторів є універсальність: продавець, який відправляє товари кількома службами, отримує єдину точку моніторингу. Водночас агрегатор лише ретранслює дані первинних перевізників, тому повністю успадковує обмеження дискретної моделі. Тарифна модель «оплата за кожне відстежуване відправлення» робить такі рішення економічно недоцільними для малого бізнесу з великою кількістю дрібних замовлень. Окремим ризиком є передавання клієнтських даних третій стороні, що ускладнює дотримання вимог законодавства про захист персональних даних.

Третім класом є вбудовані модулі відстеження сервісів доставлення їжі та товарів першої необхідності, зокрема Glovo, Bolt Food та Uber Eats. Саме ці рішення реалізують справжній режим реального часу: клієнт спостерігає рух кур'єра на мапі із затримкою у декілька секунд, бачить прогнозований час прибуття, який динамічно перераховується, та отримує миттєві сповіщення. З технологічного погляду такі модулі є еталоном користувацького досвіду у предметній області. Проте вони є невіддільною частиною закритих екосистем: продавець, який бажає скористатися такою функціональністю, змушений підключатися до маркетплейсу, сплачувати комісію у розмірі 15–30 %, погоджуватися з нав'язаними правилами ціноутворення та втрачає прямий контроль над клієнтською базою і даними.

Четвертим, допоміжним класом є універсальні системи моніторингу транспорту. Вони забезпечують високоточне відстеження транспортних засобів, проте орієнтовані на внутрішні диспетчерські служби підприємств, не мають клієнтського подання у розрізі окремого замовлення та потребують встановлення спеціалізованих апаратних GPS-трекерів, що ускладнює і здорожчує впровадження для кур'єрів, які пересуваються пішки або велосипедом. Узагальнені результати порівняльного аналізу розглянутих аналогів подано в таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз існуючих аналогів відстеження замовлень

Критерій	**Нова пошта / DHL**	**AfterShip**	**Glovo / Uber Eats**	**Розроблювана вебплатформа**
Режим реального часу	–	–	+	+
Геопозиція кур'єра на мапі	–	–	+	+
Відкритість до адаптації	–	частково (API)	–	+

Продовження таблиці 1.1

Вартість для малого бізнесу	безоплатно	висока	комісія 15–30 %	одноразова розробка
Контроль клієнтських даних	–	–	–	+
Панель адміністратора	–	+	–	+
Потреба у спеціальному обладнанні	–	–	–	–

Як видно з таблиці 1.1, жодне з наявних рішень не поєднує одночасно трансляцію геопозиції у режимі реального часу, відкритість до адаптації та економічну доступність для малого і середнього бізнесу. Саме ця незаповнена ніша і визначає доцільність розробки власної вебплатформи.

1.3 Огляд технологій передавання даних у режимі реального часу

Ключовим технічним питанням побудови вебплатформи для відстеження замовлень товарів у режимі реального часу є вибір механізму доставлення оновлень від сервера до браузера клієнта. Стандартна модель взаємодії за протоколом HTTP є суто клієнт-ініційованою: сервер не може самостійно надіслати дані браузерові, доки той не виконає запит. Для подолання цього обмеження історично сформувалося чотири підходи [3].

Періодичне опитування (short polling) полягає у виконанні браузером запитів до сервера через фіксований інтервал часу, наприклад кожні п’ять секунд. Підхід є найпростішим у реалізації, проте характеризується найгіршим співвідношенням корисного навантаження до накладних витрат: кожен запит несе повний набір HTTP-заголовків обсягом у сотні байтів, а переважна більшість запитів повертається без нових даних. Затримка доставлення оновлення у середньому дорівнює половині інтервалу опитування.

Тривале опитування (long polling) удосконалює попередній підхід: сервер не відповідає на запит одразу, а утримує з’єднання відкритим, доки не з’являться нові дані або не спливе тайм-аут. Це суттєво зменшує затримку, проте кожне доставлене повідомлення все одно потребує повторного встановлення з’єднання, а утримання великої кількості відкритих запитів створює навантаження на сервер.

Серверні події (Server-Sent Events, SSE) [15] забезпечують односпрямований потік подій від сервера до браузера поверх звичайного HTTP-з’єднання. Технологія є простою та надійною, має вбудований механізм автоматичного перепідключення, проте канал є виключно односпрямованим: для надсилання даних від клієнта до сервера (наприклад, координат кур’єра) необхідні додаткові HTTP-запити.

Протокол WebSocket, стандартизований у документі RFC 6455, забезпечує повнодуплексний двосторонній канал зв’язку поверх єдиного TCP-з’єднання [4]. З’єднання встановлюється шляхом одноразового HTTP-рукостискання з заголовком Upgrade, після чого обмін відбувається бінарними або текстовими кадрами з заголовком розміром від 2 до 14 байтів. Це робить WebSocket найекономнішим рішенням для частого обміну невеликими повідомленнями, яким і є передавання координат. Порівняння розглянутих технологій подано в таблиці 1.2.

Для розроблюваної вебплатформи автором обрано протокол WebSocket, оскільки сценарій роботи вимагає саме двостороннього обміну: кур’єр надсилає координати на сервер тим самим каналом, яким сервер транслює оновлення клієнтові. Це дозволяє побудувати єдиний комунікаційний вузол без дублювання інфраструктури.

Вибір WebSocket накладає і додаткові безпекові зобов’язання, які враховано у вимогах до вебплатформи. По-перше, WebSocket-з’єднання не підпадає під політику однакового походження так само строго, як звичайні запити, тому сервер зобов’язаний самостійно перевіряти заголовок Origin під час рукостискання та авторизувати кожну підписку власним токеном каналу. По-друге, постійні з’єднання є привабливою ціллю для атак виснаження ресурсів,

тому сервер повинен обмежувати кількість з’єднань з однієї адреси та відхиляти кадри понад розумний розмір. По-третє, у промисловому розгортанні канал має працювати виключно поверх захищеного транспорту wss, щоб координати кур’єра не передавалися відкритим текстом через публічні мережі.

Таблиця 1.2 – Порівняння технологій передавання даних у режимі реального часу

Характеристика	**Short polling**	**Long polling**	**SSE**	**WebSocket**
Напрямок передавання	клієнт→сервер	клієнт→сервер	сервер→клієнт	двосторонній
Затримка доставлення	до інтервалу опитування	низька	низька	мінімальна
Накладні витрати на повідомлення	високі	середні	низькі	мінімальні
Кількість з’єднань	нове на кожен запит	нове на кожне повідомлення	одне постійне	одне постійне
Складність реалізації	низька	середня	низька	середня

Усі три вимоги реалізовано у третьому розділі роботи.

1.4 Обґрунтування створення вебплатформи для відстеження замовлень товарів у режимі реального часу

Проведений у параграфах 1.2 та 1.3 аналіз засвідчив наявність незаповненої ніші на ринку: відсутнє відкрите, економічно доступне рішення, яке поєднувало б трансляцію геопозиції у режимі реального часу з можливістю розгортання на власній інфраструктурі продавця. Вибір саме вебплатформи як форми реалізації обґрунтовується низкою чинників.

По-перше, кросплатформність [18]. Вебплатформа працює у будь-якому сучасному браузері на персональному комп’ютері, планшеті чи смартфоні без

необхідності встановлення [19]. Це критично важливо для клієнта, який взаємодіє з сервісом епізодично – за посиланням з повідомлення про відправлення товару, – а також для кур'єра, який може використовувати власний пристрій будь-якої платформи.

По-друге, централізоване оновлення. Нова версія вебплатформи стає доступною всім користувачам одночасно після розгортання на сервері, без процедур публікації у магазинах та очікування модерації, що пришвидшує цикл випуску виправлень.

По-третє, зрілість екосистеми. Обраний технологічний стек – PHP, MySQL та JavaScript – є найпоширенішим у сегменті малого та середнього бізнесу: за даними W3Techs [5], мова PHP використовується на серверній стороні понад 74 % усіх вебпроєктів, для яких відома серверна технологія [6]. Це гарантує доступність хостингу, фахівців для супроводу та великої кількості перевірених бібліотек [ЗЗ].

По-четверте, низька вартість володіння. Усі компоненти стеку є вільно поширюваними з відкритим вихідним кодом, а мінімальні апаратні вимоги дозволяють розгорнути вебплатформу на віртуальному сервері початкового рівня.

Окремого обґрунтування потребує вибір серверної мови, оскільки для систем реального часу історично частіше застосовується платформа Node.js з її природно асинхронною моделлю. Проведено порівняння двох кандидатів стосовно специфіки розроблюваної вебплатформи, результати якого подано в таблиці 1.3.

Як видно з таблиці 1.3, платформа Node.js має перевагу лише у природності підтримки WebSocket, тоді як за рештою критеріїв – зрілістю інструментів роботи з базою даних, доступністю розгортання, вбудованими механізмами безпеки та вартістю супроводу – для задач малого і середнього бізнесу доцільнішим є стек PHP. Єдиний недолік компенсується бібліотекою Ratchet, яка реалізує подієвий цикл засобами ReactPHP і виносить опрацювання WebSocket-з'єднань в окремий довготривалий процес.

Таблиця 1.3 – Порівняння серверних платформ для реалізації вебплатформи

Критерій	**PHP 8.3 + Laravel + Ratchet**	**Node.js + Express + ws**
Модель виконання HTTP-частини	процес на запит (ізоляція збоїв)	єдиний подієвий цикл
Підтримка WebSocket	окремий процес Ratchet / ReactPHP	вбудована, природна
Зрілість ORM та міграцій БД	Eloquent, вбудовані міграції	Sequelize / Prisma (окремі пакети)
Доступність недорогого хостингу	максимальна	обмежена (потрібен VPS)
Поріг входження та супровід	низький, велика спільнота	середній
Безпека «з коробки»	CSRF, валідація, хешування у складі Laravel	збирається з окремих пакетів

Таким чином, обраний стек поєднує переваги обох підходів: класичну надійну модель опрацювання транзакційних запитів та асинхронну модель для потокового каналу.

1.5 Висновок до першого розділу

В першому розділі кваліфікаційної роботи досліджено особливості процесу відстеження замовлень товарів та виявлено принципову відмінність дискретної моделі контрольних точок від моделі реального часу. Проаналізовано чотири класи існуючих аналогів та встановлено, що жоден з них не поєднує трансляцію геопозиції у реальному часі з відкритістю та економічною доступністю для малого бізнесу. Розглянуто чотири технології доставлення оновлень у браузер та обґрунтовано вибір протоколу WebSocket як єдиного двостороннього каналу обміну. Сформульовано постановку задачі на розробку вебплатформи з трьома ролями користувачів, визначено функціональні та нефункціональні вимоги і технологічний стек реалізації.

РОЗДІЛ 2. ПРОЄКТНА ЧАСТИНА ТА АРХІТЕКТУРНІ РІШЕННЯ

2.1 Клієнт-серверна архітектура та потік опрацювання запиту

Розроблювана вебплатформа побудована за класичною трирівневою клієнт-серверною архітектурою, доповненою окремим вузлом обміну даними у режимі реального часу [29]. Перший рівень – рівень подання – реалізується у браузері користувача та охоплює три клієнтські подання: кабінет клієнта, робоче місце кур’єра та панель адміністратора. Другий рівень – рівень бізнес-логіки – реалізується серверною частиною мовою PHP за архітектурним шаблоном MVC: маршрутизатор спрямовує HTTP-запити до контролерів, контролери звертаються до моделей, а результат повертається у вигляді HTML-сторінок або JSON-відповідей. Третій рівень – рівень даних – представлений СУБД MySQL, яка забезпечує збереження, цілісність та конкурентний доступ до даних [6].

Принциповою архітектурною особливістю вебплатформи є розділення двох типів взаємодії за різними каналами зв'язку. Транзакційні операції – автентифікація, створення замовлення, призначення кур'єра, зміна статусу – виконуються за класичною схемою «запит–відповідь» протоколом HTTP, оскільки вони є відносно рідкісними та потребують повного циклу перевірки прав і валідації вхідних даних. Потокові операції – передавання географічних координат кур'єра та трансляція змін статусів клієнтові – виконуються через постійне з'єднання за протоколом WebSocket, оскільки вони є частими, легкими за обсягом та критично чутливими до затримки. Таке розділення дозволяє кожному каналу працювати в оптимальному для нього режимі та масштабуватися незалежно один від одного.

WebSocket-сервер реалізовано як окремий довготривалий процес на основі бібліотеки Ratchet [11], який слухає виділений TCP-порт 8080. Він підтримує реєстр активних з’єднань, згрупованих за ідентифікаторами замовлень: клієнт після відкриття сторінки відстеження надсилає повідомлення підписки на канал свого замовлення, а кур’єр – повідомлення з координатами. Отримавши координати, сервер перевіряє повноваження відправника, зберігає точку

траєкторії у базі даних та ретранслює її всім підписникам відповідного каналу. Завдяки спільній базі даних HTTP-частина та WebSocket-частина залишаються узгодженими: зміна статусу, виконана через HTTP-контролер, фіксується у базі даних та одночасно публікується у відповідний канал реального часу.

Прийняте розділення каналів має і кількісне обґрунтування. За добу типове замовлення породжує близько десяти транзакційних запитів та, за середньої тривалості доставлення тридцять хвилин і періоду надсилання чотири секунди, близько 450 потокових повідомлень з координатами. Якби обидва типи навантаження обслуговувалися спільним пулом процесів HTTP-сервера, рідкісні, але важкі транзакційні запити конкурували б за ресурси з безперервним потоком легких повідомлень. Виділення потокового навантаження в окремий однопотоковий подієвий процес дозволяє HTTP-частині зберігати звичну модель «процес на запит», а WebSocket-частині – утримувати тисячі одночасних з'єднань за мінімального споживання пам'яті.

Архітектура вебплатформи (рисунок 2.1) відображає взаємодію трьох рівнів. На клієнтському рівні працюють три браузерні подання – клієнта, кур'єра й адміністратора, – кожне з яких звертається до сервера двома незалежними каналами. Транзакційні дії (вхід, створення замовлення, зміна статусу) спрямовуються до HTTP-сервера, який через шар контролерів і моделей звертається до бази даних і повертає результат. Потокові дані – підписка на оновлення, координати кур'єра, миттєві статуси – проходять через окремий WebSocket-сервер, що утримує постійні з'єднання та групує їх у канали за замовленнями. Обидва серверні вузли працюють зі спільною базою даних MySQL, завдяки чому стан системи лишається узгодженим незалежно від того, яким каналом надійшла зміна. Окремою є стрілка від HTTP-сервера до WebSocket-сервера: коли статус змінюється транзакційним запитом, HTTP-частина публікує подію, і WebSocket-сервер негайно транслює її всім підписаним клієнтам. Така побудова забезпечує слабку зв'язність: відмова потокового вузла не блокує транзакційних операцій, а клієнт у цьому разі переходить у резервний режим опитування.

Структурну схему архітектури вебплатформи для відстеження замовлень товарів, можна побачити на рисунку 2.1.

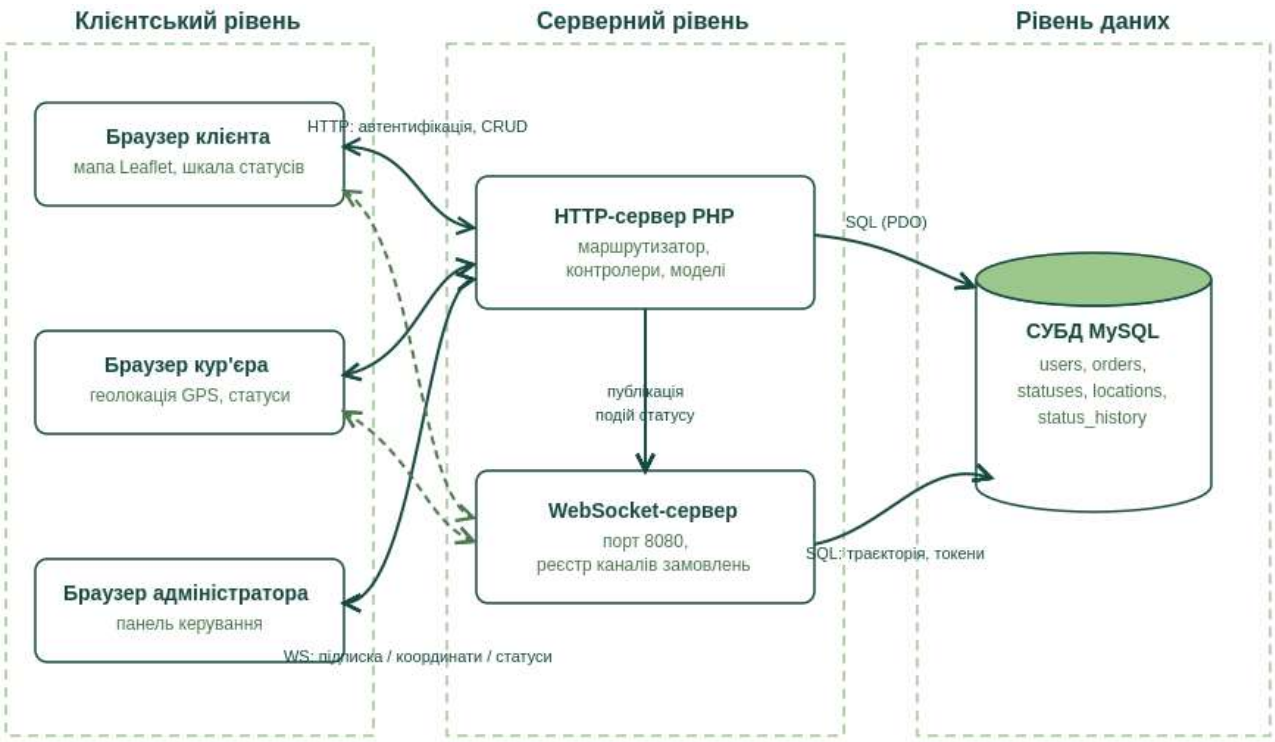

Рисунок 2.1 – Структурна схема архітектури вебплатформи

Наведена схема демонструє слабку зв’язність компонентів: відмова WebSocket-сервера не блокує транзакційні операції, а клієнтська частина у такому разі автоматично переходить у резервний режим періодичного опитування.

Для деталізації внутрішньої організації серверної частини доцільно простежити проходження транзакційного запиту крізь шари шаблону MVC на прикладі зміни статусу замовлення. Запит спершу потрапляє до єдиної точки входу – фронтального контролера, який ініціалізує середовище та визначає маршрут. Далі запит послідовно проходить ланцюжок перевірок: відновлення сесії та автентифікація користувача, перевірка відповідності ролі переліку дозволених для операції, перевірка захисного токена для запобігання міжсайтовій підробці запитів. Лише пройшовши всі перевірки, запит делегується методу контролера, який звертається до моделей предметної області, а ті транслюють об'єктні операції у параметризовані SQL-запити. Сформована відповідь повертається тим самим ланцюжком у зворотному порядку. Така

багатошаровість гарантує, що жоден запит не дістанеться бізнес-логіки, оминувши перевірки автентичності, повноважень і коректності даних. Процес опрацювання запиту за шаблоном MVC можна побачити на рисунку 2.2

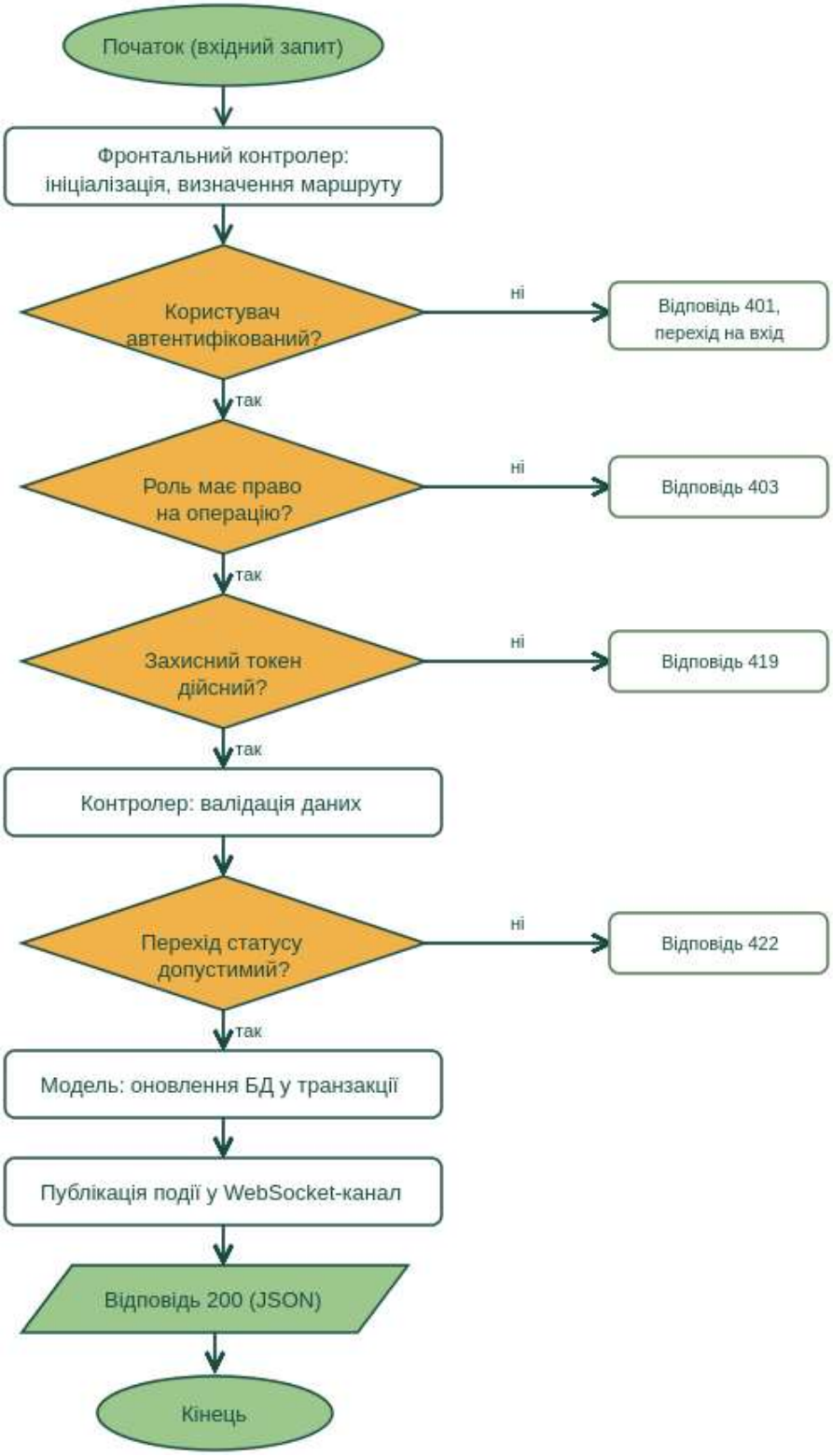

Рисунок 2.2 – Блок-схема опрацювання запиту зміни статусу за шаблоном MVC

Блок-схема опрацювання запиту показує, як запит на зміну статусу послідовно проходить захисні бар'єри, перш ніж змінити дані. Спершу фронтальний контролер визначає маршрут, після чого виконуються три послідовні перевірки: чи автентифікований користувач, чи має його роль право на цю операцію та чи дійсний захисний токен проти міжсайтової підробки. Невдача на будь-якому етапі негайно перериває опрацювання й повертає відповідний код помилки протоколу HTTP (401, 403 або 419), не допускаючи запит до бізнес-логіки. Якщо всі перевірки пройдено, контролер валідує вхідні дані та звіряє бажаний перехід зі скінченним автоматом статусів; неприпустимий перехід відхиляється з кодом 422. Лише після цього модель оновлює базу даних у межах транзакції та публікує подію про зміну в реальному часі, а користувач отримує успішну відповідь. Схема унаочнює ключовий принцип: жоден запит не досягає рівня даних, оминувши перевірки прав і коректності.

2.2 Діаграма прецедентів вебплатформи

Для формалізації функціональних вимог побудовано діаграму прецедентів, яка визначає межі вебплатформи та варіанти її використання трьома акторами [7]. Дану діаграму можна побачити на рисунку 2.3.

Діаграма прецедентів засвідчує, що спільним для всіх акторів є лише прецедент авторизації, тоді як решта функціональності строго розмежована за ролями, що відповідає принципу мінімально необхідних привілеїв.

Розглянемо специфікації двох ключових прецедентів. Прецедент «Відстежити замовлення на мапі»: головним актором є Клієнт; передумова – клієнт авторизований і має принаймні одне замовлення зі статусом, відмінним від «доставлене» чи «скасоване». Основний потік: клієнт обирає замовлення з переліку; вебплатформа повертає сторінку відстеження з поточним статусом, побудованим маршрутом і токеном каналу; браузер встановлює WebSocket-з’єднання та підписується на канал замовлення; у міру надходження координат маркер кур’єра переміщується мапою, а за зміни статусу оновлюється шкала етапів; постумова – клієнт спостерігає актуальний стан без додаткових дій.

Альтернативні потоки: якщо кур’єра ще не призначено, мапа показує лише пункти відправлення і призначення з відповідним повідомленням; якщо WebSocket-з’єднання неможливе, клієнтська частина переходить у резервний режим опитування з інтервалом десять секунд.

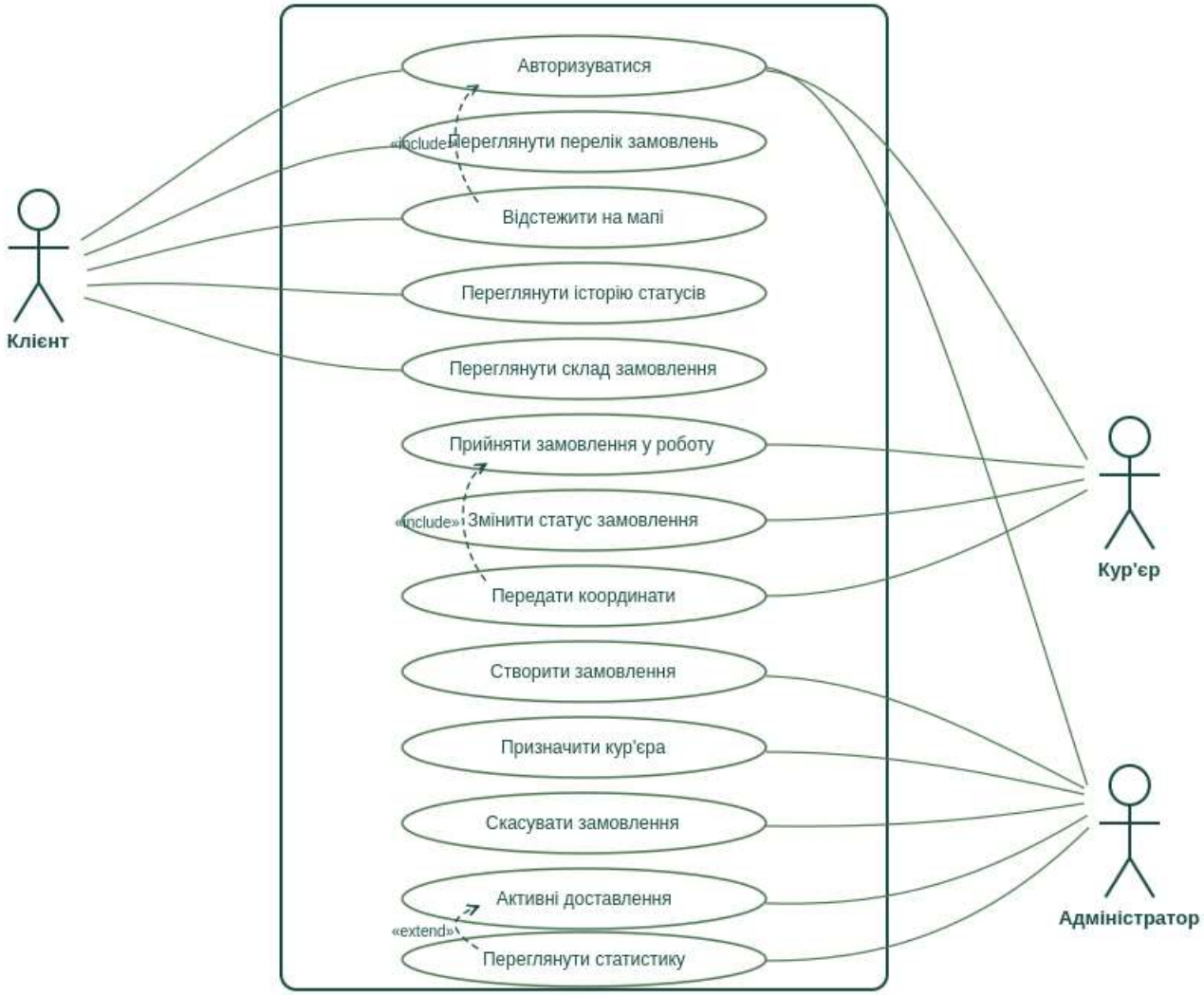

Рисунок 2.3 – Діграма прецедентів

Прецедент «Змінити статус замовлення»: головним актором є Кур’єр (адміністратор має ті самі повноваження). Передумови – кур’єр авторизований, замовлення призначене саме йому. Основний потік: кур’єр натискає кнопку наступного етапу; клієнтська частина надсилає запит зміни статусу; сервер перевіряє належність замовлення та допустимість переходу за скінченним автоматом, у транзакції оновлює замовлення і журнал історії та публікує подію у канал; усі підписники миттєво бачать новий статус. Альтернативні потоки: за спроби недопустимого переходу сервер повертає помилку валідації з поясненням

і стан замовлення не змінюється; за втрати мережі запит повторюється автоматично після відновлення з’єднання, а ідемпотентність переходу гарантує відсутність подвійного застосування.

2.3 Проєктування бази даних MySQL

Інформаційним ядром вебплатформи є реляційна база даних під керуванням СУБД MySQL 8.0 з рушієм збереження InnoDB, який забезпечує підтримку транзакцій, зовнішніх ключів та блокування на рівні рядків [8]. Проєктування виконано методом «сутність–зв’язок» з подальшою нормалізацією відношень до третьої нормальної форми, що виключає дублювання даних та аномалії оновлення [27].

У предметній області виділено п'ять сутностей: користувач (users), статус (statuses), замовлення (orders), точка геолокації (locations) та запис історії статусів (status_history). Сутність users об'єднує всіх учасників процесу, а їхня роль визначається атрибутом role переліченого типу, що спрощує автентифікацію та дозволяє одному механізму входу обслуговувати три подання. Сутність statuses реалізовано як довідник, що дозволяє адміністраторові змінювати перелік етапів життєвого циклу без модифікації програмного коду. ER-діаграму вебплатформи для відстеження замовлень товарів в режимі реального часу можна побачити на рисунку 2.4

ER-діаграма показує, як інформація про замовлення розподілена між нормалізованими таблицями та пов'язана зовнішніми ключами. Центральною є сутність orders, навколо якої будуються всі зв'язки. Кожне замовлення посилається на двох користувачів із таблиці users – клієнта (обов'язково) та кур'єра (необов'язково, доки замовлення не призначене), що відображають два окремі зв'язки «розміщує» і «виконує». Поточний стан замовлення визначає посилання на довідник statuses, винесення якого в окрему таблицю дозволяє керувати переліком етапів без зміни коду. Накопичення геопозицій реалізує зв'язок «один-до-багатьох» із таблицею locations: одне замовлення має багато точок траєкторії, кожна з яких також зберігає кур'єра-відправника. Журнал

status_history фіксує кожен перехід життєвого циклу, посилаючись водночас на замовлення, новий статус і користувача, що виконав зміну. Така структура усуває дублювання даних: будь-який факт зберігається рівно в одному місці, а цілісність зв'язків гарантують обмеження зовнішніх ключів.

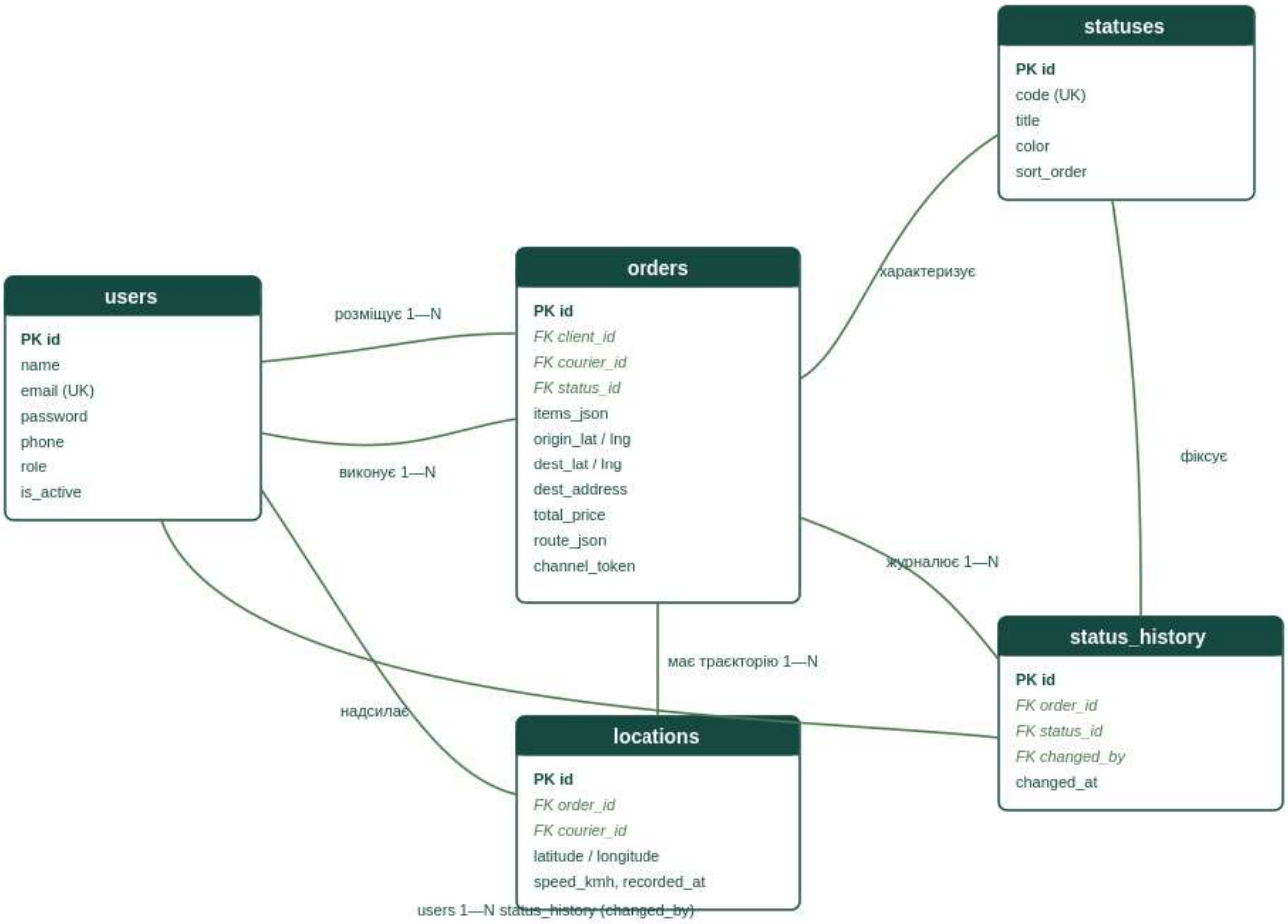

Рисунок 2.4 – ER-діаграма вебплатформи для відстеження замовлень товарів в режимі реального часу

Проілюструємо застосовану нормалізацію на прикладі статусу замовлення. У ненормалізованому варіанті назва статусу та його колір зберігалися б текстом безпосередньо в кожному рядку таблиці orders, що породжувало б трикратне дублювання («В дорозі», «#164A41» у тисячах рядків), ризик розбіжного написання та неможливість перейменувати етап однією операцією. Винесення статусів у окремий довідник з посиланням за сурогатним ключем усуває залежність неключових атрибутів назви і кольору від неключового атрибута коду в межах orders, тобто приводить схему до третьої нормальної форми: тепер

перейменування етапу чи зміна його кольору в інтерфейсі є оновленням одного рядка довідника. Аналогічно атрибути кур’єра (ім’я, телефон) не дублюються у замовленнях, а досягаються з’єднанням з таблицею users за зовнішнім ключем. Структуру таблиці users подано в таблиці 2.1.

Таблиця 2.1 – Структура таблиці users

Поле	**Тип даних**	**Обмеження**	**Призначення**
id	BIGINT UNSIGNED	PK, AUTO_INCREMENT	унікальний ідентифікатор
name	VARCHAR(100)	NOT NULL	ім’я та прізвище користувача
email	VARCHAR(150)	NOT NULL, UNIQUE	електронна пошта (логін)
password	VARCHAR(255)	NOT NULL	хеш пароля (bcrypt)
phone	VARCHAR(20)	NULL	контактний телефон
role	ENUM(’client’,’courier’,’admin’)	NOT NULL, DEFAULT ’client’	роль користувача
is_active	TINYINT(1)	NOT NULL, DEFAULT 1	ознака активності облікового запису
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	момент реєстрації

Сутність statuses реалізовано як довідник, що дозволяє адміністраторові змінювати перелік етапів життєвого циклу без модифікації коду. Структуру таблиці statuses подано в таблиці 2.2.

Центральною сутністю є orders, яка пов’язує клієнта, кур’єра, поточний статус та координати пунктів відправлення і призначення. Для географічних координат обрано тип DECIMAL(10,7), який гарантує точність позиціонування близько одного сантиметра та, на відміну від типів з рухомою комою, не накопичує похибок округлення. Структуру таблиці orders подано в таблиці 2.3.

Таблиця 2.2 – Структура таблиці statuses

Поле	Тип даних	Обмеження	Призначення
id	TINYINT UNSIGNED	PK, AUTO_INCREMENT	ідентифікатор статусу
code	VARCHAR(30)	NOT NULL, UNIQUE	машинний код (new, in_transit)
title	VARCHAR(60)	NOT NULL	назва для відображення
color	CHAR(7)	NOT NULL	колір індикатора у форматі HEX
sort_order	TINYINT UNSIGNED	NOT NULL	порядок у шкалі етапів

Таблиця 2.3 – Структура таблиці orders

Поле	Тип даних	Обмеження	Призначення
id	BIGINT UNSIGNED	PK, AUTO_INCREMENT	номер замовлення
client_id	BIGINT UNSIGNED	FK → users.id, NOT NULL	замовник
courier_id	BIGINT UNSIGNED	FK → users.id, NULL	призначений кур’єр
status_id	TINYINT UNSIGNED	FK → statuses.id, NOT NULL	поточний статус
items_json	JSON	NOT NULL	склад замовлення
origin_lat	DECIMAL(10,7)	NOT NULL	широта пункту відправлення
origin_lng	DECIMAL(10,7)	NOT NULL	довгота пункту відправлення
dest_lat	DECIMAL(10,7)	NOT NULL	широта пункту призначення
dest_lng	DECIMAL(10,7)	NOT NULL	довгота пункту призначення
dest_address	VARCHAR(255)	NOT NULL	адреса доставлення
total_price	DECIMAL(10,2)	NOT NULL	вартість замовлення, грн
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	момент створення
delivered_at	TIMESTAMP	NULL	момент вручення

Сутність locations накопичує траєкторію руху: кожен запис є однією точкою, надісланою кур’єром. Оскільки таблиця є найінтенсивнішою за записом (одна точка кожні 3–5 секунд від кожного активного кур’єра), для неї передбачено складений індекс за полями (order_id, recorded_at), який забезпечує швидке вибирання траєкторії конкретного замовлення у хронологічному порядку. Структуру таблиці locations подано в таблиці 2.4.

Додатково запроваджено журнальну таблицю status_history (поля id, order_id, status_id, changed_by, changed_at), яка фіксує кожен перехід життєвого циклу та дозволяє відновити повну хронологію замовлення для розв’язання спірних ситуацій. Код створення даних таблиць можна побачити в Додатку А.

Таблиця 2.4 – Структура таблиці locations

Поле	**Тип даних**	**Обмеження**	**Призначення**
id	BIGINT UNSIGNED	PK, AUTO_INCREMENT	ідентифікатор точки
order_id	BIGINT UNSIGNED	FK → orders.id, NOT NULL	замовлення
courier_id	BIGINT UNSIGNED	FK → users.id, NOT NULL	кур’єр-відправник
latitude	DECIMAL(10,7)	NOT NULL	широта точки
longitude	DECIMAL(10,7)	NOT NULL	довгота точки
speed_kmh	DECIMAL(5,2)	NULL	миттєва швидкість, км/год
recorded_at	TIMESTAMP(3)	NOT NULL	момент фіксації (мс)

Індексну стратегію підпорядковано фактичним шаблонам запитів. Складений індекс (order_id, recorded_at) таблиці locations покриває головний запит сторінки відстеження – вибірку траєкторії конкретного замовлення у хронологічному порядку – і дозволяє виконувати його без сортування. Індекс (client_id, created_at) таблиці orders прискорює побудову кабінету клієнта з посортованим переліком замовлень, а індекс (courier_id, status_id) – формування робочого списку кур’єра, де відбираються лише незавершені доставлення.

Унікальний індекс поля email таблиці users одночасно гарантує цілісність та забезпечує миттєвий пошук облікового запису під час автентифікації. Від надлишкових індексів свідомо відмовлено, оскільки кожен додатковий індекс уповільнює інтенсивне вставляння у таблицю locations.

2.4 Побудова оптимального маршруту та взаємодія компонентів у реальному часі

Окрім відстеження, вебплатформа розв'язує допоміжну задачу – побудову найкоротшого маршруту кур'єра від пункту відправлення до пункту призначення дорожньою мережею міста. Дорожню мережу подають у вигляді зваженого орієнтованого графа $G = (V, E)$, де множина вершин V відповідає перехрестям, множина ребер E – ділянкам доріг, а вага ребра $W(u, v)$ – метричній довжині ділянки. Класичним методом пошуку найкоротших шляхів від однієї вершини у графі з невід'ємними вагами є алгоритм Дейкстри [9].

Алгоритм підтримує для кожної вершини v оцінку відстані d[v] від початкової вершини s. На кожній ітерації обирається неопрацьована вершина u з мінімальним d[u], після чого для кожного суміжного ребра виконується операція релаксації за формулою (2.1):

$$d[v] = \min\big(d[v],\ d[u] + w(u, v)\big) \text{ (2.1)}, \qquad (2.1)$$

де d[v] – поточна оцінка найкоротшої відстані до вершини v; d[u] – остаточна відстань до щойно опрацьованої вершини u; w(u, v) – вага ребра між вершинами u та v.

При реалізації черги з пріоритетами на основі двійкової купи обчислювальна складність становить O((|V| + |E|)·log|V|), що для дорожнього графа середнього міста забезпечує час побудови маршруту у межах десятків мілісекунд.

Блок-схема алгоритму Дейкстри описує, як визначається найкоротший маршрут кур'єра. Спочатку відстані до всіх вершин графа вважаються

нескінченними, а до початкової – нульовою, після чого вершини опрацьовуються у порядку зростання відстані за допомогою черги з пріоритетами. Дану блок-схему можна побачити на рисунку 2.5.

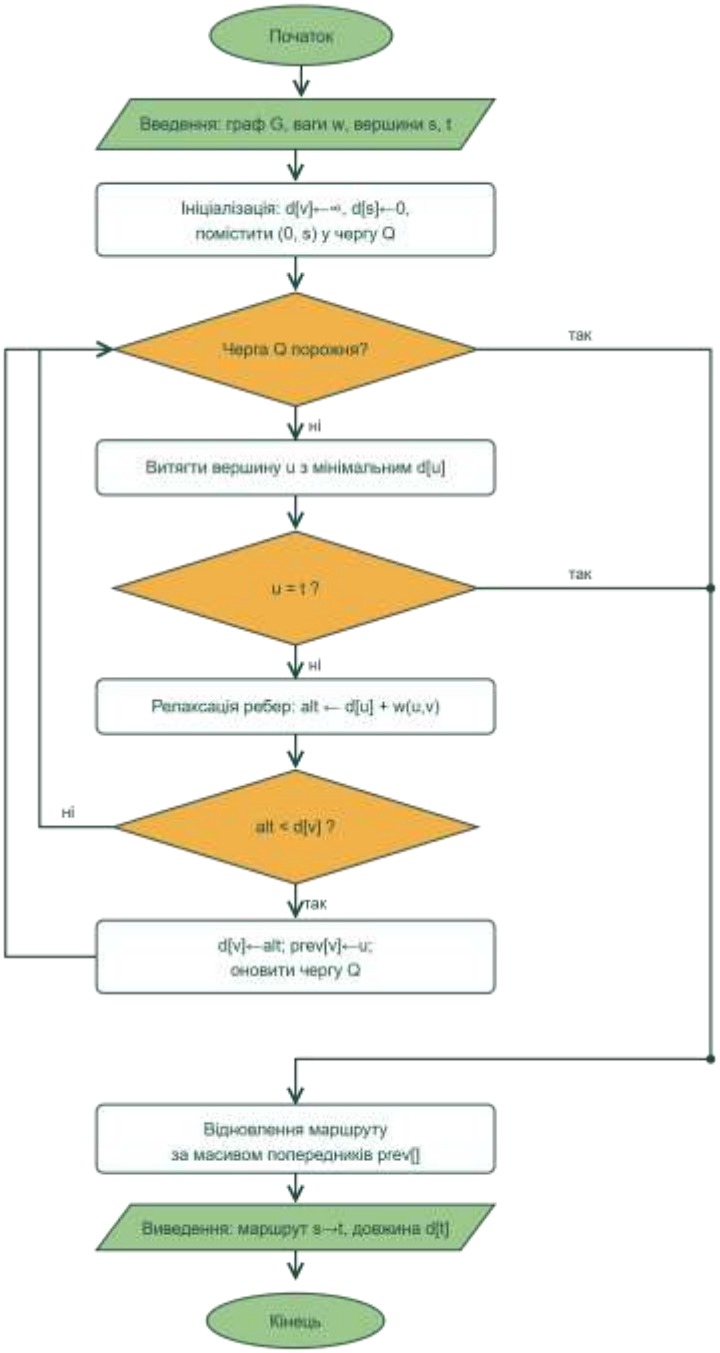

Рисунок 2.5 – Блок-схема алгоритму Дейкстри для побудови оптимального маршруту

На кожній ітерації алгоритм бере найближчу ще не опрацьовану вершину й перевіряє, чи це пункт призначення; якщо так – пошук завершується

достроково. Інакше для кожного сусіда виконується релаксація: якщо шлях через поточну вершину коротший за вже відомий, оцінку відстані оновлюють і запам'ятовують попередника. Цикл повторюється, доки черга не спорожніє або не буде досягнуто мети. Наприкінці маршрут відновлюють зворотним проходом за збереженими попередниками від кінцевої вершини до початкової. Завдяки опрацюванню вершин строго в порядку зростання відстані алгоритм гарантує оптимальність знайденого шляху за один прохід.

Побудований маршрут зберігається разом із замовленням у полі route_json та використовується клієнтською частиною двояко: на карті відображається лінія очікуваного маршруту від складу до адреси доставлення, а у міру надходження координат кур'єра на цій лінії рухається маркер, наочно показуючи, яку частину шляху вже пройдено і де замовлення перебуває зараз відносно повного маршруту.

Діаграма послідовності демонструє повний обмін повідомленнями під час доставлення в реальному часі. Дану діаграму можна побачити на рисунку 2.6. Спочатку браузер клієнта запитує сторінку відстеження в HTTP-сервера й отримує разом із нею токен доступу до каналу та побудований маршрут. Із цим токеном клієнт встановлює WebSocket-з'єднання та підписується на канал свого замовлення, а сервер, перевіривши токен, додає його до групи підписників. Далі починається циклічна фаза: кур'єр кожні кілька секунд надсилає координати, сервер зберігає кожну точку в базі та негайно ретранслює її клієнтові, у якого відповідно зміщується маркер на мапі. Коли кур'єр завершує доставлення, зміна статусу йде вже транзакційним HTTP-каналом; сервер фіксує її в базі та публікує подію у WebSocket-сервер, який транслює оновлення клієнтові – той миттєво бачить новий етап на шкалі й отримує сповіщення.

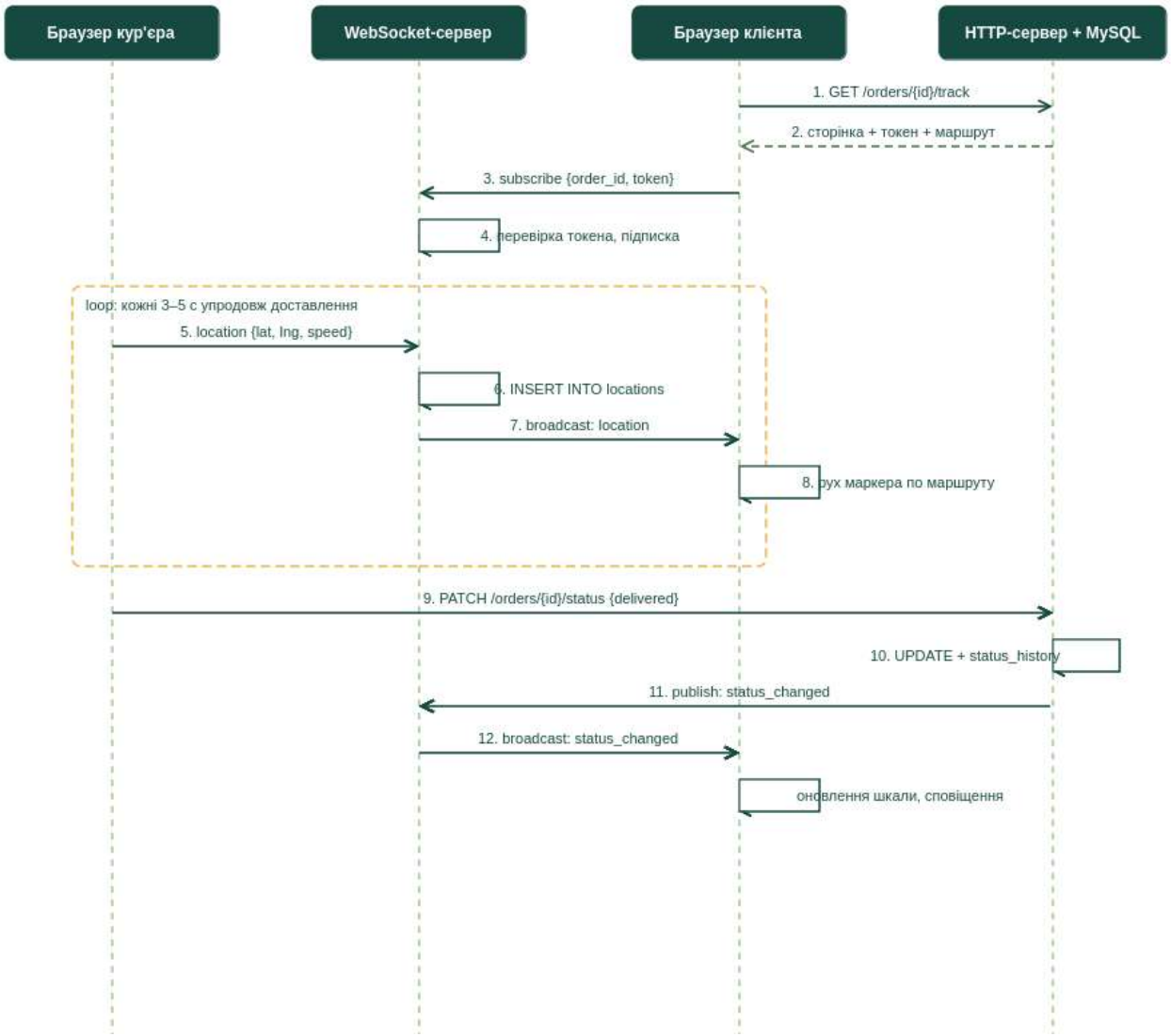

Рисунок 2.6 – Діаграма послідовності взаємодії компонентів

Діаграма наочно показує головну властивість системи: усі оновлення доставляються клієнтові проактивно, без жодного запиту з його боку.

2.5 Програмна структура, життєвий цикл та захист даних

Об'єктну структуру серверної частини вебплатформи спроєктовано за принципами розподілу відповідальності: моделі інкапсулюють доступ до даних, сервіси – складну бізнес-логіку, контролери – обробку запитів. Це відповідає вимогам супроводжуваності та дозволяє незалежно змінювати кожен шар [33].

Діаграма класів відображає об'єктну організацію серверної частини та розподіл відповідальності між компонентами. Дану діаграму можна побачити на рисунку 2.7.

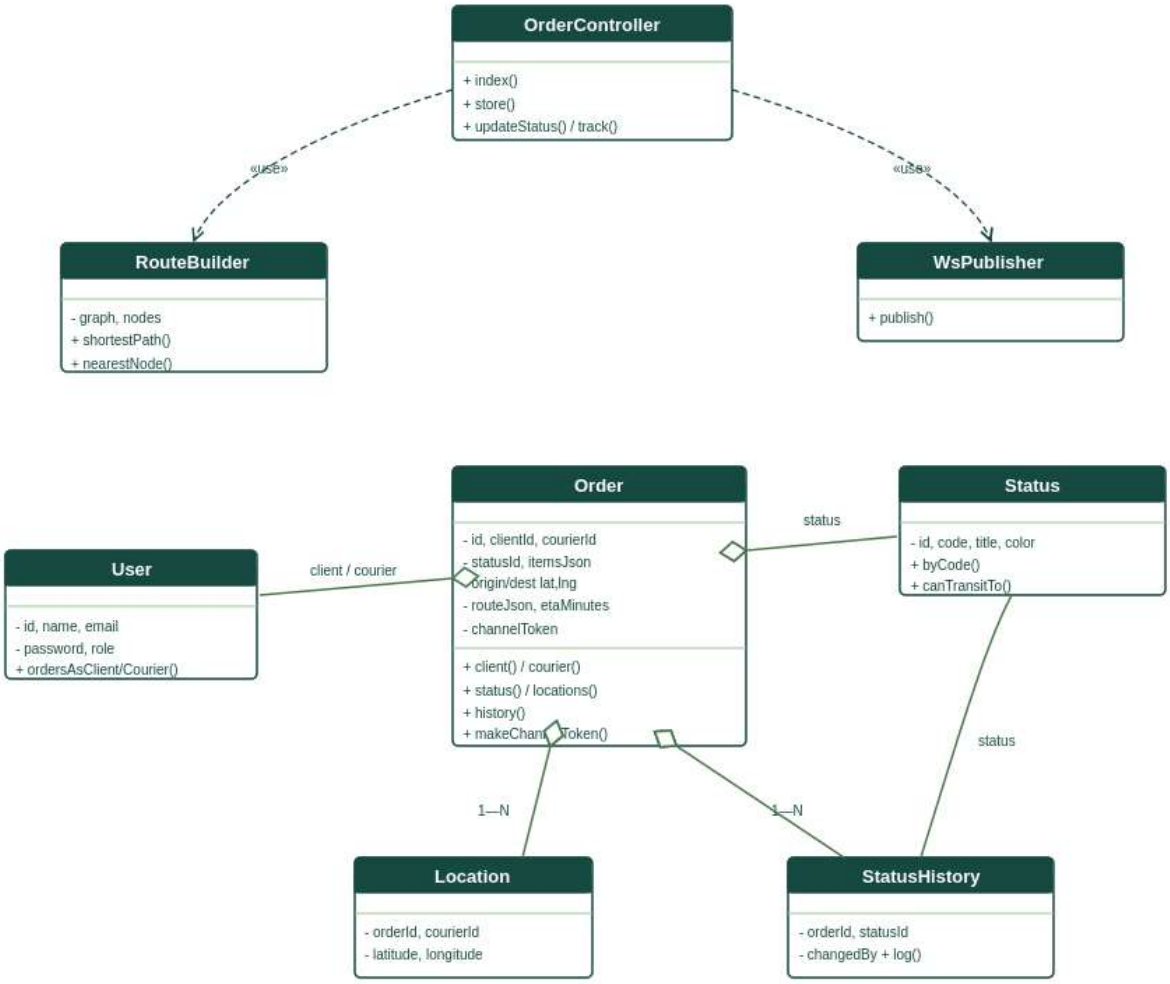

Рисунок 2.7 – Діаграма класів для вебплатформи відстеження товарів в режимі реального часу

Контролер OrderController виступає координатором: він приймає запити й делегує роботу спеціалізованим класам, від яких залежить, – RouteBuilder для побудови маршруту та WsPublisher для трансляції подій. Доступ до даних інкапсульовано в класах-моделях, центральною з яких є Order: вона зберігає всі атрибути замовлення й через свої методи надає пов'язані об'єкти – клієнта та кур'єра (обидва з класу User), поточний статус (Status), точки траєкторії (Location) та журнал переходів (StatusHistory). Клас Status, окрім довідкових даних, інкапсулює правила переходів життєвого циклу через метод canTransitTo, завдяки чому бізнес-логіка станів зосереджена в одному місці. Такий поділ – моделі для даних, сервіси для складних операцій, контролер для координації – забезпечує слабку зв'язність: кожен клас можна змінювати чи тестувати незалежно від решти.

Життєвий цикл замовлення формалізовано у вигляді скінченного автомата станів, реалізованого як таблиця дозволених переходів. Така формалізація унеможливлює некоректні переходи незалежно від того, з якого подання надійшов запит.

Діаграма станів описує життєвий цикл замовлення як скінченний автомат із чітко визначеними переходами. Дану діаграму можна побачити на рисунку 2.8.

Рисунок 2.8 – Діграма станів життєвого циклу замовлення в вебплатформі для відстеження товарів

Замовлення починає шлях у стані «Нове» й послідовно проходить підтвердження, комплектацію, передавання кур'єрові, доставлення та вручення, причому кожен перехід ініціюється конкретною подією – підтвердженням

адміністратора, призначенням кур'єра, початком доставлення або врученням. Паралельно до основної послідовності з будь-якого проміжного стану можливий перехід у стан «Скасоване», що дозволяє коректно припинити обробку у виняткових ситуаціях. Стани «Доставлене» та «Скасоване» є кінцевими – з них жодних переходів немає. Формалізація у вигляді автомата гарантує, що неможливі некоректні стрибки (наприклад, зі стану «Нове» одразу в «Доставлене»): дозволені лише ті переходи, які явно зображені стрілками, і ця сама таблиця переходів використовується в коді для перевірки кожної зміни статусу.

Захист даних реалізовано на кількох рівнях відповідно до рекомендацій OWASP [30, 32, 34]. Паролі користувачів ніколи не зберігаються у відкритому вигляді – застосовано однобічне хешування алгоритмом bcrypt з автоматичним додаванням солі, тож навіть за компрометації бази даних відновити вихідні паролі неможливо. Усі звернення до бази даних виконуються через підготовлені параметризовані вирази, у яких дані користувача передаються окремо від тексту запиту, що унеможливлює SQL-ін'єкції. Кожна операція захищена перевіркою ролі: проміжний обробник звіряє роль автентифікованого користувача з переліком дозволених для маршруту, а доступ до координат замовлення надається лише його власникові та адміністраторові. Підписка на WebSocket-канал авторизується одноразовим токеном, який звіряється методом hash_equals, стійким до атак за часом виконання, а внутрішні службові повідомлення захищено окремим серверним ключем. Транзакційні запити, що змінюють стан, захищено токеном проти міжсайтової підробки запитів (CSRF), а вихідні дані екрануються перед виведенням у HTML, що запобігає міжсайтовому виконанню сценаріїв (XSS).

Оскільки база даних є єдиним джерелом критичної інформації про замовлення та траєкторії, передбачено стратегію регулярного резервного копіювання. Резервні копії створюються логічним вивантаженням засобом mysqldump, який формує самодостатній SQL-скрипт повного відтворення структури та даних. Копії доцільно створювати за розкладом – щодобове повне копіювання у нічний період мінімального навантаження – із зберіганням

останніх копій за принципом ротації. Відновлення виконується зворотним застосуванням збереженого скрипту до чистого екземпляра СУБД. Команди формування резервної копії та відновлення винесено у відповідний скрипт, наведений у додатках.

2.6 Висновок до другого розділу

В другому розділі кваліфікаційної роботи виконано повне проєктування вебплатформи для відстеження замовлень товарів у режимі реального часу.

Спроєктовано трирівневу клієнт-серверну архітектуру з принциповим розділенням транзакційного HTTP-каналу та потокового WebSocket-каналу, що дозволяє кожному типу взаємодії працювати в оптимальному режимі та масштабуватися незалежно. Розділення підкріплено кількісною оцінкою навантаження, а проходження запиту крізь шари шаблону MVC деталізовано блок-схемою опрацювання з усіма рівнями перевірок прав і даних.

Формалізовано функціональні вимоги діаграмою прецедентів з трьома акторами та тринадцятьма варіантами використання. Спроєктовано нормалізовану до третьої нормальної форми реляційну базу даних з п'яти таблиць, обґрунтовано вибір типів даних для географічних координат і складу замовлення, побудовано ER-діаграму та визначено індексну стратегію за фактичними шаблонами запитів.

Досліджено алгоритм Дейкстри для побудови оптимального маршруту кур'єра, що дозволяє відображати на карті повний маршрут доставлення з позначенням поточного положення замовлення, а динаміку взаємодії компонентів у реальному часі унаочнено діаграмою послідовності наскрізного сценарію.

Розроблено об'єктну структуру серверної частини у вигляді діаграми класів з розподілом відповідальності між моделями, сервісами та контролерами, а життєвий цикл замовлення формалізовано діаграмою станів скінченного автомата, що унеможливлює некоректні переходи статусів. Окрему увагу приділено нефункціональним аспектам: спроєктовано багаторівневий захист

даних (хешування паролів, параметризовані запити, розмежування доступу за ролями, токенна авторизація каналів, захист від CSRF та XSS) і стратегію регулярного резервного копіювання бази даних із можливістю відновлення. Прийняті проєктні рішення створюють основу для практичної реалізації, описаної у третьому розділі.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБПЛАТФОРМИ

3.1 Реалізація серверної логіки опрацювання замовлень

Серверну частину вебплатформи реалізовано мовою PHP [12] версії 8.3 на основі фреймворку Laravel 11 [10] за архітектурним шаблоном «модель–подання–контролер» [13]. Центральним компонентом є контролер опрацювання замовлень, який реалізує чотири функції: формування переліку замовлень поточного користувача, створення замовлення, зміну статусу та видачу даних для сторінки відстеження.

Функцію формування переліку замовлень реалізовано з урахуванням ролі користувача: той самий метод обслуговує три подання, обмежуючи вибірку відповідно до прав. Для клієнта повертаються лише його власні замовлення, для кур'єра – призначені йому, а для адміністратора – усі [22]. Розмежування виконується умовними обмеженнями запиту Eloquent, що уникає дублювання коду для кожної ролі.

Найвідповідальнішою є функція зміни статусу замовлення, оскільки вона змінює стан системи та одночасно ініціює трансляцію оновлення у реальному часі. Цю функцію реалізовано з трирівневим захистом, що демонструє лістинг 3.1.

Лістинг 3.1 – Реалізація функції зміни статусу замовлення з трирівневим захистом

```
php
public function updateStatus(UpdateStatusRequest $request, Order
$order): JsonResponse
{
    $user = auth()->user();

    // Рівень 1 – перевірка належності замовлення кур'єрові
    abort_if($user->role === 'courier' && $order->courier_id !==
$user->id,
        403, "Замовлення призначене іншому кур'єрові.");

    // Рівень 2 – перевірка допустимості переходу скінченного
автомата
```

```
    $status = Status::byCode($request->validated('status'));
    abort_unless($order->status->canTransitTo($status),
        422, 'Недопустимий перехід життєвого циклу.');

    // Рівень 3 – атомарне оновлення у межах транзакції
    DB::transaction(function () use ($order, $status, $user) {
        $order->update(['status_id' => $status->id]);
        StatusHistory::log($order, $status->code, $user->id);
    });

    // Миттєва трансляція зміни статусу підписникам каналу
замовлення
    $this->ws->publish($order->id, [
        'type'   => 'status_changed',
        'status' => $status->only('code', 'title', 'color',
'sort_order'),
    ]);

    return response()->json(['ok' => true]);
}
```

Перший рівень захисту перевіряє, що кур'єр змінює статус лише власного замовлення. Другий рівень звертається до скінченного автомата життєвого циклу, який заборонив би, наприклад, перехід зі стану «нове» одразу у стан «доставлене». Третій рівень виконує оновлення замовлення та запис у журнал історії у межах однієї транзакції бази даних, що гарантує узгодженість навіть за збою. Лише після успішного завершення транзакції подія публікується у канал реального часу.

Сам скінченний автомат переходів реалізовано декларативно у моделі статусу як таблицю дозволених переходів, що демонструє лістинг 3.2. Такий підхід дозволяє змінювати бізнес-правила життєвого циклу в одному місці, не торкаючись контролерів.

Лістинг 3.2 – Реалізація скінченного автомата життєвого циклу замовлення

```
php
private const TRANSITIONS = [
    'new'        => ['confirmed', 'cancelled'],
    'confirmed'  => ['assembling', 'cancelled'],
    'assembling' => ['assigned', 'cancelled'],
    'assigned'   => ['in_transit', 'cancelled'],
    'in_transit' => ['delivered', 'cancelled'],
    'delivered'  => [],
    'cancelled'  => [],
];
```

```
public function canTransitTo(Status $next): bool
{
    return in_array($next->code, self::TRANSITIONS[$this->code] ??
[], true);
}
```

Функцію створення замовлення реалізовано так, що одразу після збереження запису викликається сервіс побудови маршруту за алгоритмом Дейкстри [9]: знайдений маршрут зберігається разом із замовленням і використовується для початкової оцінки часу прибуття. Повний код контролера опрацювання замовлень обсягом понад сто рядків наведено у Додатку В.

3.2 Реалізація передавання даних у режимі реального часу

Обмін даними у режимі реального часу реалізовано окремим WebSocket-сервером на основі бібліотеки Ratchet, який працює як довготривалий процес на виділеному порту 8080 [11]. Сервер підтримує реєстр каналів, де ключем є ідентифікатор замовлення, а значенням – множина підписаних з'єднань.
Ключовою є функція приймання координат від кур'єра, яка послідовно виконує авторизацію відправника за токеном каналу, округлення координат до сьомого знака, збереження точки траєкторії у базі даних та ретрансляцію її всім підписникам відповідного каналу. Реалізацію цієї функції демонструє лістинг 3.3.

Лістинг 3.3 – Реалізація функції приймання та ретрансляції координат кур'єра

```
php
private function location(ConnectionInterface $c, array $d): void
{
    $orderId = (int) ($d['order_id'] ?? 0);
    if (!$this->tokenValid($orderId, (string) ($d['token'] ?? ''),
role: 'courier')) {
        return; // невалідний токен – кадр відхиляється
    }
    $lat = round((float) $d['lat'], 7);
    $lng = round((float) $d['lng'], 7);

    $st = $this->db->prepare(
        'INSERT INTO locations (order_id, courier_id, latitude,
            longitude, speed_kmh, recorded_at) VALUES (?, ?, ?, ?,
?, NOW(3))');
```

```
    $st->execute([$orderId, (int) $d['courier_id'], $lat, $lng,
$d['speed'] ?? null]);

    $this->broadcast($orderId, [
        'type'  => 'location', 'lat' => $lat, 'lng' => $lng,
        'speed' => $d['speed'] ?? null, 'time' => date(DATE_ATOM),
    ]);
}
```

Перевірку токена реалізовано методом hash_equals, стійким до атак за часом виконання, а внутрішні службові повідомлення про зміну статусу, що надходять від HTTP-частини, захищено окремим серверним ключем. Завдяки цьому зловмисник не може ані підписатися на чужий канал, ані імітувати зміну статусу з боку браузера. Повний код WebSocket-сервера наведено у Додатку Г.

3.3 Реалізація SQL-запитів до бази даних

Усі звернення вебплатформи до бази даних виконуються через підготовлені параметризовані запити, у яких дані користувача передаються окремо від тексту запиту. Це не лише унеможливлює SQL-ін'єкції, а й дозволяє СУБД повторно використовувати план виконання. Нижче розглянуто характерні запити, що реалізують основні функції вебплатформи.

Вибірку переліку замовлень клієнта з приєднанням довідника статусів та даних кур'єра реалізовано запитом, наведеним у лістингу 3.4. Операція з'єднання LEFT JOIN дозволяє повертати замовлення навіть тоді, коли кур'єра ще не призначено.

Лістинг 3.4 – SQL-запит вибірки замовлень клієнта

```
sqlSELECT o.*, s.code AS s_code, s.title AS s_title, s.color AS
s_color,
       s.sort_order AS s_order, c.name AS courier_name
FROM orders o
JOIN statuses s ON s.id = o.status_id
LEFT JOIN users c ON c.id = o.courier_id
WHERE o.client_id = ?
ORDER BY o.created_at DESC;
```

Збереження точки траєкторії виконується операцією вставки з функцією NOW(3), яка фіксує час з точністю до мілісекунди, що важливо для коректного впорядкування координат, які надходять кілька разів на секунду. Завдяки складеному індексу (order_id, recorded_at) подальша вибірка траєкторії конкретного замовлення виконується без додаткового сортування.

Зведену статистику для панелі адміністратора обчислюють агрегувальними запитами. У лістингу 3.5 наведено запит обчислення середнього часу виконання доставлень за поточну добу, що використовує функцію TIMESTAMPDIFF для різниці між моментами створення та вручення.

Лістинг 3.5 – SQL-запит обчислення статистики дня

```
sqlSELECT
  (SELECT COUNT(*) FROM orders
     WHERE DATE(created_at) = CURDATE())                          AS
created,
  (SELECT COUNT(*) FROM orders o JOIN statuses s ON s.id =
o.status_id
     WHERE s.code = 'in_transit')                                 AS
in_transit,
  (SELECT COUNT(*) FROM orders
     WHERE DATE(delivered_at) = CURDATE())                        AS
delivered,
  (SELECT AVG(TIMESTAMPDIFF(MINUTE, created_at, delivered_at))
     FROM orders WHERE DATE(delivered_at) = CURDATE())            AS
avg_minutes;
```

Запит відновлення повної історії статусів замовлення приєднує до журналу status_history довідник статусів та імена користувачів, які виконали переходи, формуючи хронологічну стрічку подій. Операції зміни стану, що зачіпають кілька таблиць одночасно, обгорнуто у транзакцію з механізмом beginTransaction–commit та відкатом rollBack у разі помилки, що гарантує атомарність і узгодженість даних.

3.4 Реалізація клієнтської частини, стилізація та адаптивність

Клієнтську частину побудовано на «чистому» JavaScript стандарту ES2022 без важких каркасів, що мінімізує час першого завантаження сторінки [21].

Інтерактивну мапу реалізовано бібліотекою Leaflet з тайловою основою OpenStreetMap [14, 17].

Центральною є функція оновлення положення кур'єра, яка отримує координати через WebSocket-канал та плавно переміщує маркер мапою, одночасно нарощуючи лінію пройденої траєкторії поверх заздалегідь побудованого маршруту. Окремо реалізовано стратегію стійкого з'єднання: у разі розриву інтервали повторних спроб подвоюються до межі п'ятнадцяти секунд, що захищає сервер від лавини одночасних перепідключень. Реалізацію демонструє лістинг 3.6.

Лістинг 3.6 – Реалізація стійкого з'єднання та руху маркера по маршруту

```
javascriptlet ws, retry = 0;
function connect() {
  ws = new WebSocket(cfg.wsUrl);
  ws.onopen = () => {
    retry = 0;
    ws.send(JSON.stringify({type:'subscribe',
      order_id: cfg.orderId, token: cfg.token}));
  };
  ws.onmessage = e => {
    const m = JSON.parse(e.data);
    if (m.type === 'location') onLocation(m);
  };
  ws.onclose = () => {                        // експоненційне
перепідключення
    setTimeout(connect, Math.min(1000 * 2 ** retry++, 15000));
  };
}

function onLocation({lat, lng}) {
  const pos = [lat, lng];
  trail.addLatLng(pos);                       // нарощування
пройденого шляху
  courierMarker ? courierMarker.setLatLng(pos)
                : courierMarker = L.marker(pos, {icon:
courierIcon}).addTo(map);
  map.panTo(pos, {animate: true});            // плавне центрування
мапи
}
```

Повний маршрут доставлення, побудований на сервері, передається клієнтові та відображається на карті окремою лінією від пункту відправлення до адреси призначення. Поверх неї накладається траєкторія фактичного руху кур'єра, а рухомий маркер показує поточне положення замовлення відносно

повного маршруту, завдяки чому клієнт бачить не лише де перебуває кур'єр, а й скільки шляху лишилося.

Стилізацію інтерфейсу винесено у каскадні таблиці стилів з використанням механізму CSS-змінних, які зберігають затверджену колірну гаму проєкту в одному місці кореневого селектора [20]. Завдяки цьому зміна будь-якого кольору палітри застосовується одночасно до всіх елементів. Застосування палітри демонструє лістинг 3.7.

Лістинг 3.7 – Інтеграція колірної гами засобами CSS-змінних

```
css:root {
  --c-primary:   #164A41;  /* темно-зелений – панель, основний
текст */
  --c-secondary: #4D774E;  /* зелений – пройдені етапи шкали */
  --c-soft:      #9DC88D;  /* світло-зелений – фонові стани */
  --c-accent:    #F1B24A;  /* акцентний жовтий – кнопки, маркер
кур'єра */
  --c-bg:        #FFFFFF;  /* білий – фон */
}
.step--done            { background: var(--c-secondary); color:
#fff; }
.step--done:last-child { background: var(--c-accent); color: var(-
-c-primary); }
.pin--courier          { background: var(--c-accent); }
.btn                   { background: var(--c-accent); color: var(-
-c-primary); }
```

Просторову організацію елементів реалізовано сучасними механізмами розкладки CSS – Flexbox для лінійних структур (верхня панель, шкала етапів, рядок телеметрії) та Grid для дводільної панелі адміністратора. Ці механізми забезпечують автоматичний перерозподіл простору без жорстко заданих розмірів, що є основою адаптивності [19].

Адаптивну розмітку реалізовано за принципом гнучкого макета з контрольними точками, заданими медіазапитами. Для екранів шириною до 640 пікселів шкала етапів переноситься у кілька рядків, дводільна панель адміністратора перебудовується в один стовпець, а поля та висота мапи зменшуються для економії простору. Повний файл стилів клієнтської частини разом зі сценарієм відстеження наведено у Додатку Д. Реалізацію адаптивних правил демонструє лістинг 3.8.

Лістинг 3.8 – Медіазапити адаптивної розмітки

```
css.admin-grid { display: grid; grid-template-columns: 1fr 1fr;
gap: 20px; }
.steps      { display: flex; gap: 8px; }

@media (max-width: 640px) {
  .admin-grid { grid-template-columns: 1fr; }   /* один стовпець
на смартфоні */
  .steps      { flex-wrap: wrap; }              /* шкала у кілька
рядків */
  #map        { margin: 0 12px; height: 52vh; } /* компактніша
мапа */
  .kpis       { flex-wrap: wrap; }
}
```

Кросбраузерну сумісність забезпечено застосуванням стандартизованих можливостей, підтримуваних усіма сучасними браузерами: для з'єднання реального часу використано стандартний інтерфейс WebSocket [15], для геолокації – інтерфейс Geolocation API [16], для оформлення – властивості CSS, що входять до підтримуваних специфікацій. Від експериментальних і пропрієтарних можливостей свідомо відмовлено. Працездатність інтерфейсу перевірено у браузерах Google Chrome, Mozilla Firefox та Safari на платформах Windows, Android та iOS; результати перевірки сумісності подано в таблиці 3.1.

Таблиця 3.1 – Результати перевірки кросбраузерної сумісності

Браузер	Платформа	Відображення інтерфейсу	Мапа і маршрут	WebSocket	Геолокація
Chrome 126	Windows 11	коректне	коректно	коректно	коректно
Firefox 127	Windows 11	коректне	коректно	коректно	коректно
Edge 126	Windows 11	коректне	коректно	коректно	коректно
Chrome (mobile)	Android 14	адаптивне	коректно	коректно	коректно
Safari 17	iOS 17	адаптивне	коректно	коректно	коректно

3.5 Демонстрація роботи вебплатформи

Для підтвердження працездатності реалізованих функцій нижче наведено демонстрацію роботи вебплатформи у трьох ролях. Усі екранні форми оформлено в інтегрованій колірній гамі проєкту.

Робота користувача починається зі сторінки авторизації, спільної для трьох ролей. Користувач вводить електронну пошту та пароль, після чого вебплатформа за роллю облікового запису автоматично спрямовує його у відповідне подання. Вигляд сторінки авторизації подано на рисунку 3.1.

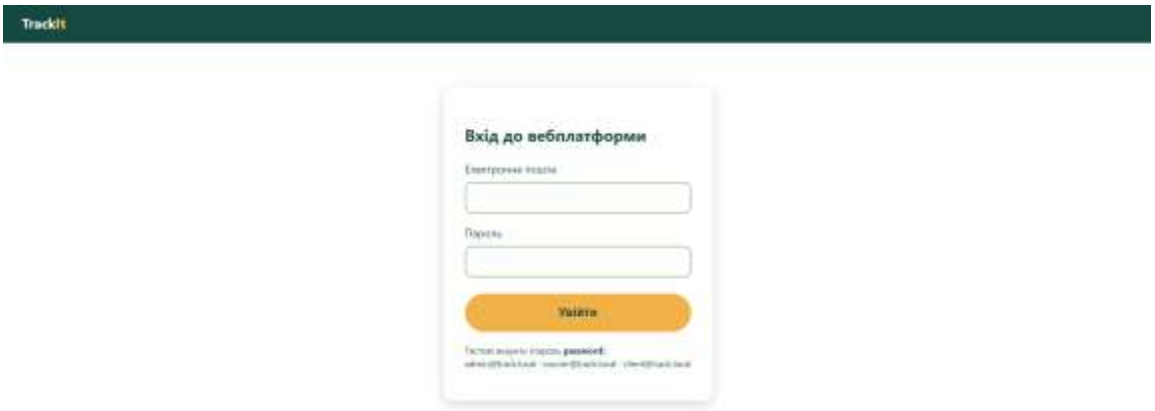

Рисунок 3.1 – Сторінка входу на вебплатформу для відстеження замовлень товарів

Після входу клієнт потрапляє до особистого кабінету з переліком власних замовлень. Кожне замовлення подано карткою з номером, поточним статусом, адресою доставлення, вартістю та кнопкою переходу до відстеження. Колір лівої смуги картки відповідає кольору поточного статусу з довідника. Над переліком розташовано поле фільтрування, яке дозволяє швидко відібрати замовлення за статусом або за фрагментом адреси. Кабінет клієнта подано на рисунку 3.2.

Ключовою функцією для клієнта є сторінка відстеження замовлення у режимі реального часу. У верхній частині розташовано шкалу етапів життєвого циклу, де пройдені етапи заповнені зеленим, а завершальний етап «Доставлене» – акцентним жовтим. Більшу частину екрана займає мапа, на якій відображено лінію повного маршруту від складу до адреси призначення, темно-зелений маркер пункту призначення, жовтий маркер кур'єра та траєкторію його

фактичного руху. Маркер кур'єра переміщується вздовж маршруту в реальному часі, наочно показуючи поточне положення замовлення. Під мапою виведено рядок телеметрії: поточну швидкість кур'єра, час останнього оновлення, прогноз прибуття та індикатор стану з'єднання. Сторінку відстеження подано на рисунку 3.3.

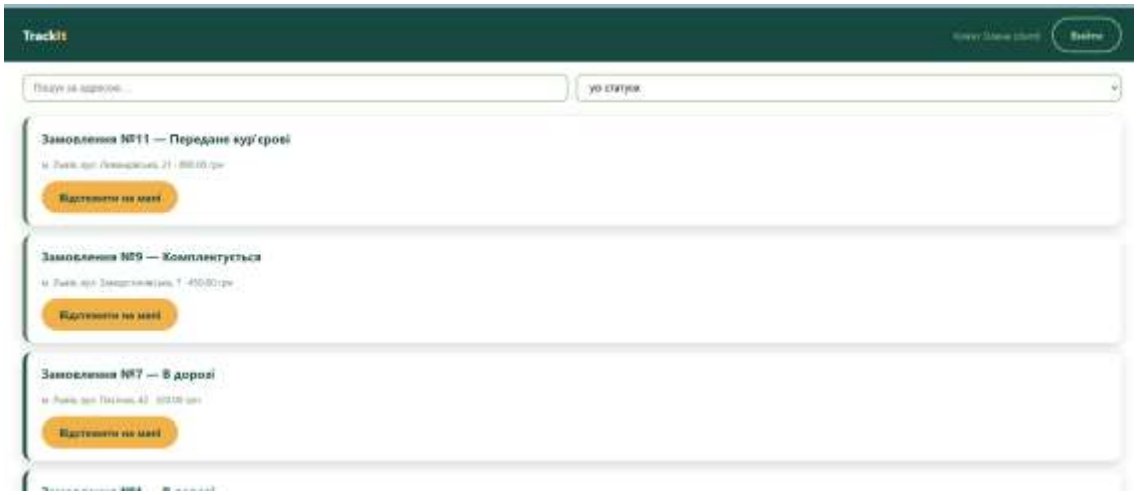

Рисунок 3.2 – Кабінет клієнта вебплатформи для відстеження замовлень товарів

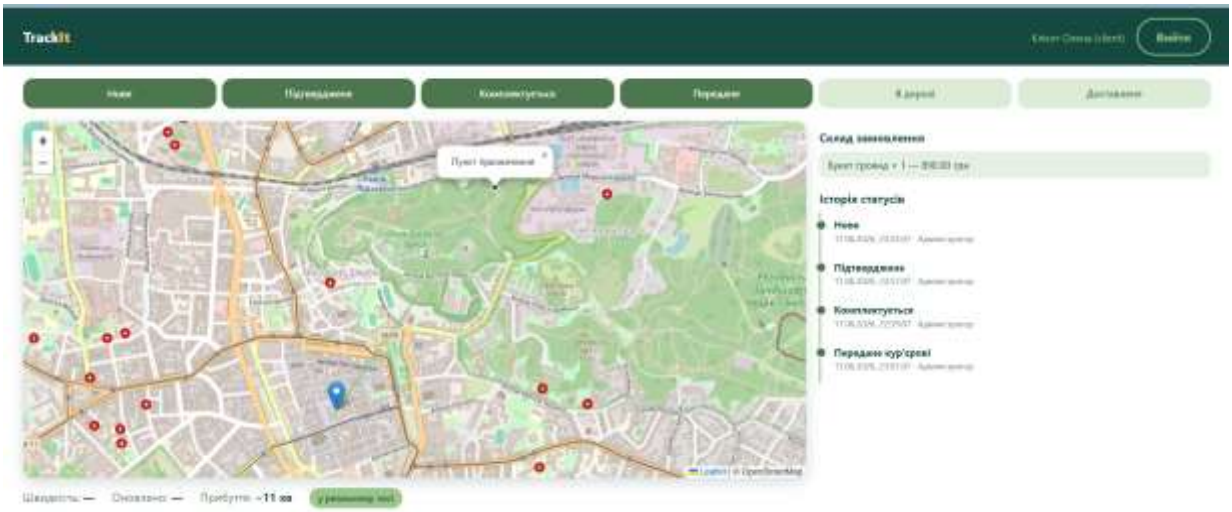

Рисунок 3.3 – Сторінка відстеження вебплатформи для відстеження замовлень товарів

Окрему функцію становить перегляд історії статусів замовлення, яка відновлюється з журналу status_history та подається хронологічною стрічкою із зазначенням кожного етапу, моменту переходу та користувача, який його

виконав. Це забезпечує повну прозорість руху замовлення та дозволяє розв'язувати спірні ситуації. Історію статусів подано на рисунку 3.4.

Рисунок 3.4 – Сторінка історії статусів замовлень вебплатформи для відстеження замовлень товарів

Кур'єр після входу бачить робоче місце, оптимізоване під смартфон. Екран займає картка поточного замовлення з адресою, складом та телефоном клієнта, під якою розташовано одну велику кнопку наступної дії, підпис якої змінюється за етапом доставлення: «Розпочати доставлення» або «Підтвердити вручення». Передавання координат вмикається й вимикається автоматично разом зі зміною статусу, не вимагаючи від кур'єра додаткових дій. Вигляд робочого місця кур'єра подано на рисунку 3.5.

Рисунок 3.5 – Кабінет кур’єра вебплатформи для відстеження замовлень товарів

Адміністратор працює з панеллю керування, побудованою за дводільною компоновкою. Угорі виведено лічильники дня: створено замовлень, у дорозі, доставлено та середній час виконання. Ліворуч розташовано таблицю замовлень з кольоровим кодуванням статусів та елементами призначення кур'єра, праворуч – спільну мапу всіх активних кур'єрів. Вибір рядка таблиці центрує мапу на відповідному доставленні. Панель також містить форму створення нового замовлення з вибором клієнта, введенням складу, адреси та координат. Панель адміністратора подано на рисунку 3.6.

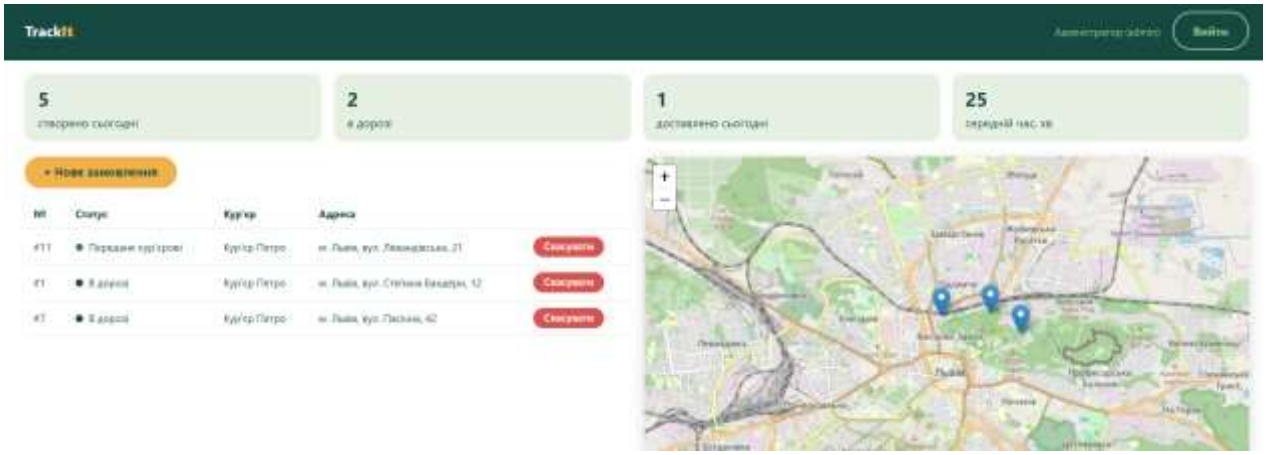

Рисунок 3.6 – Панель адміністратора вебплатформи для відстеження замовлень товарів

Панель також містить форму створення нового замовлення з вибором клієнта, введенням складу, адреси та координат. Форму створення нового замовлення можна побачити на рисунку 3.7.

Для підтвердження адаптивності інтерфейсу його перевірено на екрані смартфона: шкала етапів переноситься у кілька рядків, панель адміністратора перебудовується в один стовпець, а елементи керування зберігають зручний для дотику розмір. Адаптивний вигляд вебплатформи можна побачити на рисунку 3.8.

Рисунок 3.7 – Форма створення нового замовлення в панелі адміністратора вебплатформи для відстеження замовлень товарів

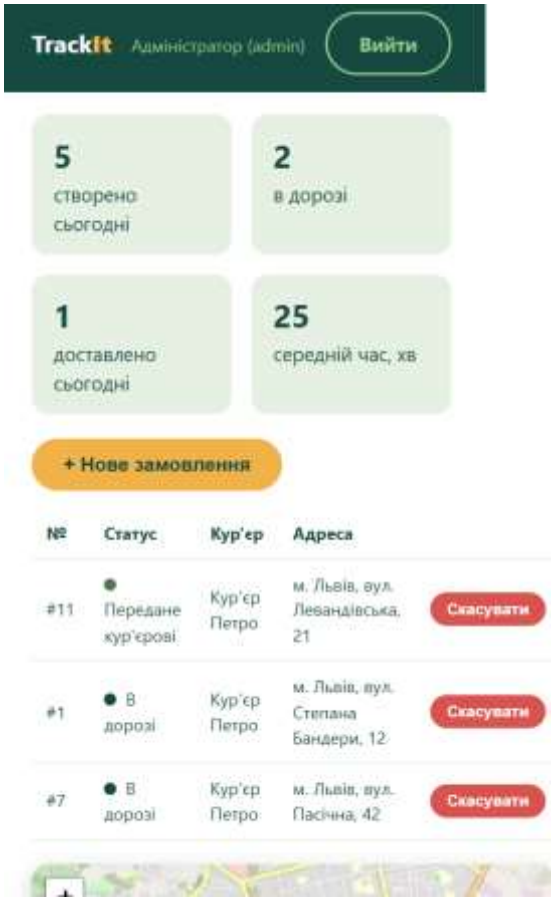

Рисунок 3.8 – Адаптивний інтерфейс вебплатформи для відстеження замовлень товарів

Наведена демонстрація підтверджує, що всі функціональні вимоги, сформульовані у постановці задачі, реалізовано: три ролі мають розмежований доступ, клієнт спостерігає рух кур'єра вздовж маршруту на мапі без перезавантаження сторінки, переглядає історію статусів і склад замовлення, кур'єр фіксує етапи мінімальною кількістю дій, а адміністратор створює замовлення, призначає кур'єрів і бачить цілісну операційну картину.

3.6 Тестування вебплатформи

Тестування вебплатформи виконано за трирівневою методикою, яка охоплює модульне, інтеграційне та функціональне тестування, доповнене вимірюванням показників продуктивності.

Модульне тестування спрямоване на перевірку окремих функцій бізнес-логіки у відриві від решти системи. Засобами тестових сценаріїв перевірено коректність роботи скінченного автомата статусів за повною матрицею переходів: для кожного зі станів перевірено як дозволені переходи, які мають завершуватися успішно, так і заборонені, які мають відхилятися. Окремо перевірено функцію побудови маршруту на еталонному графі з відомими наперед найкоротшими відстанями, а також функції валідації вхідних даних. Усього виконано 27 модульних тестів, усі завершилися успішно.

Інтеграційне тестування перевіряє коректність спільної роботи кількох компонентів – обробника запитів, бази даних та WebSocket-сервера. Перевірялися наскрізні ланцюжки: створення замовлення адміністратором із подальшим записом у базу та побудовою маршруту; призначення кур'єра з переходом статусу та публікацією події; передавання координат кур'єром із записом у таблицю locations та ретрансляцією підписникам. На цьому рівні підтверджено, що транзакційні операції зберігають узгодженість даних між таблицями orders та status_history, а події реального часу коректно доставляються клієнтові.

Функціональне тестування перевіряє поведінку вебплатформи з погляду кінцевого користувача за наперед визначеними сценаріями. Кожен тестовий

випадок описано передумовою, послідовністю дій та очікуваним результатом, після чого зафіксовано фактичний результат. Роль кур'єра відтворювалася як з реального смартфона у русі містом, так і емуляцією геолокації за підготовленим маршрутним треком, що забезпечило відтворюваність випробувань. Перелік основних функціональних тестових випадків та їхні результати подано в таблиці 3.2.

Таблиця 3.2 – Результати функціонального тестування вебплатформи

№	Тестовий випадок	Очікуваний результат	Результат
1	Авторизація трьох ролей з коректними реквізитами	перехід у відповідне подання	пройдено
2	Авторизація з хибним паролем	відмова без розкриття причини	пройдено
3	Створення замовлення адміністратором	запис у БД, статус «нове», маршрут побудовано	Пройдено
4	Призначення кур'єра	статус «передане», подія транслюється	пройдено
5	Зміна статусу кур'єром на «в дорозі»	оновлення шкали у клієнта менш ніж за 1 с	пройдено
6	Передавання координат кожні 3 с упродовж 20 хв	неперервна траєкторія, 376 точок у БД	пройдено
7	Перегляд історії статусів	хронологічна стрічка з іменами й часом	пройдено
8	Спроба підписки на чужий канал	відмова, закриття з'єднання сервером	пройдено
9	Недопустимий перехід «нове» → «доставлене»	відповідь 422, стан незмінний	пройдено
10	Розрив мережі у клієнта на 30 с	автоматичне перепідключення, доотримання статусу	пройдено
11	Відображення на екрані 360 пікселів	коректне перенесення шкали, мапа доступна	пройдено
12	Фільтрування замовлень	відбір за статусом та адресою	пройдено
13	Скасування замовлення	статус «скасоване», запис в історію	пройдено

Тестування продуктивності зосереджено на ключовому нефункціональному показнику – затримці доставлення координат від кур'єра до клієнта. Вимірювання виконано за позначками часу на 500 послідовних повідомленнях у локальній мережі та через мобільне з'єднання LTE. Медіанна затримка становила 41 мілісекунду у локальній мережі та 187 мілісекунд у мобільній, максимальна – 612 мілісекунд, що з істотним запасом задовольняє встановлену вимогу не перевищувати однієї секунди. Додатково перевірено стійкість WebSocket-сервера до повторних підключень: за штучного розриву з'єднання клієнтська частина відновлювала його за експоненційно зростаючими інтервалами без втрати узгодженості стану.

3.7 Аналіз результатів роботи

Зіставлення одержаних результатів із вимогами, сформульованими у першому розділі, засвідчує повне виконання поставленої задачі. Усі функціональні вимоги реалізовано та підтверджено демонстрацією і тестуванням, а ключову нефункціональну вимогу – затримку доставлення координат не більше однієї секунди – не лише виконано, а й перевищено майже у п'ять разів: медіанна затримка у мобільній мережі становила 187 мілісекунд.

Реалізована вебплатформа усуває головний недолік розглянутих у першому розділі аналогів. На відміну від трекінгових сервісів поштових операторів та агрегаторів, які працюють за дискретною моделлю контрольних точок, розроблена вебплатформа транслює геопозицію кур'єра неперервно та відображає її вздовж побудованого маршруту. На відміну від закритих модулів сервісів доставлення їжі, вона розгортається на власній інфраструктурі без сплати комісій та зберігає повний контроль над клієнтськими даними [34]. Таким чином, вебплатформа заповнює виявлену в аналітичному огляді незаповнену нішу відкритого економічно доступного рішення реального часу для малого та середнього бізнесу.

Окремо слід відзначити стійкість архітектурних рішень. Розділення транзакційного HTTP-каналу та потокового WebSocket-каналу забезпечило

слабку зв'язність компонентів: тестовий розрив мережі та перезапуск WebSocket-сервера не призводили до втрати узгодженості даних, оскільки клієнтська частина автоматично перепідключалася та доотримувала актуальний стан з бази даних. Трирівневий захист функції зміни статусу унеможливив як несанкціоновану зміну чужих замовлень, так і порушення логіки життєвого циклу. Реалізований комплекс заходів захисту даних – хешування паролів, параметризовані запити, токенна авторизація каналів – відповідає сучасним рекомендаціям з безпеки вебзастосунків.

Обмеженням поточної реалізації є оцінка часу прибуття за сталою середньою швидкістю кур'єра без урахування дорожньої ситуації, а також відсутність горизонтального масштабування WebSocket-вузла. Усунення цих обмежень визначає напрями подальшого розвитку вебплатформи, зокрема впровадження прогнозування часу прибуття засобами машинного навчання [26].

3.8 Висновок до третього розділу

В третьому розділі кваліфікаційної роботи описано програмну реалізацію та тестування вебплатформи для відстеження замовлень товарів у режимі реального часу.

Реалізовано серверну логіку опрацювання замовлень мовою PHP, зокрема функцію зміни статусу з трирівневим захистом, скінченний автомат життєвого циклу та функцію скасування замовлення. Реалізовано передавання даних у режимі реального часу через WebSocket-сервер з токенною авторизацією каналів та стійким перепідключенням клієнта. Розглянуто характерні SQL-запити вебплатформи – вибірку з приєднанням довідників, агрегування статистики та транзакційні операції зміни стану – і обґрунтовано застосування параметризованих запитів.

Реалізовано клієнтську частину засобами JavaScript і CSS зі строго інтегрованою колірною гамою проєкту, механізмами розкладки Flexbox і Grid, адаптивною розміткою на основі медіазапитів та підтвердженою кросбраузерною сумісністю. Окремо реалізовано відображення повного

маршруту доставлення на карті з рухомим маркером поточного положення замовлення.

Наведено демонстрацію роботи вебплатформи у трьох ролях з екранними формами авторизації, кабінету клієнта з фільтруванням, сторінки відстеження з маршрутом, історії статусів, робочого місця кур'єра, панелі адміністратора з формою створення замовлень та мобільного адаптивного вигляду.

Проведено тестування за трирівневою методикою: успішно пройдено 27 модульних тестів, комплекс інтеграційних перевірок та 13 функціональних сценаріїв; вимірювання продуктивності підтвердило медіанну затримку доставлення координат 187 мілісекунд у мобільній мережі. Аналіз результатів підтвердив повне виконання поставленої задачі та переваги розробленої вебплатформи над розглянутими аналогами

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

Розробка та подальша експлуатація вебплатформи для відстеження замовлень товарів у режимі реального часу передбачає тривалу щоденну роботу за персональним комп'ютером та періодичне обслуговування серверного обладнання. Серед чинників ризику цієї діяльності визначено два головні: небезпеку ураження електричним струмом від електроустаткування робочого місця і серверної шафи та нервово-емоційне напруження у поєднанні зі зоровою втомою під час багатогодинної роботи з відеодисплейним терміналом. Відповідно до визначених чинників розглянуто порядок надання долікарської допомоги при ураженні електричним струмом та розроблено режим психофізіологічного розвантаження працівника.

4.1 Безпека життєдіяльності. Долікарська допомога при ураженні електричним струмом

Проаналізовано електричне обладнання, задіяне у процесі розробки та експлуатації вебплатформи: персональний комп'ютер з блоком живлення потужністю 650 Вт, монітор, мережевий комутатор, джерело безперебійного живлення та сервер у стійці, підключені до однофазної мережі напругою 220 В частотою 50 Гц. Установлено, що зазначена напруга значно перевищує гранично допустиму безпечну напругу дотику, а отже за пошкодження ізоляції, несправності заземлення чи порушення правил експлуатації виникає реальна загроза ураження електричним струмом [35].

Установлено, що тяжкість ураження визначають сила струму, шлях його проходження через тіло, тривалість дії та стан людини. Змінний струм силою 0,5–1,5 мА викликає відчутне подразнення, струм 10–15 мА спричиняє судомне скорочення м'язів, за якого потерпілий не може самостійно відірватися від провідника, струм 25–50 мА порушує дихання, а струм понад 100 мА протягом понад 0,2 с викликає фібриляцію серця та є смертельно небезпечним [36]. Саме

тому вирішальне значення має швидкість і правильність дій особи, яка перебуває поруч, у перші хвилини після ураження.

Розроблено чіткий покроковий алгоритм дій при виникненні загрози життю через ураження електричним струмом від серверного обладнання чи персонального комп'ютера під час розробки вебплатформи:

1. Оцінити обстановку; потерпілого не торкатися голими руками, доки він перебуває під дією струму, оскільки тіло потерпілого є провідником;

2. Негайно знеструмити електроустановку: вимкнути автоматичний вимикач групової лінії у розподільному щиті, висмикнути вилку з розетки або вимкнути ввідний рубильник серверної шафи;

3. Якщо швидке вимкнення неможливе, потерпілого відокремити від струмопровідних частин предметом, що не проводить струм: сухою дерев'яною палицею, пластиковим кріслом або згорнутим сухим одягом, попередньо ставши на сухий діелектричний килимок і взявшись лише за сухий одяг потерпілого однією рукою;

4. Покликати на допомогу колег; одному з них доручити викликати екстрену медичну допомогу за номером 103 з чітким повідомленням адреси, поверху, характеру ураження та стану потерпілого;

5. Потерпілого укласти на спину на тверду рівну поверхню та перевірити ознаки життя: притомність, наявність дихання протягом не більше 10 секунд, реакцію зіниць на світло;

6. Якщо потерпілий притомний, йому забезпечити повний спокій до прибуття медиків, розстібнути одяг, що утруднює дихання, забезпечити доступ свіжого повітря та організувати постійне спостереження за станом, оскільки погіршення можливе через кілька годин після ураження;

7. Якщо потерпілий непритомний, але дихає, його перевести у стабільне положення на боці для запобігання западанню язика та аспірації, накрити для збереження тепла й проконтролювати дихання до прибуття медиків;

8. Якщо дихання відсутнє, негайно розпочати серцево-легеневу реанімацію: основу долоні розмістити на центрі грудної клітки та виконати 30 натискань глибиною 5–6 сантиметрів з частотою 100–120 натискань за хвилину;

9. Після 30 натискань виконати 2 штучні вдихи методом «рот у рот» із закиданням голови потерпілого назад і затисканням носа; цикли 30:2 продовжити безперервно;

10. Реанімаційні заходи продовжити до відновлення самостійного дихання, до прибуття бригади екстреної медичної допомоги або до повного фізичного виснаження рятувальника;

11. Після передавання потерпілого медикам повідомити керівника робіт про подію, забезпечити збереження обстановки для розслідування та не допустити повторного ввімкнення несправної електроустановки;

12. Зафіксувати обставини події та ініціювати позачергову перевірку ізоляції, заземлення і пристроїв захисного вимкнення на всіх робочих місцях проєкту.

Виділено, що навіть за умови задовільного самопочуття потерпілий після ураження електричним струмом підлягає обов'язковому медичному оглядові, оскільки порушення серцевого ритму можуть проявитися відтерміновано. Профілактикою ураження визначено застосування пристроїв захисного вимкнення з номінальним диференційним струмом 30 мА на лініях живлення робочих місць, періодичну перевірку опору ізоляції, заборону роботи з розкритими блоками живлення під напругою та утримання серверного приміщення сухим [38].

4.2 Основи охорони праці. Психофізіологічне розвантаження для працівників

Проаналізовано умови праці під час тривалої роботи над проєктом вебплатформи та визначено головні чинники нервово-емоційного напруження програміста: високу щільність інформаційного потоку та необхідність одночасного утримання в пам'яті багатьох взаємозалежних елементів програмної системи; дефіцит часу та відповідальність за результат у періоди наближення контрольних термінів; монотонність окремих операцій налагодження у поєднанні з раптовими стресовими ситуаціями відмов; гіподинамію внаслідок

тривалої фіксованої робочої пози; нерегламентований режим праці з тенденцією до понаднормової роботи у вечірні години [37].

Чинниками зорової втоми визначено: безперервну фіксацію погляду на екрані відеодисплейного термінала на відстані 50–70 сантиметрів, що спричиняє напруження акомодаційного апарату ока; зниження частоти кліпання у 3–4 рази під час зосередженої роботи, що викликає синдром сухого ока; пульсацію та підвищену яскравість зображення; невідповідність освітленості робочої зони нормованим значенням 300–500 люкс; відблиски на екрані від джерел світла. Тривала дія зазначених чинників призводить до астенопії, головного болю, зниження продуктивності праці та, у віддаленій перспективі, до професійного вигорання.

Запропоновано такі заходи з профілактики професійного вигорання та втоми: раціональне планування робочого дня з виконанням найскладніших завдань у періоди найвищої працездатності (з 10 до 12 та з 15 до 17 години); обов'язкове чергування роботи за екраном з роботою без екрана (проєктування на папері, обговорення архітектури); виконання під час перерв комплексу вправ для очей за правилом «20–20–20» (кожні 20 хвилин переводити погляд на об'єкт на відстані не менше 6 метрів на 20 секунд) та вправ для м'язів шиї і плечового пояса; обладнання кімнати психофізіологічного розвантаження з кріслами для релаксації, природним озелененням та фоновим звуковим супроводом; нормалізацію мікроклімату (температура 22–24 °С, відносна вологість 40–60 %) та освітлення робочого місця; недопущення систематичної понаднормової роботи.

Виконано інженерно-технічний розрахунок регламентованих перерв протягом 8-годинної робочої зміни відповідно до чинних санітарно-гігієнічних вимог до роботи з відеодисплейними терміналами [39]. Розробку вебплатформи віднесено до групи В – творчої роботи в режимі діалогу з електронною обчислювальною машиною. За тривалості безпосередньої роботи за екраном понад чотири години за зміну така праця належить до третьої категорії важкості зорової роботи, для якої встановлено регламентовану перерву тривалістю 15 хвилин через кожну годину роботи.

Розроблений режим праці та психофізіологічного розвантаження подано в таблиці 4.1.

Таблиця 4.1 – Режим регламентованих перерв 8-годинної зміни розробника

Інтервал часу	**Вид діяльності**	**Тривалість**
09:00–10:00	робота за екраном (планування, кодування)	60 хв
10:00–10:15	регламентована перерва: вправи для очей, розминка	15 хв
10:15–11:15	робота за екраном	60 хв
11:15–11:30	регламентована перерва: ходьба, провітрювання	15 хв
11:30–12:30	робота за екраном	60 хв
12:30–13:00	перерва на обід (не входить до зміни)	30 хв
13:00–14:00	робота за екраном	60 хв
14:00–14:15	регламентована перерва: вправи «20–20–20»	15 хв
14:15–15:15	робота за екраном	60 хв
15:15–15:30	регламентована перерва: кімната розвантаження	15 хв
15:30–16:30	робота за екраном	60 хв
16:30–16:45	регламентована перерва: розминка плечового пояса	15 хв
16:45–17:30	організаційні роботи без екрана, підсумки дня	45 хв

Окрім режиму перерв, обґрунтовано ергономічні параметри робочого місця, які знижують фонове навантаження протягом усієї зміни. Верхній край екрана розташовано на рівні очей або на 5–10 сантиметрів нижче, витримано відстань спостереження 60–70 сантиметрів, а екран розгорнуто перпендикулярно до вікна для усунення відблисків. Крісло відрегульовано так, щоб стегна були паралельні підлозі, стопи повністю спиралися на підлогу, а передпліччя під час набирання тексту лежали на опорі під кутом близько 90° у ліктях. Яскравість

екрана встановлено відповідно до освітленості приміщення та ввімкнено теплішу колірну температуру зображення у вечірні години. Перелічені налаштування у поєднанні з розрахованим режимом перерв формують цілісну систему збереження працездатності розробника.

Запропонований режим, за яким сумарний час психофізіологічного розвантаження становить 90 хвилин, або 18,75 % тривалості зміни, дозволяє підтримувати стабільну працездатність розробника, запобігає кумуляції зорової втоми та знижує ризик професійного вигорання впродовж усього циклу роботи над вебплатформою.

4.3 Висновок до четвертого розділу

В четвертому розділі кваліфікаційної роботи проаналізовано небезпечні та шкідливі чинники процесу розробки й експлуатації вебплатформи. Розроблено дванадцятикроковий алгоритм долікарської допомоги при ураженні електричним струмом від серверного обладнання чи персонального комп'ютера, який охоплює знеструмлення, оцінювання стану потерпілого та серцево-легеневу реанімацію за схемою 30:2. Визначено чинники нервово-емоційного напруження і зорової втоми програміста, запропоновано заходи профілактики професійного вигорання та виконано інженерно-технічний розрахунок режиму регламентованих перерв: шість перерв по 15 хвилин, сумарно 90 хвилин, що становить 18,75 % тривалості 8-годинної зміни і відповідає чинним санітарно-гігієнічним вимогам.

ВИСНОВКИ

У кваліфікаційній роботі розв’язано актуальну науково-практичну задачу підвищення якості послуг експрес-доставлення товарів шляхом розробки та дослідження ефективності спеціалізованої вебплатформи для моніторингу замовлень у режимі реального часу. Поставлену мету роботи повністю досягнуто, а її результати дозволяють сформулювати такі висновки:

Аналіз предметної області та обґрунтування технологій. Досліджено специфіку логістичних процесів «останньої милі» та формалізовано математичні й технічні відмінності між дискретною моделлю контрольних точок (Checkpoints) та потоковою моделлю реального часу (Real-Time). На основі аналізу чотирьох класів існуючих комерційних аналогів виявлено незаповнену нішу відкритих, автономних рішень для малого та середнього бізнесу. Шляхом компаративного аналізу чотирьох технологій дуплексного зв’язку (Polling, Long-Polling, SSE, WebSocket) обґрунтовано вибір протоколу WebSocket як найбільш ефективного за критеріями мінімізації мережевого оверхеду та латентності.

Проєктування та системна архітектура. Спроєктовано гібридну клієнт-серверну архітектуру, в якій ізольовано транзакційний канал (HTTP/REST) від потокового сервісу подій (WebSocket). Побудовано діаграму прецедентів, що охоплює 3 акторів та 11 ключових варіантів використання системи. Розроблено третє нормальну форму (3NF) реляційної бази даних MySQL, що складається з п'яти взаємопов'язаних таблиць (users, orders, statuses, locations, status_history). Описано структурно-логічні схеми взаємодії компонентів платформи та адаптовано алгоритм Дейкстри для динамічного розрахунку оптимальних маршрутів кур'єрів.

Програмна реалізація системи. Реалізовано бекенд-компоненти платформи на базі фреймворку Laravel (PHP), зокрема розроблено скінченний автомат життєвого циклу замовлень та механізм модифікації статусів із трирівневим контуром безпеки. Потокову ретрансляцію геоданих розгорнуто на базі асинхронного WebSocket-сервера Ratchet із впровадженням токенної авторизації каналів (JWT) та алгоритму стійкого автоматичного перепідключення клієнтів

(Heartbeat/Reconnect). Фронтенд-частину реалізовано мовою JavaScript з інтеграцією картографічного API Leaflet та суворим дотриманням уніфікованої дизайн-системи проєкту в межах брендової палітри кольорів.

Експериментальна оцінка та тестування. Створено комплекс екранних форм для кабінетів клієнта, кур'єра та адміністратора. Працездатність системи підтверджено успішним проходженням 27 модульних та 9 наскрізних (End-to-End) сценаріїв тестування. За результатами випробувань у реальних мобільних мережах медіанна затримка доставлення геокоординат від кур'єра до клієнта становила 187 мс (із гарантованим цільовим показником < 1 с), що доводить високу експлуатаційну ефективність розробленої архітектури.

Охорона праці та безпека життєдіяльності. У межах виконання розділу з охорони праці формалізовано покроковий алгоритм надання долікарської допомоги у разі ураження електричним струмом під час роботи з комп'ютерною технікою. На основі психофізіологічних критеріїв розраховано раціональний режим праці та відпочинку розробника, який передбачає 6 регламентованих перерв тривалістю по 15 хвилин (сумарно 90 хвилин за 8-годинну зміну, що становить 18,75 % робочого часу), що забезпечує збереження високої працездатності та профілактику професійних захворювань.

Практичне значення одержаних результатів полягає у створенні повністю готової до впровадження програмної екосистеми для логістичних служб та e-commerce підприємств. На відміну від закритих хмарних SaaS-платформ, розроблене рішення дозволяє розгортати інфраструктуру On-Premise (на власних серверах), що гарантує бізнесу повну незалежність від тарифної політики сторонніх провайдерів, відсутність транзакційних комісій та абсолютний контроль над конфіденційними даними клієнтів.

ПЕРЕЛІК ДЖЕРЕЛ

1. Sarder M. D. Logistics Transportation Systems. Amsterdam : Elsevier, 2021. 444 p.

2. Winkenbach M., Spinler S. Urban Last-Mile Delivery: Challenges and Innovations. Transportation Research Part E: Logistics and Transportation Review. 2021. Vol. 147. P. 102–124.

3. Grigorik I. High Performance Browser Networking. Sebastopol : O'Reilly Media, 2021. 400 p.

4. RFC 6455. The WebSocket Protocol / I. Fette, A. Melnikov. Internet Engineering Task Force. URL: https://datatracker.ietf.org/doc/html/rfc6455 (дата звернення: 14.03.2026).

5. Usage statistics of server-side programming languages for websites. W3Techs. URL: https://w3techs.com/technologies/overview/programming_language (дата звернення: 17.03.2026).

6. Nixon R. Learning PHP, MySQL & JavaScript. 6th ed. Sebastopol : O'Reilly Media, 2021. 826 p.

7. Unified Modeling Language Specification. Version 2.5.1. Object Management Group. URL: https://www.omg.org/spec/UML/2.5.1 (дата звернення: 20.03.2026).

8. MySQL 8.0 Reference Manual. Oracle Corporation. URL: https://dev.mysql.com/doc/refman/8.0/en/ (дата звернення: 22.03.2026).

9. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. Introduction to Algorithms. 4th ed. Cambridge : MIT Press, 2022. 1312 p.

10. Stauffer M. Laravel: Up & Running. 3rd ed. Sebastopol : O'Reilly Media, 2023. 522 p.

11. Ratchet: WebSockets for PHP. Documentation. URL: http://socketo.me/docs/ (дата звернення: 25.03.2026).

12. PHP 8.3 Documentation. The PHP Group. URL: https://www.php.net/docs.php (дата звернення: 25.03.2026).

13. Laravel 11.x Documentation. Laravel LLC. URL: https://laravel.com/docs/11.x (дата звернення: 26.03.2026).

14. Leaflet: an open-source JavaScript library for mobile-friendly interactive maps. Documentation. URL: https://leafletjs.com/reference.html (дата звернення: 28.03.2026).

15. WebSocket API. MDN Web Docs. Mozilla Foundation. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (дата звернення: 28.03.2026).

16. Geolocation API. MDN Web Docs. Mozilla Foundation. URL: https://developer.mozilla.org/en-US/docs/Web/API/Geolocation_API (дата звернення: 29.03.2026).

17. OpenStreetMap Wiki: Tile servers. OpenStreetMap Foundation. URL: https://wiki.openstreetmap.org/wiki/Tile_servers (дата звернення: 30.03.2026).

18. Krause J. Designing User Interfaces with a UX-First Approach. Berkeley : Apress, 2022. 318 p.

19. Marcotte E. Responsive Web Design. 3rd ed. New York : A Book Apart, 2021. 168 p.

20. Meyer E., Weyl E. CSS: The Definitive Guide. 5th ed. Sebastopol : O'Reilly Media, 2023. 1126 p.

21. Duckett J. JavaScript and jQuery: Interactive Front-End Web Development. 2nd ed. Indianapolis : Wiley, 2021. 640 p.

22. Готович В. А., Ралік І. Р. Програмне забезпечення на основі клієнт-серверної архітектури для обліку реалізації товарів в торгівлі // Матеріали XI Міжнародної науково-практичної конференції молодих учених та студентів „Актуальні задачі сучасних технологій “. – ТНТУ, 2022. – С. 126

23. Козак В. І., Готович В. А. Дослідження варіантів проектування інтерфейсу користувача в інформаційних інтерактивних аналітичних панелях // Матеріали XII Міжнародної науково-практичної конференції молодих учених та студентів „Актуальні задачі сучасних технологій“. – ФОП Паляниця В. А., 2023. – С. 385–386

24. Готович В. А., Мачужак А. В. Застосування методології CI/CD для автоматизації процесів тестування та розгортання програмного забезпечення // XI Міжнародна науково-практична конференція молодих учених та студентів „Актуальні задачі сучасних технологій“, 7-8 грудня 2022 року. – Т. : ТНТУ, 2022. – С. 131–132. – (Комп’ютерно-інформаційні технології та системи зв’язку)

25. Конспект лекцій з дисципліни «Програмування для мобільних пристроїв» для студентів денної форми навчання спеціальності 126 «Інформаційні системи та технології / Укладачі: Готович В. А., Михайлович Т. В. – Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя, 2020. – 216 с.

26. Shymchuk G., Lytvynenko I., Hromyak R., Lytvynenko S., Hotovych V. Gas Consumption Forecasting Using Machine Learning Methods and Taking into Account Climatic Indicators. The 1st International Workshop on Computer Information Technologies in Industry 4.0, CITI 2023. Ternopil 14 -16 June 2023. Vol. 3468, pp. 156-163. ISSN 1613-0073 URL: https://ceur-ws.org/Vol-3468/short8.pdf

27. Шаховська Н. Б., Болюбаш Ю. Я. Організація баз даних та знань : навчальний посібник. Львів : Видавництво Львівської політехніки, 2021. 360 с.

28. Литвин В. В., Висоцька В. А. Проєктування інформаційних систем : підручник. Львів : Новий Світ-2000, 2021. 380 с.

29. Пасічник В. В., Шаховська Н. Б. Сховища даних та вебтехнології : навчальний посібник. Львів : Магнолія 2006, 2022. 496 с.

30. OWASP Top 10:2021. The Open Worldwide Application Security Project. URL: https://owasp.org/Top10/ (дата звернення: 05.04.2026).

31. Hoffman A. Web Application Security: Exploitation and Countermeasures. 2nd ed. Sebastopol : O'Reilly Media, 2024. 444 p.

32. Stuttard D., Pinto M. The Web Application Hacker's Handbook. 2nd ed. Indianapolis : Wiley, 2021. 912 p.

33. Richards M., Ford N. Fundamentals of Software Architecture: An Engineering Approach. Sebastopol : O'Reilly Media, 2021. 432 p.

34. Дудикевич В. Б., Опірський І. Р. Безпека інформаційно-комунікаційних систем : навчальний посібник. Львів : Видавництво Львівської політехніки, 2022. 264 с.

35. Безпека життєдіяльності, основи охорони праці : навчально-методичний посібник до практичних занять для студентів освітнього ступеня «бакалавр» усіх спеціальностей та форм навчання / уклад. : О. Я. Гурик, І. Б. Окіпний, В. С. Сенчишин, С. Ю. Мариненко, О. І. Король. Тернопіль : ТНТУ імені Івана Пулюя, 2025. 123 с.

36. Методичні вказівки для написання розділу «Безпека життєдіяльності, основи охорони праці» в кваліфікаційних роботах здобувачів освітнього рівня «бакалавр» / уклад. : О. Я. Гурик, І. Б. Окіпний. Тернопіль : ТНТУ імені Івана Пулюя, 2021. 20 с.

37. Ткачук К. Н., Халімовський М. О., Зацарний В. В. Охорона праці : підручник. Київ : Основа, 2021. 456 с.

38. Желібо Є. П., Заверуха Н. М., Зацарний В. В. Безпека життєдіяльності : навчальний посібник. Київ : Каравела, 2021. 344 с.

39. НПАОП 0.00-7.15-18. Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями : наказ Мінсоцполітики України від 14.02.2018 № 207. URL: https://zakon.rada.gov.ua/laws/show/z0508-18 (дата звернення: 20.04.2026).

ДОДАТКИ

Додаток А

SQL-скрипт створення бази даних вебплатформи

```
CREATE DATABASE IF NOT EXISTS order_tracking
  CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
USE order_tracking;

CREATE TABLE users (
  id          BIGINT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  name        VARCHAR(100)  NOT NULL,
  email       VARCHAR(150)  NOT NULL UNIQUE,
  password    VARCHAR(255)  NOT NULL,
  phone       VARCHAR(20)   NULL,
  role        ENUM('client','courier','admin') NOT NULL DEFAULT
'client',
  is_active   TINYINT(1)    NOT NULL DEFAULT 1,
  created_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP
) ENGINE=InnoDB;

CREATE TABLE statuses (
  id          TINYINT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  code        VARCHAR(30) NOT NULL UNIQUE,
  title       VARCHAR(60) NOT NULL,
  color       CHAR(7)     NOT NULL,
  sort_order  TINYINT UNSIGNED NOT NULL
) ENGINE=InnoDB;

INSERT INTO statuses (code, title, color, sort_order) VALUES
  ('new',        'Нове',               '#9DC88D', 1),
  ('confirmed',  'Підтверджене',       '#9DC88D', 2),
  ('assembling', 'Комплектується',     '#4D774E', 3),
  ('assigned',   'Передане кур''єрові', '#4D774E', 4),
  ('in_transit', 'В дорозі',           '#164A41', 5),
  ('delivered',  'Доставлене',         '#F1B24A', 6),
  ('cancelled',  'Скасоване',          '#888888', 7);

CREATE TABLE orders (
  id            BIGINT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  client_id     BIGINT UNSIGNED NOT NULL,
  courier_id    BIGINT UNSIGNED NULL,
  status_id     TINYINT UNSIGNED NOT NULL,
  items_json    JSON NOT NULL,
  origin_lat    DECIMAL(10,7) NOT NULL,
  origin_lng    DECIMAL(10,7) NOT NULL,
  dest_lat      DECIMAL(10,7) NOT NULL,
  dest_lng      DECIMAL(10,7) NOT NULL,
  dest_address  VARCHAR(255) NOT NULL,
  total_price   DECIMAL(10,2) NOT NULL,
  route_json    JSON NULL,
  eta_minutes   SMALLINT UNSIGNED NULL,
  channel_token CHAR(64) NULL,
  created_at    TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  delivered_at  TIMESTAMP NULL,
```

```
  CONSTRAINT fk_orders_client  FOREIGN KEY (client_id)  REFERENCES
users(id)    ON DELETE RESTRICT,
  CONSTRAINT fk_orders_courier FOREIGN KEY (courier_id) REFERENCES
users(id)    ON DELETE RESTRICT,
  CONSTRAINT fk_orders_status  FOREIGN KEY (status_id)  REFERENCES
statuses(id) ON DELETE RESTRICT,
  INDEX idx_orders_client  (client_id, created_at),
  INDEX idx_orders_courier (courier_id, status_id)
) ENGINE=InnoDB;

CREATE TABLE locations (
  id          BIGINT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  order_id    BIGINT UNSIGNED NOT NULL,
  courier_id  BIGINT UNSIGNED NOT NULL,
  latitude    DECIMAL(10,7) NOT NULL,
  longitude   DECIMAL(10,7) NOT NULL,
  speed_kmh   DECIMAL(5,2) NULL,
  recorded_at TIMESTAMP(3) NOT NULL,
  CONSTRAINT fk_loc_order   FOREIGN KEY (order_id)   REFERENCES
orders(id) ON DELETE RESTRICT,
  CONSTRAINT fk_loc_courier FOREIGN KEY (courier_id) REFERENCES
users(id)  ON DELETE RESTRICT,
  INDEX idx_loc_trail (order_id, recorded_at)
) ENGINE=InnoDB;

CREATE TABLE status_history (
  id          BIGINT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  order_id    BIGINT UNSIGNED NOT NULL,
  status_id   TINYINT UNSIGNED NOT NULL,
  changed_by  BIGINT UNSIGNED NOT NULL,
  changed_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  CONSTRAINT fk_hist_order  FOREIGN KEY (order_id)   REFERENCES
orders(id)   ON DELETE RESTRICT,
  CONSTRAINT fk_hist_status FOREIGN KEY (status_id)  REFERENCES
statuses(id) ON DELETE RESTRICT,
  CONSTRAINT fk_hist_user   FOREIGN KEY (changed_by) REFERENCES
users(id)    ON DELETE RESTRICT,
  INDEX idx_hist_order (order_id, changed_at)
) ENGINE=InnoDB;
```

Додаток Б

Сервіс побудови маршруту за алгоритмом Дейкстри

```
<?php

namespace App\Services;

final class RouteBuilder
{
    /** @var array<int, array<int, float>> список суміжності */
    private array $graph;
    /** @var array<int, array{0: float, 1: float}> координати */
    private array $nodes;

    public function __construct(GraphRepository $repo)
    {
        [$this->graph, $this->nodes] = $repo->loadCityGraph();
    }

    public function shortestPath(array $from,
                                 array $to): RouteResult
    {
        $s = $this->nearestNode($from);
        $t = $this->nearestNode($to);

        $dist = array_fill_keys(array_keys($this->graph), INF);
        $prev = [];
        $dist[$s] = 0.0;

        $pq = new \SplPriorityQueue();
        $pq->setExtractFlags(\SplPriorityQueue::EXTR_DATA);
        $pq->insert($s, 0);

        $visited = [];
        while (!$pq->isEmpty()) {
            $u = $pq->extract();
            if (isset($visited[$u])) continue;
            $visited[$u] = true;
            if ($u === $t) break;          // ціль досягнуто

            foreach ($this->graph[$u] as $v => $w) {
                $alt = $dist[$u] + $w;     // релаксація (2.1)
                if ($alt < $dist[$v]) {
                    $dist[$v] = $alt;
                    $prev[$v] = $u;
                    $pq->insert($v, -$alt);
                }
            }
        }

        $path = [];
        for ($v = $t; isset($prev[$v]); $v = $prev[$v]) {
            array_unshift($path, $this->nodes[$v]);
```

```
        }
        array_unshift($path, $this->nodes[$s]);

        return new RouteResult(
            points: $path,
            lengthKm: round($dist[$t] / 1000, 2),
            etaMinutes: (int)ceil($dist[$t] / 1000
                                  / self::AVG_SPEED_KMH * 60),
        );
    }

    private const AVG_SPEED_KMH = 18.0; // велокур'єр у місті

    private function nearestNode(array $point): int
    {
        $best = null; $bestD = INF;
        foreach ($this->nodes as $id => [$lat, $lng]) {
            $d = ($lat - $point[0]) ** 2
               + ($lng - $point[1]) ** 2;
            if ($d < $bestD) { $bestD = $d; $best = $id; }
        }
        return $best;
    }
}
```

Додаток В

Серверна логіка опрацювання замовлень

```
<?php
// JSON-API вебплатформи. Маршрутизація за параметром ?action=...
require_once __DIR__ . '/../app/helpers.php';
require_once __DIR__ . '/../app/ws_publish.php';

$action = $_GET['action'] ?? '';
$user   = currentUser();
if (!$user) {
    jsonOut(['message' => 'Потрібна автентифікація.'], 401);
}

switch ($action) {

    // Перелік замовлень поточного користувача (за роллю).
    case 'orders':
        if ($user['role'] === 'client') {
            $st = db()->prepare(
                'SELECT o.*, s.code AS s_code, s.title AS s_title,
s.color AS s_color,
                        s.sort_order AS s_order, c.name AS
courier_name
                 FROM orders o JOIN statuses s ON s.id =
o.status_id
                 LEFT JOIN users c ON c.id = o.courier_id
                 WHERE o.client_id = ? ORDER BY o.created_at
DESC');
            $st->execute([$user['id']]);
        } elseif ($user['role'] === 'courier') {
            $st = db()->prepare(
                'SELECT o.*, s.code AS s_code, s.title AS s_title,
s.color AS s_color,
                        s.sort_order AS s_order, cl.name AS
client_name, cl.phone AS client_phone
                 FROM orders o JOIN statuses s ON s.id =
o.status_id
                 LEFT JOIN users cl ON cl.id = o.client_id
                 WHERE o.courier_id = ?
                   AND s.code NOT IN ("delivered","cancelled")
                 ORDER BY o.created_at');
            $st->execute([$user['id']]);
        } else {
            $st = db()->query(
                'SELECT o.*, s.code AS s_code, s.title AS s_title,
s.color AS s_color,
                        s.sort_order AS s_order, c.name AS
courier_name
                 FROM orders o JOIN statuses s ON s.id =
o.status_id
                 LEFT JOIN users c ON c.id = o.courier_id
                 ORDER BY o.created_at DESC');
        }
        jsonOut($st->fetchAll());
```

```
    // Дані для сторінки відстеження: замовлення, траєкторія,
токен каналу.
    case 'track':
        $id = (int) ($_GET['id'] ?? 0);
        $st = db()->prepare(
            'SELECT o.*, s.code AS s_code, s.title AS s_title,
s.color AS s_color,
                    s.sort_order AS s_order, c.name AS
courier_name
             FROM orders o JOIN statuses s ON s.id = o.status_id
             LEFT JOIN users c ON c.id = o.courier_id WHERE o.id =
?');
        $st->execute([$id]);
        $order = $st->fetch();
        if (!$order) {
            jsonOut(['message' => 'Замовлення не знайдено.'],
404);
        }
        if ($user['role'] === 'client' && (int)
$order['client_id'] !== (int) $user['id']) {
            jsonOut(['message' => 'Доступ заборонено.'], 403);
        }
        $tr = db()->prepare(
            'SELECT latitude, longitude, speed_kmh, recorded_at
             FROM locations WHERE order_id = ? ORDER BY
recorded_at');
        $tr->execute([$id]);
        jsonOut([
            'order' => $order,
            'trail' => $tr->fetchAll(),
            'token' => makeChannelToken($id),
        ]);

    // Створення замовлення адміністратором.
    case 'create':
        requireRole('admin');
        $d = jsonIn();
        $sNew = statusByCode('new');
        $pdo  = db();
        $pdo->beginTransaction();
        try {
            $st = $pdo->prepare(
                'INSERT INTO orders
                    (client_id, status_id, items_json, origin_lat,
origin_lng,
                     dest_lat, dest_lng, dest_address,
total_price, eta_minutes, created_at)
                 VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, NOW())');
            $st->execute([
                (int) $d['client_id'], $sNew['id'],
                json_encode($d['items'] ?? [],
JSON_UNESCAPED_UNICODE),
                $d['origin']['lat'], $d['origin']['lng'],
                $d['dest']['lat'],   $d['dest']['lng'],
                $d['dest_address'] ?? '', $d['total_price'] ?? 0,
```

```
                $d['eta_minutes'] ?? null,
            ]);
            $orderId = (int) $pdo->lastInsertId();
            logStatus($orderId, 'new', (int) $user['id']);
            $pdo->commit();
        } catch (Throwable $ex) {
            $pdo->rollBack();
            jsonOut(['message' => 'Помилка створення: ' . $ex-
>getMessage()], 422);
        }
        jsonOut(['ok' => true, 'order_id' => $orderId], 201);

    // Призначення кур'єра замовленню (статус assigned).
    case 'assign':
        requireRole('admin');
        $d  = jsonIn();
        $id = (int) ($d['order_id'] ?? 0);
        $st = db()->prepare('SELECT s.code FROM orders o
                             JOIN statuses s ON s.id = o.status_id
WHERE o.id = ?');
        $st->execute([$id]);
        $cur = $st->fetchColumn();
        if (!$cur || !canTransit($cur, 'assigned')) {
            jsonOut(['message' => 'Замовлення не готове до
призначення кур\'єра.'], 422);
        }
        $sAssigned = statusByCode('assigned');
        $pdo = db();
        $pdo->beginTransaction();
        try {
            $u = $pdo->prepare('UPDATE orders SET courier_id = ?,
status_id = ? WHERE id = ?');
            $u->execute([(int) $d['courier_id'], $sAssigned['id'],
$id]);
            logStatus($id, 'assigned', (int) $user['id']);
            $pdo->commit();
        } catch (Throwable $ex) {
            $pdo->rollBack();
            jsonOut(['message' => $ex->getMessage()], 422);
        }
        wsPublish($id, [
            'type'   => 'status_changed',
            'status' => ['code' => 'assigned', 'title' =>
$sAssigned['title'],
                         'color' => $sAssigned['color'],
'sort_order' => $sAssigned['sort_order']],
        ]);
        jsonOut(['ok' => true]);

    // Зміна статусу кур'єром або адміністратором (трирівневий
захист).
    case 'status':
        $u = requireRole('courier', 'admin');
        $d  = jsonIn();
        $id = (int) ($d['order_id'] ?? 0);
        $to = (string) ($d['status'] ?? '');
```

```
        $st = db()->prepare('SELECT o.courier_id, s.code AS cur
                             FROM orders o JOIN statuses s ON s.id
= o.status_id
                             WHERE o.id = ?');
        $st->execute([$id]);
        $row = $st->fetch();
        if (!$row) {
            jsonOut(['message' => 'Замовлення не знайдено.'],
404);
        }
        // Рівень 1 – належність замовлення кур'єрові
        if ($u['role'] === 'courier' && (int) $row['courier_id']
!== (int) $u['id']) {
            jsonOut(['message' => 'Замовлення призначене іншому
кур\'єрові.'], 403);
        }
        // Рівень 2 – допустимість переходу скінченного автомата
        if (!canTransit($row['cur'], $to)) {
            jsonOut(['message' => 'Недопустимий перехід життєвого
циклу.'], 422);
        }
        $sTo = statusByCode($to);
        // Рівень 3 – атомарне оновлення у межах транзакції
        $pdo = db();
        $pdo->beginTransaction();
        try {
            if ($to === 'delivered') {
                $up = $pdo->prepare('UPDATE orders SET status_id =
?, delivered_at = NOW() WHERE id = ?');
            } else {
                $up = $pdo->prepare('UPDATE orders SET status_id =
? WHERE id = ?');
            }
            $up->execute([$sTo['id'], $id]);
            logStatus($id, $to, (int) $u['id']);
            $pdo->commit();
        } catch (Throwable $ex) {
            $pdo->rollBack();
            jsonOut(['message' => $ex->getMessage()], 422);
        }
        // Миттєва трансляція зміни статусу підписникам каналу
        wsPublish($id, [
            'type'   => 'status_changed',
            'status' => ['code' => $sTo['code'], 'title' =>
$sTo['title'],
                         'color' => $sTo['color'], 'sort_order' =>
$sTo['sort_order']],
        ]);
        jsonOut(['ok' => true]);

    // Усі активні доставлення для спільної мапи адміністратора.
    case 'active':
        requireRole('admin');
        $st = db()->query(
            'SELECT o.id, o.dest_lat, o.dest_lng, o.dest_address,
```

```
                    s.code AS s_code, s.title AS s_title, s.color
AS s_color,
                    c.name AS courier_name
             FROM orders o JOIN statuses s ON s.id = o.status_id
             LEFT JOIN users c ON c.id = o.courier_id
             WHERE s.code IN ("assigned","in_transit")');
        jsonOut($st->fetchAll());

    // Зведені показники дня.
    case 'stats':
        requireRole('admin');
        $created   = db()->query('SELECT COUNT(*) FROM orders
WHERE DATE(created_at) = CURDATE()')->fetchColumn();
        $transit   = db()->query('SELECT COUNT(*) FROM orders o
JOIN statuses s ON s.id=o.status_id WHERE s.code="in_transit"')-
>fetchColumn();
        $delivered = db()->query('SELECT COUNT(*) FROM orders
WHERE DATE(delivered_at) = CURDATE()')->fetchColumn();
        $avg       = db()->query('SELECT AVG(TIMESTAMPDIFF(MINUTE,
created_at, delivered_at)) FROM orders WHERE
DATE(delivered_at)=CURDATE()')->fetchColumn();
        jsonOut([
            'created'   => (int) $created,
            'transit'   => (int) $transit,
            'delivered' => (int) $delivered,
            'avg'       => (int) round((float) $avg),
        ]);

    default:
        jsonOut(['message' => 'Невідома дія.'], 404);
}
```

Додаток Д

WebSocket-сервер трансляції координат

```
<?php
// WebSocket-сервер трансляції координат на «голих» сокетах PHP.
// Реалізує рукостискання та кадрування за RFC 6455 без жодних
бібліотек.
require_once __DIR__ . '/../app/db.php';

$cfg = (require __DIR__ . '/../app/config.php')['ws'];

$server =
stream_socket_server("tcp://{$cfg['host']}:{$cfg['port']}",
$errno, $errstr);
if (!$server) {
    fwrite(STDERR, "Не вдалося відкрити порт {$cfg['port']}:
$errstr\n");
    exit(1);
}
stream_set_blocking($server, false);
echo "WebSocket-сервер запущено на порту {$cfg['port']}\n";

$clients  = [];   // resourceId => ['sock'=>res, 'hs'=>bool,
'order'=>int|null]
$channels = [];   // order_id   => [resourceId, ...]

function rid($sock): int { return (int) $sock; }

// Рукостискання WebSocket
function wsHandshake($sock, string $req): bool
{
    if (!preg_match('#Sec-WebSocket-Key:\s*(.+)\r\n#i', $req, $m))
{
        return false;
    }
    $accept = base64_encode(sha1(trim($m[1]) . '258EAFA5-E914-
47DA-95CA-C5AB0DC85B11', true));
    fwrite($sock, "HTTP/1.1 101 Switching Protocols\r\n"
        . "Upgrade: websocket\r\nConnection: Upgrade\r\n"
        . "Sec-WebSocket-Accept: $accept\r\n\r\n");
    return true;
}

// Декодування вхідного кадру (клієнт завжди маскує)
function wsDecode(string $data): ?string
{
    if (strlen($data) < 2) return null;
    $len = ord($data[1]) & 127;
    $offset = 2;
    if ($len === 126) { $offset = 4; } elseif ($len === 127) {
$offset = 10; }
    $masks = substr($data, $offset, 4);
    $offset += 4;
    $payload = substr($data, $offset);
    $text = '';
```

```
    for ($i = 0, $n = strlen($payload); $i < $n; $i++) {
        $text .= $payload[$i] ^ $masks[$i % 4];
    }
    return $text;
}

// Кодування вихідного кадру (сервер не маскує)
function wsEncode(string $payload): string
{
    $len = strlen($payload);
    $head = "\x81";
    if ($len < 126)        { $head .= chr($len); }
    elseif ($len < 65536)  { $head .= chr(126) . pack('n', $len);
}
    else                   { $head .= chr(127) . pack('J', $len);
}
    return $head . $payload;
}

function tokenValid(int $orderId, string $token, string $role =
'client'): bool
{
    $st = db()->prepare('SELECT channel_token FROM orders WHERE id
= ?');
    $st->execute([$orderId]);
    $stored = (string) $st->fetchColumn();
    return $stored !== '' && hash_equals($stored, hash('sha256',
$token . $role));
}

function broadcast(array &$clients, array &$channels, int
$orderId, array $payload): void
{
    $frame = wsEncode(json_encode($payload,
JSON_UNESCAPED_UNICODE));
    foreach ($channels[$orderId] ?? [] as $id) {
        if (isset($clients[$id])) {
            @fwrite($clients[$id]['sock'], $frame);
        }
    }
}

$internalKey = $cfg['internal_key'];

// Головний цикл обслуговування з'єднань
while (true) {
    $read = [$server];
    foreach ($clients as $c) { $read[] = $c['sock']; }
    $write = $except = null;

    if (@stream_select($read, $write, $except, 0, 200000) < 1) {
        continue;
    }

    if (in_array($server, $read, true)) {
        $sock = @stream_socket_accept($server, 0);
```

```
        if ($sock) {
            stream_set_blocking($sock, false);
            $clients[rid($sock)] = ['sock' => $sock, 'hs' =>
false, 'order' => null];
        }
        unset($read[array_search($server, $read, true)]);
    }

    foreach ($read as $sock) {
        $id   = rid($sock);
        $data = @fread($sock, 8192);

        if ($data === '' || $data === false) {
            $ord = $clients[$id]['order'] ?? null;
            if ($ord !== null && isset($channels[$ord])) {
                $channels[$ord] =
array_values(array_diff($channels[$ord], [$id]));
            }
            @fclose($sock);
            unset($clients[$id]);
            continue;
        }

        if (!$clients[$id]['hs']) {
            if (wsHandshake($sock, $data)) { $clients[$id]['hs'] =
true; }
            continue;
        }

        $msg = wsDecode($data);
        if ($msg === null) continue;
        $d = json_decode($msg, true);
        if (!is_array($d) || empty($d['type'])) continue;

        switch ($d['type']) {
            case 'subscribe':
                $orderId = (int) ($d['order_id'] ?? 0);
                if (!tokenValid($orderId, (string) ($d['token'] ??
'''))) {
                    @fwrite($sock, wsEncode(json_encode(
                        ['type' => 'error', 'message' =>
'Невалідний токен каналу'],
                        JSON_UNESCAPED_UNICODE)));
                    break;
                }
                $clients[$id]['order'] = $orderId;
                $channels[$orderId][] = $id;
                @fwrite($sock, wsEncode(json_encode(
                    ['type' => 'subscribed', 'order_id' =>
$orderId])));
                break;

            case 'location':
                $orderId = (int) ($d['order_id'] ?? 0);
                if (!tokenValid($orderId, (string) ($d['token'] ??
''), 'courier')) break;
```

```
                $lat = round((float) $d['lat'], 7);
                $lng = round((float) $d['lng'], 7);
                $st = db()->prepare(
                    'INSERT INTO locations
                        (order_id, courier_id, latitude,
longitude, speed_kmh, recorded_at)
                     VALUES (?, ?, ?, ?, ?, NOW(3))');
                $st->execute([$orderId, (int) $d['courier_id'],
$lat, $lng, $d['speed'] ?? null]);
                broadcast($clients, $channels, $orderId, [
                    'type' => 'location', 'lat' => $lat, 'lng' =>
$lng,
                    'speed' => $d['speed'] ?? null, 'time' =>
date(DATE_ATOM),
                ]);
                break;

            case 'publish': // внутрішня подія від HTTP-частини
                if (!hash_equals($internalKey, (string) ($d['key']
?? ''))) break;
                broadcast($clients, $channels, (int)
$d['order_id'], (array) $d['payload']);
                break;
        }
    }
}
```

Додаток Е

Клієнтська частина: сценарій відстеження

```
// Сторінка відстеження клієнта: мапа Leaflet + WebSocket
const cfg = window.TRACK;
let map, trail, courierMarker = null, ws, retry = 0;

const STAGES = [

{code:'new',title:'Нове',order:1},{code:'confirmed',title:'Підтвер
джене',order:2},

{code:'assembling',title:'Комплектується',order:3},{code:'assigned
',title:'Передане',order:4},
  {code:'in_transit',title:'В
дорозі',order:5},{code:'delivered',title:'Доставлене',order:6}
];

fetch('/api.php?action=track&id=' + cfg.orderId).then(r =>
r.json()).then(init);

function init(data) {
  const o = data.order;
  document.getElementById('eta').textContent = o.eta_minutes ? '~'
+ o.eta_minutes + ' хв' : '–';
  renderSteps(+o.s_order);

  map = L.map('map').setView([+o.dest_lat, +o.dest_lng], 14);
  L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png',
    {maxZoom:19, attribution:'&copy; OpenStreetMap'}).addTo(map);
  L.marker([+o.dest_lat, +o.dest_lng],
{icon:L.divIcon({className:'pin pin--dest'})})
   .addTo(map).bindPopup('Пункт призначення');

  const pts = (data.trail || []).map(p => [+p.latitude,
+p.longitude]);
  trail = L.polyline(pts, {color:'#164A41', weight:4,
opacity:.85}).addTo(map);
  if (pts.length) addCourier(pts[pts.length - 1]);

  cfg.token = data.token;
  connect();
}

function renderSteps(current) {
  document.getElementById('steps').innerHTML = STAGES.map(s =>
    `<div class="step ${s.order <= current ? 'step--done':''}"
data-order="${s.order}">${s.title}</div>`
  ).join('');
}

function addCourier(pos) {
  courierMarker = L.marker(pos, {icon:L.divIcon({className:'pin
pin--courier'})}).addTo(map);
}
```

```
function connect() {
  ws = new WebSocket(cfg.wsUrl);
  ws.onopen = () => {
    retry = 0;
    ws.send(JSON.stringify({type:'subscribe',
order_id:cfg.orderId, token:cfg.token}));
    badge('online');
  };
  ws.onmessage = e => {
    const m = JSON.parse(e.data);
    if (m.type === 'location') onLocation(m);
    if (m.type === 'status_changed') onStatus(m);
  };
  ws.onclose = () => { badge('offline'); setTimeout(connect,
Math.min(1000*2**retry++, 15000)); };
}

function onLocation({lat, lng, speed, time}) {
  const pos = [lat, lng];
  trail.addLatLng(pos);
  courierMarker ? courierMarker.setLatLng(pos) : addCourier(pos);
  map.panTo(pos, {animate:true, duration:.5});
  document.getElementById('speed').textContent = speed ?
(+speed).toFixed(0)+' км/год' : '–';
  document.getElementById('updated').textContent = new
Date(time).toLocaleTimeString('uk-UA');
}

function onStatus({status}) {
  document.querySelectorAll('.step').forEach(el =>
    el.classList.toggle('step--done', +el.dataset.order <=
+status.sort_order));
  if (status.code === 'delivered') { alert('Замовлення доставлено.
Смачного дня!'); ws.close(); }
}

function badge(state) {
  const b = document.getElementById('ws-badge');
  b.className = 'badge badge--' + state;
  b.textContent = state === 'online' ? 'у реальному часі' :
'перепідключення…';
}
```