ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

КВАЛІФІКАЦІЙНА РОБОТА

Тема: «Вебзастосунок ''Delivra'' для реалізації логістичних функцій з використанням алгоритму пошуку шляху»

Ступінь вищої освіти – бакалавр
Спеціальність – 122 «Комп'ютерні науки»
Освітня програма «Комп'ютерні науки»

ПОЯСНЮВАЛЬНА ЗАПИСКА

Виконав: здобувач 4 курсу
групи КН-22
Тальвін ДЖАПАРОВ
Керівник: викладач кафедри інформаційного
менеджменту, математики та статистики
Олег МУШИНСЬКИЙ

Засвідчую, що кваліфікаційна робота оформлена відповідно до ДСТУ 3008:2015 та не містить запозичень з праць інших авторів без відповідних посилань.

Здобувач: _____
(підпис)

м. Київ – 2026 рік

ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

ЗАТВЕРДЖУЮ:
завідувач кафедри комп'ютерних наук
_____ Сергій МІЧКІВСЬКИЙ
«__» _____ 20__ р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Джапаров Тальвін Рефатович

Тема роботи	Вебзастосунок «Delivra» для реалізації логістичних функцій з використанням алгоритму пошуку шляху
Номер та дата наказу про затвердження теми	№ 133-7 від 22.12.2025 р.
Коротка постановка завдання	Розробити вебзастосунок для логістичних компаній, який автоматизує планування маршрутів, розподіл завдань, відстеження водіїв і формування звітів з використанням алгоритму пошуку шляху
Посилання на джерела інформації (не більше п'яти найменувань, які рекомендує науковий керівник)	1. Гринченко М., Куценко Д. Моделі бізнес-процесів медичного центру для підвищення ефективності прийняття рішень. Сучасний стан наукових досліджень та технологій в промисловості, 2025 №34 DOI: https://doi.org/10.30837/2522-9818.2025.4.005 2. Бульба С. С., Соловйова О. І., Семеренко Ю. О. Дослідження алгоритмів пошуку оптимального шляху. Системи управління, навігації та зв'язку. Збірник наукових праць. 2024. Т. 2, № 76. С. 67–69. DOI: https://doi.org/10.26906/sunz.2024.2.067 3. Qu L., Li H. Analysis of distribution path optimization algorithm based on big data technology. Journal of King Saud University - Science. 2022. P. 102019. DOI: https://doi.org/10.1016/j.jksus.2022.102019
Вимоги до кваліфікаційної роботи	Кваліфікаційна робота має передбачити теоретичне, системотехнічне або експериментальне дослідження складного спеціалізованого завдання або практичної проблеми в галузі комп'ютерних наук, яке характеризується комплексністю та невизначеністю умов і потребує застосування теорій і методів інформаційних технологій.

Дата видачі завдання 27 грудня 2025 р.

Керівник Олег МУШИНСЬКИЙ

Здобувач освітнього ступеня бакалавра Тальвін ДЖАПАРОВ

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання	Примітка
	Підготовчий етап		
1	Вибір напряму дослідження	01.12.2025 р.	Виконано
2	Формування теми та призначення керівника	15.12.2025 р.	Виконано
3	Затвердження теми кваліфікаційної роботи	22.12.2025 р.	Виконано
4	Затвердження завдання на кваліфікаційну роботу	26.12.2025 р.	Виконано
	Основний етап		
5	Розробка концепції кваліфікаційної роботи	23.01.2026 р.	Виконано
6	Підбір та вивчення джерел інформації з напряму дослідження. Огляд існуючих аналогів	13.03.2026 р.	Виконано
7	Затвердження розширеної постановки завдання. Підготовка та подання керівникові розділу 1 кваліфікаційної роботи	20.03.2026 р.	Виконано
8	Проєктування. Підготовка та подання керівникові розділу 2 кваліфікаційної роботи	27.03.2026 р.	Виконано
9	Підготовка доповіді для експертизи стану виконання кваліфікаційної роботи (проміжний контроль)	30.03-04.04.2026 р.	Виконано
10	Реалізація. Підготовка та подання керівникові розділу 3 кваліфікаційної роботи	06.04.2026 р.	Виконано
11	Підготовка та подання керівнику першого варіанту всієї кваліфікаційної роботи	13.04.2026 р.	Виконано
12	Доопрацювання кваліфікаційної роботи з урахуванням зауважень керівника та представлення керівникові доопрацьованого варіанту кваліфікаційної роботи	20.04.2026 р.	Виконано
	Завершальний етап		
13	Представлення рукопису для перевірки на плагіат	27.04-03.05.2026 р.	Виконано
14	Підготовка презентації та доповіді на передзахист	04.05-10.05.2026 р.	Виконано
15	Передзахист кваліфікаційної роботи	11.05-15.05.2026 р.	Виконано
16	Доопрацювання роботи за результатами передзахисту	18.05-05.06.2026 р.	Виконано
17	Експертиза роботи керівником та зовнішнім експертом	08.06-14.06.2026 р.	Виконано
18	Доопрацювання доповіді та презентації для захисту	08.06-14.06.2026 р.	Виконано
19	Захист кваліфікаційної роботи	15.06-21.06.2026 р.	Виконано

Керівник Олег МУШИНСЬКИЙ

Здобувач освітнього ступеня бакалавра Тальвін ДЖАПАРОВ

АННОТАЦІЯ

Джапаров Т.Р. Вебзастосунок «Delivra» для реалізації логістичних функцій з використанням алгоритму пошуку шляху

Пояснювальна записка кваліфікаційної роботи за спеціальністю 122 - Комп'ютерні науки (освітня програма - Комп'ютерні науки) СО Бакалавр. - ВНЗ "Університет економіки та права "КРОК", Навчально-науковий інститут інформаційних та комунікаційних технологій, кафедра комп'ютерних наук, Київ, 2026.

Розглянуто проблему сучасних логістичних систем на прикладі розробки логістичної платформи «Delivra». Розроблено логістичну систему «Delivra» з використанням фреймворків Spring для серверної частини, та бібліотеку «React» для клієнтської частини системи

Ключові слова: логістична система, Spring, алгоритм пошуку оптимального шляху.

Рис. 32. Бібліограф.: 13 найм.

Dzhaparov T.R. Web Application "Delivra" for Implementation of Logistics Functions Using Path-Finding Algorithm

Explanatory note of the qualification work in specialty 122 - Computer Science (educational program - Computer Science) Bachelor's degree. - Higher Educational Institution "University of Economics and Law 'KROK'", Educational and Scientific Institute of Information and Communication Technologies, Department of Computer Science, Kyiv, 2026.

The problem of modern logistics systems is examined through the example of developing the logistics platform "Delivra". The logistics system "Delivra" has been developed using Spring frameworks for the server-side and the React library for the client-side of the system.

Keywords: logistics system, Spring, optimal path-finding algorithm.

Fig. 32. References: 13 items.

ЗМІСТ

ВСТУП

Актуальність теми. Доставити товар до клієнта вчасно є дуже важливим завданням для компаній, адже це прямо впливає на репутацію компанії та її прибуток. Постійні зміни дорожніх умов і погана комунікація між водіями та диспетчерами є одними з найбільших проблем вчасної доставки та транспортної логістики в цілому. Ці проблеми не вирішені навіть сьогодні, що остаточно вказує на актуальність теми.

У сфері вантажних перевезень відсутня автоматизація процесів передачі інформації між диспетчерами та водіями. Зокрема:

- відсутність автоматичного призначення завдань водіям з урахуванням їхньої завантаженості та місцезнаходження;
- неможливість автоматичної побудови маршруту з урахуванням габаритних та вагових обмежень вантажного транспорту;
- відсутність алгоритмів оптимізації при необхідності доставки до декількох точок;
- використання загальних навігаційних систем (Google Maps), які не враховують специфіку вантажного транспорту.

Це призводить до затримок у доставці, перевитрат палива та зниження ефективності логістичних процесів.

Мета дослідження полягає в розробці логістичної системи «Delivra» з використанням алгоритму пошуку шляху та методу зваженої суми критеріїв.

Завдання дослідження полягають в наступному:

- опис домену логістики
- проаналізувати існуючі логістичні системи та визначити їхні сильні та слабкі сторони;
- спроектувати архітектуру логістичної системи «Delivra» з використанням діаграм проєктування;

- розробити логістичну платформу «Delivra», зокрема: розробка представлення для водія, окремого представлення для диспетчера та адміністратора системи.

Об'єктом дослідження логістичні процеси у сфері вантажних перевезень, зокрема побудова маршрутів, призначення завдань водіям, навігація та відстеження доставки.

Предмет дослідження методи та алгоритми оптимізації логістичних процесів, зокрема алгоритм рекомендації на основі зваженої суми критеріїв та алгоритми побудови маршрутів для вантажного транспорту.

Методи дослідження У роботі використано комплекс загальнонаукових, аналітичних і проєктно-технологічних методів. Для дослідження предметної області застосовано метод аналізу логістичних процесів, що дозволив визначити основні проблеми диспетчеризації, маршрутизації та взаємодії між водіями й диспетчерами. Для моделювання предметної області використано підхід Domain-Driven Design, на основі якого визначено ключові сутності, об’єкти-значення, доменні сервіси та взаємозв’язки між ними. Для функціонального моделювання процесів логістичної системи застосовано методологію IDEF0, що дала змогу формалізувати вхідні дані, керуючі впливи, механізми виконання та результати основних логістичних процесів. Для реалізації маршрутизації використано підходи й алгоритми пошуку шляху, які забезпечують побудову маршруту з урахуванням параметрів вантажного транспорту. Для автоматизованого вибору оптимального водія застосовано метод зваженої суми критеріїв, що дозволяє враховувати відстань до точки завантаження, поточну завантаженість водія та його досвід виконання доставок.

Практичне значення. Розроблена система дозволяє: зменшити час на планування маршрутів на 40-60% порівняно з ручним плануванням; знизити витрати палива за рахунок оптимізації послідовності доставки; уникнути штрафів та простоїв через врахування габаритних обмежень вантажного транспорту; автоматизувати процес призначення завдань водіям з урахуванням

їхньої завантаженості та місцезнаходження; підвищити якість сервісу та репутацію логістичної компанії завдяки своєчасній доставці.

Структура роботи. Кваліфікаційна робота складається зі вступу, трьох розділів, висновків та списку посилань (13 найменувань). Загальний обсяг пояснювальної записки 87 сторінок 32 рисунка та 3 таблиці

РОЗДІЛ 1
ПОСТАНОВКА ЗАВДАННЯ НА РОЗРОБКУ ЛОГІСТИЧНОЇ СИСТЕМИ «DELIVRA»

1.1 Предметна область

Транспортна логістика є одним з ключових елементів для бізнесу, адже це частина логістики, яка займається переміщенням товарів між постачальниками, складами, виробничими об’єктами та кінцевими споживачами.

Справа в тому, що вантажний транспорт має низку обмежень, через які йому не дозволено рухатися. Наприклад, потрібно слідкувати, щоб висота вантажного транспорту не перевищувала висоту мостів, які можуть потенційно зустрітися на шляху, або просто заборонено рухатися, через наявність районів де повинна бути тиша. У таких випадках на дорозі стоїть відповідний дорожній знак. Нажаль не завжди ці знаки, як і висота деяких мостів враховуються навігаторами при побудові маршруту. Існують багато випадків, коли вантажний транспорт зустрічає подібний знак на дорозі, або низький міст, але можливості розвернути транспорт немає. Це ускладнює рух на дорозі, створюючи затори та дорожньо-транспортні пригоди Навіть якщо є достатньо місця для маневру, вимушена зміна маршруту призводить до додаткових витрат пального, збільшення часу в дорозі та, відповідно, до зростання собівартості перевезення і погіршення економічних показників компаній.

Ще одним викликом в сфері вантажних перевезень є погана комунікація водія та диспетчера. Справа в тому, що у більшості компаній, які займаються доставкою товарів, не мають автоматизацію процесу передачі даних отримувача товару водію, та відсутні можливості для водія автоматично побудувати оптимальний за часом та витратою пального маршрут до отримувача. Водієві відправляють адресу доставки звичайним повідомленням

у месенджері. Це в свою чергу призводить до наступних ризиків які критично впливають на прибуток та швидкість доставки:

1. Ризики витоку чутливих даних клієнтів компанії, що у більшості випадків порушує політику конфіденційності компанії.

2. Ризики, що водій загубить, або випадково видалить чутливі дані, що призведе до зайвого турбування диспетчера.

3. Ризики, що диспетчер помилково відправить повідомлення іншому водію, або допустить помилку при набиранні повідомлення тощо.

Для комплексного аналізу предметної області та проєктування такої інформаційної системи доцільно застосувати підхід предметно-орієнтованого проєктування (Domain-Driven Design) [8].

Domain-Driven Design - це підхід до розробки програмного забезпечення, що ставить у центр уваги предметну область (domain) та бізнес-логіку, а не технічні деталі реалізації. Основна ідея DDD полягає в тому, що розробники та експерти предметної області (диспетчери, логісти) повинні використовувати єдину мову (ubiquitous language) для опису системи. У межах цього підходу відокремлено ключові сутності, що відображають основні об’єкти логістичної системи, зокрема, маршрут, завдання, водій, диспетчер, клієнт та локація:

1. Entities (Сутності) - об'єкти з унікальним ідентифікатором, які зберігають свою ідентичність протягом життєвого циклу:
    - DeliveryTask (Завдання доставки) - основна сутність з унікальним ID, статусом виконання, адресами завантаження та розвантаження;
    - ChatMessage – сутність з повідомленнями у чаті;
    - Company – сутність, яка виражає клієнта-компанію та ізолює дані компанії від інших користувачів;
    - NavigationSession – сутність, яка зберігає поточну сессію навігації користувача. Це потрібно для динамічного оновлення позиції користувача, та демонстрації цієї позиції диспетчеру;

- RefreshToken – сутність потрібна для оновлення сессії користувача. Якщо користувач наразі користується системою, то його JWT токен оновлюється, що надає можливість користувачу не входити повторно на платформу, якщо термін дії токену завершився;
- Role – сутність зберігає доступні ролі платформи;
- User – сутність зберігає користувача не залежно від ролі;

2. Value Objects (Об'єкти-значення) - незмінні об'єкти без унікального ідентифікатора, що визначаються своїми атрибутами:
   - Address (Адреса) - об'єкт, що містить вулицю, місто, GPS координати;
   - CargoParameters (Параметри вантажу) - вага, габарити, тип вантажу;
   - DriverScore (Оцінка водія) - результат роботи алгоритму WSM (відстань, завантаженість, досвід, загальний бал).
3. Domain Services (Доменні сервіси) - бізнес-логіка, що не належить до конкретної entity:
   - DriverRecommendationService - реалізація алгоритму WSM для автоматичного вибору оптимального водія;
   - NavigationService - інтеграція з HERE Maps Truck Routing API та оптимізація послідовності доставки;
   - HereApiService – перетворення адреси на координати для подальшої навігації;
   - EmailService – сервіс для надсилання повідомлень на електронну пошту;
   - ReportService – сервіс для формування та вивантаження звіту.
4. Repositories (Репозиторії) - абстракції для доступу до даних, що приховують деталі роботи з базою даних:

- DeliveryTaskRepository - методи для пошуку завдань за статусом, водієм, дедлайном;
- ChatMessageRepository – методи для пошуку повідомлень у чаті та маркування повідомлення як прочитане;
- CompanyRepository – методи для пошуку та рахування за певним критерієм компаній;
- NavigationSessionRepository – методи для пошуку навігаційних сессій;
- RefreshTokenRepository – методи для пошуку оновлених токенів;
- RoleRepository – методи для пошуку ролі;
- UserRepository – методи для пошуку користувачів за ролями, за доступністю водіїв та ін.;

5. Aggregates (Агрегати) - кластери пов'язаних entities та value objects, що розглядаються як єдине ціле для забезпечення консистентності даних:
   - TaskAggregate - включає Task, призначеного Driver, побудований Route та параметри CargoParameters, забезпечуючи цілісність даних при зміні статусу завдання.
6. Ubiquitous Language (Єдина мова) - використання однакових термінів в коді та в спілкуванні з експертами предметної області:
   - "Призначити водія" - assignDriver(task, driver);
   - "Побудувати маршрут" - buildRoute(task, vehicle);
   - "Розрахувати оцінку" - calculateScore(driver, task);
   - "Завдання" - логістичне завдання для водія, яке містить адресу та інші дані клієнта;

Застосування DDD дозволило:

- чітко відокремити бізнес-логіку (алгоритм WSM, оптимізація маршрутів) від технічних деталей (REST API, база даних);

- забезпечити відповідність коду реальним логістичним процесам;
- полегшити взаємодію між розробниками та експертами предметної області завдяки використанню єдиної термінології.

1.2 Огляд аналогів

Серед спеціалізованих навігаційних рішень для вантажного транспорту найбільш поширеними є:

Рисунок 1.1 - навігатор TomTom GO Professional

Джерело: [1].

TomTom GO Professional [1] - комерційна навігаційна система, яка враховує габаритні параметри транспортного засобу (висота, ширина, вага, довжина). Система дозволяє уникати маршрутів з низькими мостами, вузькими дорогами та обмеженнями для вантажного транспорту.

Недоліки TomTom GO Professional:

- висока вартість використання: одноразова оплата за карту країни становить €50-150, річна передплата на мобільний додаток - €155.88;

- необхідність ручного завантаження карт для кожної країни маршруту;
- відсутність актуальної інформації про дорожню ситуацію в режимі реального часу;
- оновлення карт відбувається не частіше одного разу на квартал, що не відповідає динаміці змін дорожньої інфраструктури (особливо в умовах воєнного стану);
- відсутність інтеграції з системами управління завданнями та диспетчеризації.

Переваги TomTom GO Professional:

- як спеціалізований додаток він добре виконує функцію навігації вантажівки;
- є як стаціонарні рішення так і мобільний додаток;
- зрозумілий інтерфейс користувача.

Рисунок 1.2 - навігатор Sygic Truck GPS Navigation

Джерело: [2].

Sygic Truck GPS Navigation [2] - мобільний навігаційний додаток з підтримкою офлайн-карт та врахуванням параметрів вантажівки. Вартість передплати становить €9.99/місяць або €49.99/рік.

Недоліки:

- обмежена точність габаритних обмежень (база даних неповна);
- відсутність автоматичної синхронізації з логістичними системами;

Переваги:

- спеціалізований навігатор для вантажівок;
- зрозуміло та чітко відображена візуальна частина мапи.

Висновок: існуючі навігаційні системи зосереджені виключно на побудові маршруту, але не інтегровані з процесами управління завданнями, призначення водіїв та комунікації між диспетчерами і водіями.

Однак окрім навігаційних систем, існують платформи для управління доставкою та відстеження:

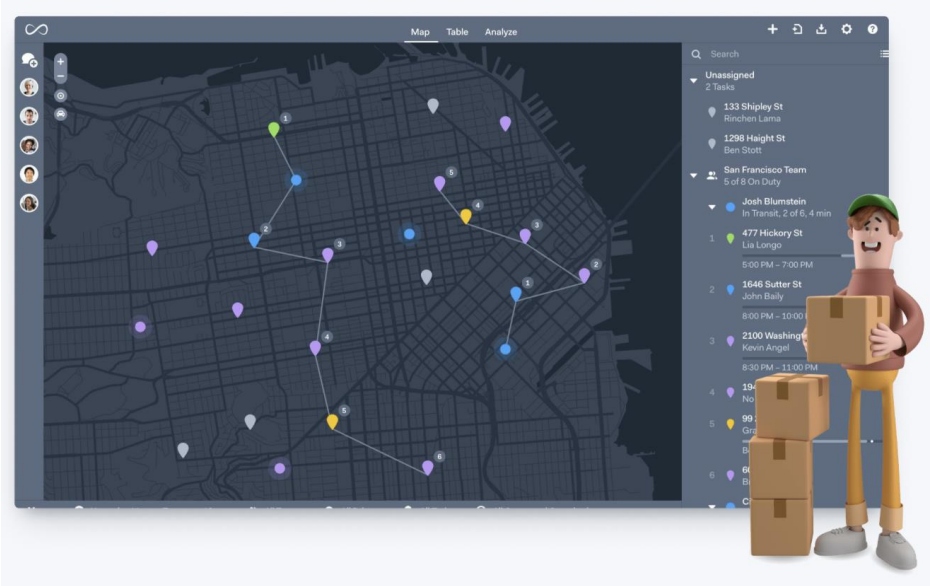

Рисунок 1.3 - платформа Onfleet

Джерело: [3].

Onfleet [4] - SaaS-платформа для управління доставкою "останньої милі" (last-mile delivery). Основні функції:

- автоматичне призначення завдань водіям;
- побудова маршрутів через Google Maps Directions API;
- відстеження водіїв в режимі реального часу;

- чат між диспетчерами та водіями.

Недоліки:

- використання Google Maps, яка не враховує габаритні та вагові обмеження вантажного транспорту;
- відсутність алгоритмів оптимізації при виборі водія (призначення відбувається за простими правилами: найближчий вільний водій);
- висока вартість ($149/місяць за 2000 завдань).

Переваги:

- функціонал зручний для перевезення невеликих кур'єрських відправлень;
- є окреме представлення для диспетчера та водія;
- клієнт відслідковує кур'єра на мапі;

На основі цього аналізу аналогів було складено таблицю загальних характеристик аналогів:

Таблиця 1.1 – Характеристики аналогічних систем

Характеристика	**TomTom Go Pro**	**Sygic Truck**	**Onfleet**	**Delivra**
Вартість	155$ рік	209.99$ + оплата за апдейти	599$\1299$\ 2999$ на м. залежно від тарифу	3999\8999\індивіду-ально грн на місяць
Навігація	Так, власна	Так, власна	Google Maps	Here Truck API
Оновлення карт	Раз на місяць	Раз на квартал	Реальний час	Реальний час
Управління завданнями	Ні	Ні	Так	Так
Автоматичне призначення водія	Ні	Ні	Так (Просте)	Так (Алогоритм зважених критеріїв)
Час призначення водія	-	-	2-3 хвилини	2-3 секунди
Точність габаритів	100%	70-80%	80-90%	100%
Вбудована комунікація	Ні	Ні	Так	Так

Розглядаючи вище зазначених конкурентів можна зробити висновок, що проєкт «Delivra» має як сильні так і слабкі сторони. По-перше проєкт «Delivra» одна з не багатьох систем, які втілюють не тільки навігацію, але й усі диспетчерські процеси. Наприклад: аналізувати місцеположення водіїв на карті в режимі реального часу, створювати логістичні завдання для водіїв, обирати найвигіднішого водія для конкретного завдання, підтримувати спілкування не відходячи від платформи. Це робить логістичну систему не тільки зручною для користувача, але й корисною та вигідною для бізнеса.

За результатами аналізу існуючих рішень, визначено ключові переваги логістичної системи «Delivra»:

- інтеграція з спеціальними API для навігації вантажного транспорту (HERE Maps Truck Routing API [4] замість загальних навігаційних систем дозволяє враховувати габаритні та вагові обмеження транспортних засобів);
- алгоритм рекомендації водіїв на основі методу зваженої суми критеріїв (автоматичне призначення завдань з урахуванням відстані до точки завантаження, поточної завантаженості водія та його досвіду);
- єдина платформа для комунікації (вбудовані засоби взаємодії мід диспетчерами та водіями без необхідності використання сторонніх месенджерів, що забезпечує безпеку корпоративних даних);
- веб-орієнтована архітектура (доступ до системи через браузер без необхідності встановлення спеціального обладнання на відміну від деяких аналогів);

1.3 Постановка задачі

Для відображення функцій які потребують автоматизації було розроблено концептуальну модель логістичної системи з використанням методології IDEF0. На рис. 1.4 представлено модель системи управління вантажними перевезеннями. Діаграма відображає систему як єдиний процес з визначенням потоків даних: входів (замовлення на доставку), виходів

(виконані доставки, звітність), управління (нормативні документи, SLA) та механізмів виконання (персонал, транспорт, програмне забезпечення.

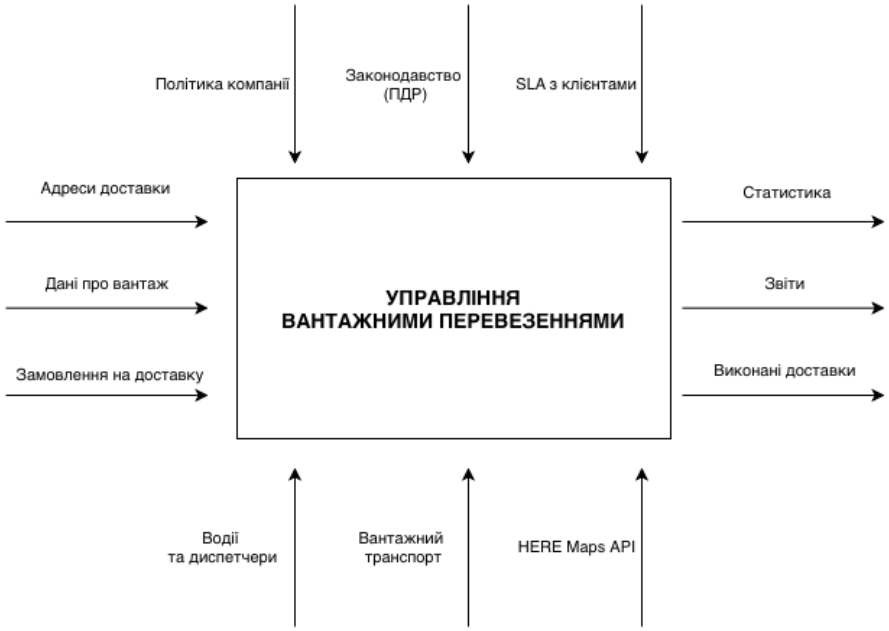

Рисунок 1.4 – Діаграма IDEF0

Джерело: розроблено автором

Модель відображає систему як єдиний процес трансформації вхідних даних (замовлення клієнтів) у результат (виконані доставки зі звітністю). Ключовою особливістю є чітке розділення потоків управління та потоків даних: управлінські параметри (політика компанії, законодавчі обмеження, SLA) визначають умови виконання процесу, але не трансформуються функцією, на відміну від входів, які перетворюються у виходи.

Вхідні дані: замовлення на доставку (адреси, дедлайн), параметри вантажу (вага, габарити, тип), список водіїв (локація, завантаженість, досвід), параметри вантажівок (висота, ширина, вага).

Вихідні дані: рекомендований водій (топ-3 варіанти за Score), оптимальний маршрут з урахуванням габаритних обмежень, час та відстань доставки, звіти по KPI (своєчасність, витрати палива).

Функціональні вимоги: система повинна автоматично призначати водіїв на основі WSM алгоритму, будувати маршрути з урахуванням габаритів через HERE Maps API, відстежувати GPS координати в реальному часі, забезпечувати комунікацію диспетчер-водій, генерувати звіти та аналітику.

Нефункціональні вимоги: час призначення водія < 3 секунди, час побудови маршруту < 2.5 секунди, uptime системи > 99%, підтримка до 500 одночасних користувачів, JWT автентифікація, інтуїтивний інтерфейс для всіх ролей.

Автоматизовані функції:

- призначення водіїв;
- побудова маршрутів;
- відстеження виконання;
- звітність;

Для досягнення мети дослідження необхідно виконати наступні етапи:

- збір та аналіз інформації (вивчення існуючих навігаційних систем та логістичних платформ з метою виділення переваг та недоліків, дослідження алгоритмів рекомендаційних систем, аналіз методів розв'язання задачі комівояжера, вивчення документації API для побудови маршрутів);

- проектування системи (розробка концептуальної моделі системи з визначенням основних функціональних модулів, побудова діаграм варіантів використання (Use Case Diagram) для ролей: адміністратор, диспетчер, водій, проєктування архітектури бази даних, розробка діаграми класів (Class Diagram) для серверної частини, побудова діаграм послідовності (Sequence Diagram) для ключових сценаріїв використання, вибір технологічного стеку, проєктування REST API для взаємодії клієнтської та серверної частин);

- реалізація системи (розробка серверної частини, розробка клієнтської частини);

- тестування системи (функціональне тестування REST API (Postman, JUnit), тестування алгоритмів рекомендації та оптимізації на тестових наборах даних, порівняльний аналіз ефективності розроблених

алгоритмів, інтеграційне тестування взаємодії frontend та backend, користувацьке тестування інтерфейсів (usability testing)).

Висновок до розділу 1

У першому розділі проведено аналіз предметної області управління вантажними перевезеннями та виявлено ключові проблеми існуючих рішень.

Встановлено що основними проблемами є відсутність урахування габаритних обмежень транспортних засобів при побудові маршрутів та неефективна комунікація між диспетчерами і водіями через звичайні месенджери. Вантажівки часто потрапляють під низькі мости або на дороги з обмеженням ваги, що призводить до затримок та додаткових витрат. Відправка адрес через месенджери створює ризики витоку конфіденційних даних клієнтів та помилок при передачі інформації.

Для структурування предметної області застосовано підхід Domain-Driven Design з виділенням ключових сутностей DeliveryTask, User, Company, NavigationSession та доменних сервісів для реалізації бізнес-логіки. Використання єдиної мови з експертами предметної області дозволило чітко відокремити бізнес-логіку від технічних деталей.

Проведено порівняльний аналіз існуючих рішень. Спеціалізовані навігатори TomTom GO Professional та Sygic Truck враховують габарити вантажівок, але мають високу вартість та рідкісні оновлення карт. Платформа Onfleet надає функціонал управління доставкою, але використовує Google Maps без урахування габаритних обмежень. Складено таблицю порівняння що виявила відсутність комплексних рішень які поєднують спеціалізовану навігацію з автоматизацією диспетчерських процесів.

Визначено ключові переваги системи «Delivra»: інтеграція з HERE Maps Truck Routing API для врахування габаритних обмежень, алгоритм WSM для автоматичного призначення водіїв що скорочує час з кількох хвилин до кількох секунд, вбудована комунікація без сторонніх месенджерів.

Розроблено концептуальну модель системи за методологією IDEF0 з визначенням вхідних даних, вихідних даних та потоків управління. Сформульовано функціональні вимоги: автоматичне призначення водіїв, побудова маршрутів з урахуванням габаритів, відстеження в реальному часі, комунікація та звітність.

Визначено нефункціональні вимоги: час призначення водія менше трьох секунд, час побудови маршруту менше двох з половиною секунд, uptime більше дев'яносто дев'яти відсотків. Визначено основні етапи виконання роботи: аналіз існуючих рішень та алгоритмів, проєктування системи з діаграмами варіантів використання та архітектури, реалізація серверної та клієнтської частин, функціональне та інтеграційне тестування.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ «DELIVRA»

2.1 Моделювання поведінки продукту

Для повноцінного формування діаграми варіантів використання спочатку потрібно зрозуміти конкретні функції системи, якими зможе користуватися той чи інший актор. Для цього потрібно базово описати користувацькі історії. За своєю суттю користувацька історія є неформальним способом побудувати спілкування між користувачами або представниками бізнесу та розробниками. Користувацька історія не є обов'язковим елементом документації, але майже завжди використовується у ситуаціях, коли потрібно зрозуміти та показати взаємодію користувача з програмним продуктом. Тож побудуємо користувацькі історії для кращого розуміння різних випадків використання майбутньої системи.

Перед написанням історій користувача варто чітко зазначити які ролі існують у системі. Логіст є користувачем, який планує маршрути, створює завдання для водіїв та контролює їх виконання через веб-інтерфейс. Водій є користувачем, який виконує надані логістом завдання доставки, оновлює статус завдання на різних етапах виконання та передає координати свого місцезнаходження для відстеження. Адміністратор є користувачем, який керує обліковими записами користувачів, призначає ролі та налаштовує права доступу до функцій системи. Таким чином можна проаналізувати ключові дії, які виконує кожна роль. Переходимо до створення історій користувача (Таблиця 2.1).

Таблиця 2.1 – Користувацькі історії користувача системи Delivra

№	User Story	Критерії прийнятності	Пріоритет
1	Як логіст я хочу створити новий маршрут із точками відправлення та призначення, щоб організувати доставку для певного водія	- Поля «Початкова точка», «Кінцева точка», «Дата», «Водій» є обов'язковими - Після створення маршрут зберігається в базі даних і відображається у списку маршрутів - Водій бачить новий маршрут у своєму профілі та отримує повідомлення	Високий
2	Як логіст я хочу автоматично оптимізувати маршрут доставки, щоб зменшити відстань та витрати на пальне	- Система застосовує алгоритм Дейкстри або інший алгоритм пошуку найкоротшого шляху - Оптимізований маршрут відображається на карті - Користувач може порівняти звичайний та оптимізований маршрути	Високий
3	Як логіст я хочу створити завдання для водія, щоб передати йому інформацію про доставку (адресу, час, товар)	- Завдання має опис, маршрут, дату та відповідального водія - Статус завдання за замовчуванням - «Очікує виконання» - Водій отримує повідомлення про нове завдання	Високий
4	Як логіст, я хочу бачити поточне місцезнаходження водія на карті, щоб контролювати виконання доставки в реальному часі	- Координати оновлюються кожні 10 секунд - На карті відображається маршрут і позиція водія - Якщо водій зупинився більш ніж на 5 хвилин, система фіксує це у звіті	Високий
5	Як користувач системи, я хочу авторизуватися за допомогою логіна та пароля, щоб отримати доступ до власного функціоналу	- Користувач вводить email і пароль - Після успішного входу система визначає його роль (логіст, водій, адміністратор) - Помилки відображаються при неправильних даних	Високий
6	Як водій, я хочу змінювати статус завдання ("у процесі", "виконано"), щоб логіст міг бачити актуальний стан доставки	- Водій може змінювати статус лише своїх завдань - Зміна статусу відображається миттєво у логіста - Після завершення завдання воно переходить у архів	Середній
7	Як адміністратор, я хочу переглядати, додавати, редагувати та видаляти користувачів, щоб підтримувати актуальність списку співробітників	- Можна призначати ролі користувачам - Видалення користувача підтверджується діалоговим вікном - Список користувачів оновлюється без перезавантаження сторінки	Середній
8	Як логіст, я хочу сформувати звіт за обраний період, щоб проаналізувати ефективність виконаних доставок	- Звіт містить кількість виконаних завдань, відстань, час доставки - Формат експорту - PDF або CSV - Звіт зберігається у профілі користувача	Низький
9	Як водій, я хочу переглядати історію своїх маршрутів, щоб бачити, які доставки я виконав і скільки часу це зайняло	- Історія містить дату, маршрут, відстань і статус - Водій бачить лише власні маршрути - Дані зберігаються у базі протягом 90 днів	Низький

На основі існуючих User Story побудуємо Use Case 3.0 діаграму зі слайсами (Рис. 2.1).

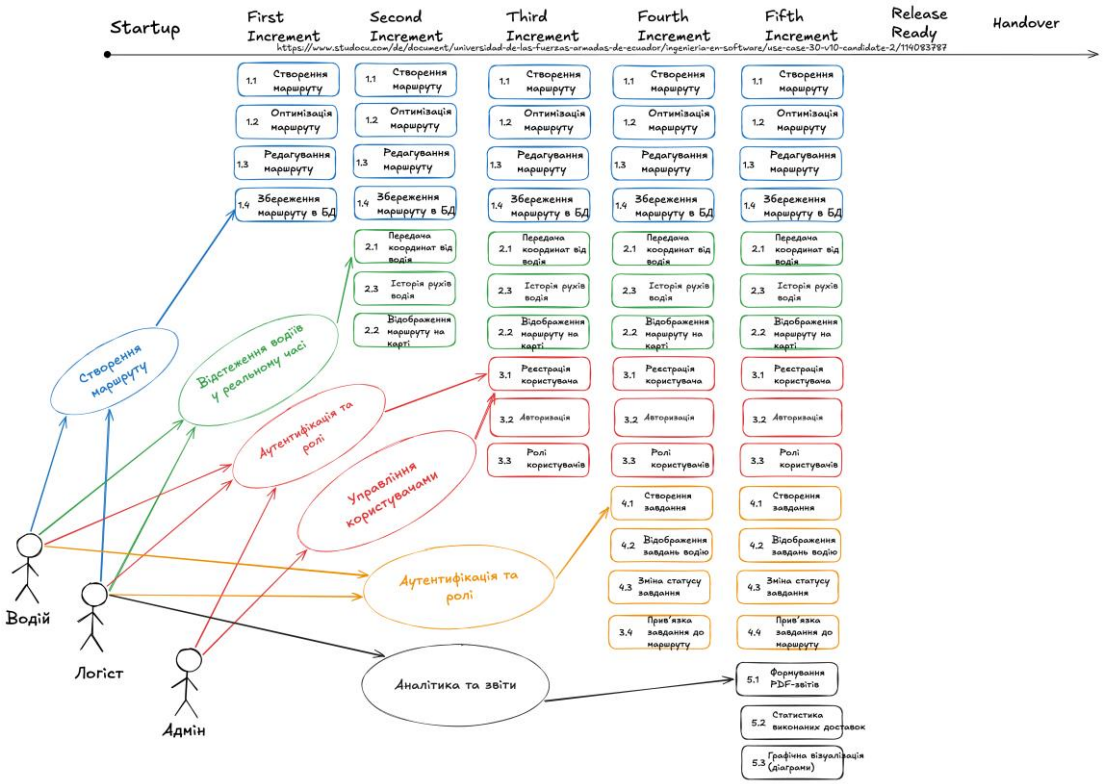

Рисунок 2.1 – Діаграма Use Case 3.0

Джерело: розроблено автором

Use Case 3.0 [10] є сучасною еволюцією класичних діаграм варіантів використання, що інтегрує принципи гнучкої розробки програмного забезпечення. Основна відмінність від традиційних Use Case діаграм полягає у додаванні часової осі з інкрементами розробки та організації функцій за пріоритетом від найважливіших до другорядних.

Класичні діаграми варіантів використання показують статичний набір функцій системи без визначення послідовності їх впровадження. Use Case 3.0 через методологію User Story Mapping візуалізує еволюцію продукту, пріоритизацію функцій та послідовність їх реалізації через інкременти. Це дозволяє чітко визначити мінімально життєздатний продукт та спланувати послідовне нарощування функціональності.

Методологія User Story Mapping організує користувацькі історії у двовимірну структуру. Горизонтальна вісь представляє шлях користувача або послідовність дій для досягнення мети. Вертикальна вісь відображає пріоритет функцій, де найважливіші розміщені вгорі. Такий підхід дозволяє візуально побачити як буде розвиватися продукт від простого до складного, від критично необхідного до бажаного.

Ключовою перевагою Use Case 3.0 є можливість визначення меж релізів. Горизонтальна лінія через карту відокремлює функції, що увійдуть в поточний реліз, від тих що залишаться на майбутнє. Це забезпечує прозорість планування та допомагає команді зосередитися на найважливішому.

Діаграма організована у двох вимірах. Горизонтальна вісь відображає етапи розробки від початкового налаштування до передачі замовнику. Вертикальна вісь групує функціональність за ролями користувачів.

Горизонтальна вісь починається з етапу Startup для початкового налаштування проєкту.

Перший інкремент реалізує базову функціональність створення, оптимізації та редагування маршрутів для мінімально життєздатного продукту.

Другий інкремент додає передачу координат в реальному часі, історію руху та відображення на інтерактивній карті.

Третій інкремент впроваджує реєстрацію, автентифікацію та розмежування трьох ролей: водій, логіст та адміністратор.

Четвертий інкремент інтегрує навігацію з управлінням завданнями через створення завдань доставки, відображення на панелі адміністратора, зміну статусів та автоматичне призначення до маршрутів.

П'ятий інкремент додає формування звітів, статистику доставок та графічну візуалізацію ключових показників.

Етап Release Ready включає оптимізацію продуктивності, тестування навантаження та підготовку документації. Handover завершує передачу системи замовнику.

Вертикальна вісь відображає розподіл одинадцяти функцій водія для навігації та виконання доставок, восьми функцій логіста для планування маршрутів та дев'яти функцій адміністратора для управління завданнями та аналітики. Кольорові овали групують функції за областями: синій для створення маршрутів, зелений для відстеження координат, червоний для автентифікації, помаранчевий для управління завданнями та коричневий для звітності.

Застосування карти користувацьких історій забезпечує інкрементальну розробку з отриманням робочого продукту після кожного інкременту. Це дозволяє збирати зворотний зв'язок від користувачів та швидко реагувати на зміни вимог, зменшуючи ризики розробки невідповідного функціоналу.

Чітка пріоритизація функціональності забезпечує розміщення найважливіших функцій у перших інкрементах. Створення маршруту та навігація, як критично важливі для основної діяльності водіїв, реалізовані вже в першому інкременті.

Візуалізація шляху користувача через карту історій показує не лише що система вміє робити, але й як користувач взаємодіє з нею на різних етапах. Така візуалізація полегшує комунікацію між розробниками, замовником та кінцевими користувачами.

Узгодження з гнучкими методологіями досягається через відповідність кожного інкременту ітерації розробки. Така структура полегшує планування та розподіл задач між членами команди. Кожна ітерація завершується демонстрацією працюючого функціоналу.

Діаграми діяльності використовуються для моделювання послідовності дій та логіки виконання бізнес-процесів системи. На відміну від діаграм варіантів використання, що описують функціональність системи, діаграми діяльності деталізують алгоритми виконання конкретних процесів, включаючи умовні переходи, паралельні дії та цикли.

На рис. 2.2 зображено діаграму діяльності.

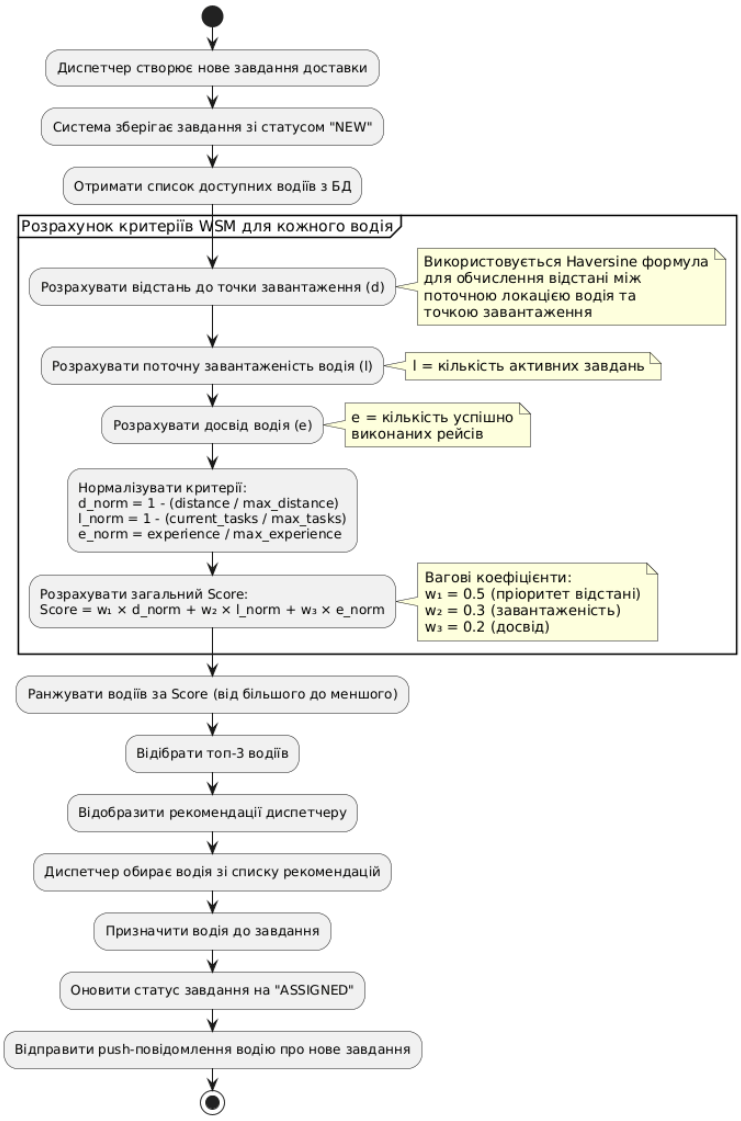

Рисунок 2.2 – Діаграма діяльності (Призначення водія)

Джерело: розроблено автором

Процес розпочинається зі створення диспетчером нового завдання з вказанням адрес, параметрів вантажу та терміну виконання. Система зберігає завдання зі статусом "нове" та ініціює пошук оптимального водія.

Центральною частиною процесу є розрахунок трьох критеріїв для кожного доступного водія. Перший критерій, відстань до точки завантаження, обчислюється для визначення найближчого водія. Другий критерій, завантаженість, визначається як кількість активних завдань для уникнення перевантаження. Третій критерій, досвід, розраховується на основі кількості успішно виконаних рейсів.

Всі критерії нормалізуються до шкали від нуля до одиниці. Загальний бал обчислюється як зважена сума з коефіцієнтами п'ять десятих для відстані, три десятих для завантаженості та два десятих для досвіду.

Система ранжує водіїв за спаданням балу та відбирає трьох найкращих кандидатів. Диспетчер обирає водія зі списку рекомендацій, враховуючи автоматичні рекомендації та додаткові фактори. Після призначення статус завдання змінюється на "призначено" та водій отримує повідомлення про нове завдання.

На рис. 2.3 представлено діаграму діяльності процесу побудови маршруту з урахуванням габаритних обмежень вантажівки.

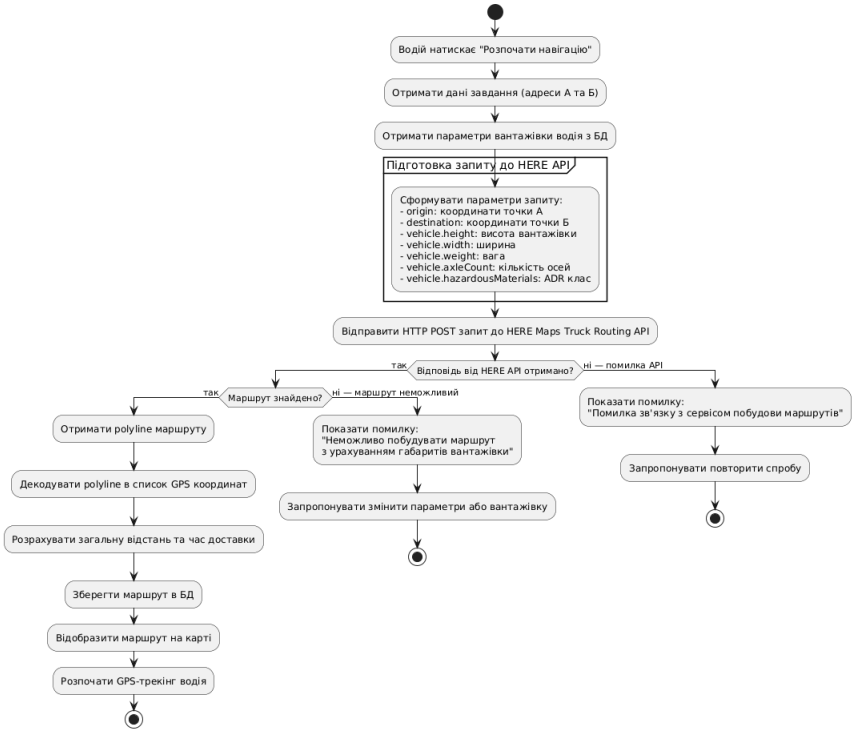

Рисунок 2.3 – Діаграма діяльності (Побудова маршруту)

Джерело: розроблено автором

Процес ініціюється водієм через натискання кнопки "Розпочати навігацію". Система отримує дані завдання, включаючи координати точок завантаження та розвантаження, а також параметри вантажівки водія. Система формує запит з параметрами маршрутизації. Висота вантажівки

використовується для уникнення низьких мостів та тунелів. Ширина враховується при виборі вузьких доріг. Вага впливає на вибір мостів з обмеженням навантаження. Кількість осей важлива для деяких регіональних обмежень. Клас небезпечних вантажів визначає можливість руху через населені пункти та тунелі. Система відправляє запит до зовнішнього сервісу маршрутизації. У разі успішного отримання відповіді перевіряється наявність можливого маршруту. Якщо маршрут знайдено, система отримує набір координат та розраховує загальну відстань і прогнозований час доставки. Маршрут зберігається для подальшого аналізу, відображається на інтерактивній карті та розпочинається відстеження фактичного руху водія.

На рис. 2.4 представлено діаграму діяльності, що описує повний цикл виконання доставки від створення завдання до його завершення з відображенням взаємодії між диспетчером, системою та водієм.

Діаграма використовує доріжки для відображення дій різних акторів. Доріжка диспетчера містить дії зі створення завдання та призначення водія. Доріжка системи включає автоматичні процеси зміни статусів та побудови маршрутів. Доріжка водія відображає дії з навігації та виконання доставки.

Процес розпочинається з створення диспетчером завдання доставки. Після призначення водія система змінює статус завдання на "призначено" та відправляє повідомлення водію. Водій переглядає деталі завдання та розпочинає навігацію, що ініціює побудову маршруту та зміну статусу на "в процесі".

Під час руху водія система постійно отримує координати з частотою один раз на десять секунд. Для кожної позиції перевіряється відхилення від побудованого маршруту. Якщо відхилення перевищує п'ятдесят метрів, система повідомляє водія та пропонує перерозрахувати маршрут. У разі підтвердження будується новий маршрут від поточної локації до точки призначення.

Після прибуття до точки розвантаження водій змінює статус завдання на "доставлено". Система зберігає час завершення, розраховує фактичні витрати

палива, оновлює статистику водія та генерує звіт по доставці. Диспетчер отримує повідомлення про завершення та може переглянути згенерований звіт.

Діаграми діяльності забезпечують детальне розуміння логіки виконання ключових процесів системи та слугують основою для реалізації.

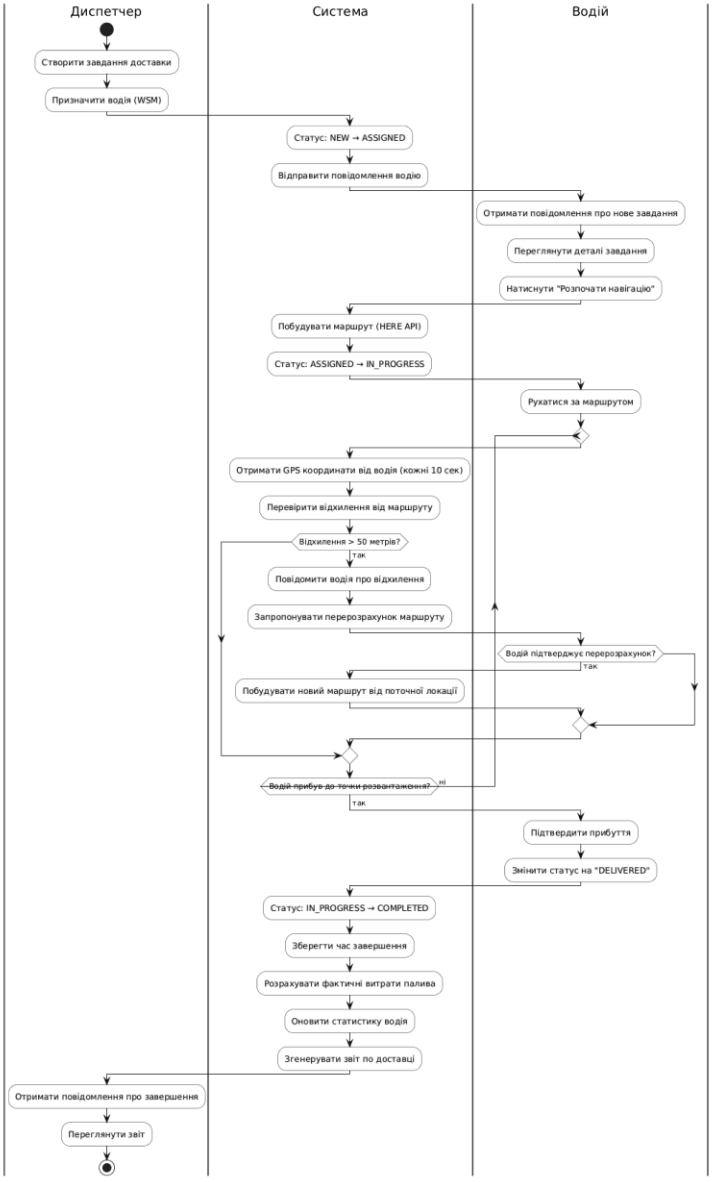

Рисунок 2.4 – Діаграма діяльності (Повний цикл доставки)

Джерело: розроблено автором

На рис. 2.5 представлено діаграму послідовності процесу створення завдання доставки диспетчером та автоматичного призначення оптимального водія з використанням алгоритму зваженої суми критеріїв.

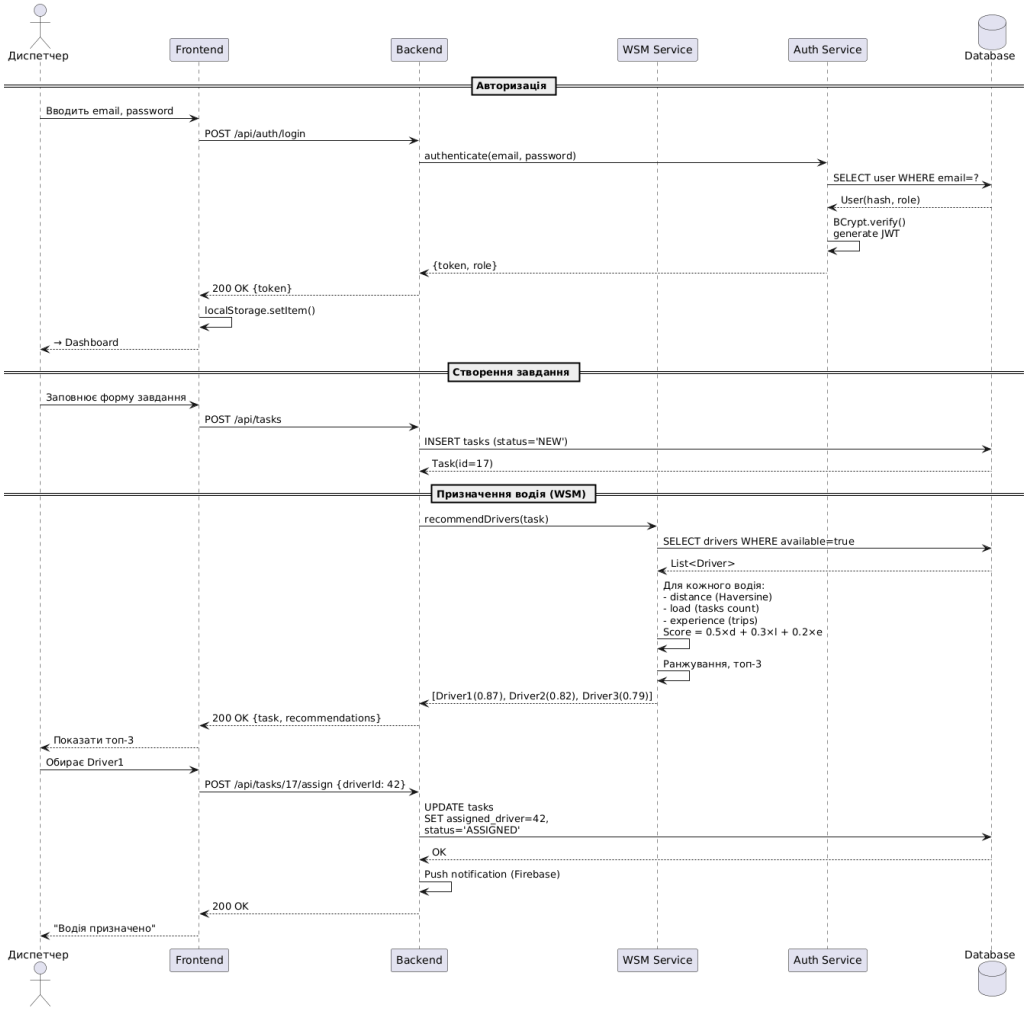

Рисунок 2.5 – Діаграма послідовності (Створення завдання та призначення водія)

Джерело: розроблено автором

Діаграма складається з трьох логічних блоків: авторизації диспетчера, створення завдання та призначення водія.

Процес розпочинається з авторизації диспетчера в системі. Диспетчер вводить облікові дані у веб-інтерфейс. Система відправляє запит з обліковими даними та виконує верифікацію пароля порівнюючи введений пароль з збереженим. У випадку успішної верифікації генерується токен для подальшої автентифікації. Токен зберігається для використання в наступних запитах. Після успішної авторизації диспетчер перенаправляється на панель управління.

Диспетчер створює нове завдання доставки, заповнюючи форму з адресами, параметрами вантажу та терміном виконання. Система зберігає завдання зі статусом "нове" та отримує унікальний ідентифікатор.

Центральною частиною процесу є автоматичне призначення оптимального водія. Система запитує список всіх доступних водіїв та виконує розрахунок трьох критеріїв для кожного. Перший критерій, відстань до точки завантаження, обчислюється для визначення найближчого водія. Другий критерій, завантаженість, визначається як кількість активних завдань для уникнення перевантаження. Третій критерій, досвід, розраховується на основі кількості виконаних рейсів.

Після нормалізації всіх критеріїв система обчислює загальний бал для кожного водія з коефіцієнтами п'ять десятих для відстані, три десятих для завантаженості та два десятих для досвіду. Водії ранжуються за спаданням балу та відбираються три найкращі кандидати.

Диспетчер бачить топ три водіїв з їх балами. Після вибору водія система оновлює завдання, встановлюючи призначеного водія та змінюючи статус на "призначено". Водій отримує повідомлення про нове завдання.

На рис. 2.6 представлено діаграму послідовності процесу виконання доставки водієм, включаючи побудову маршруту з урахуванням габаритних обмежень та відстеження позиції в реальному часі.

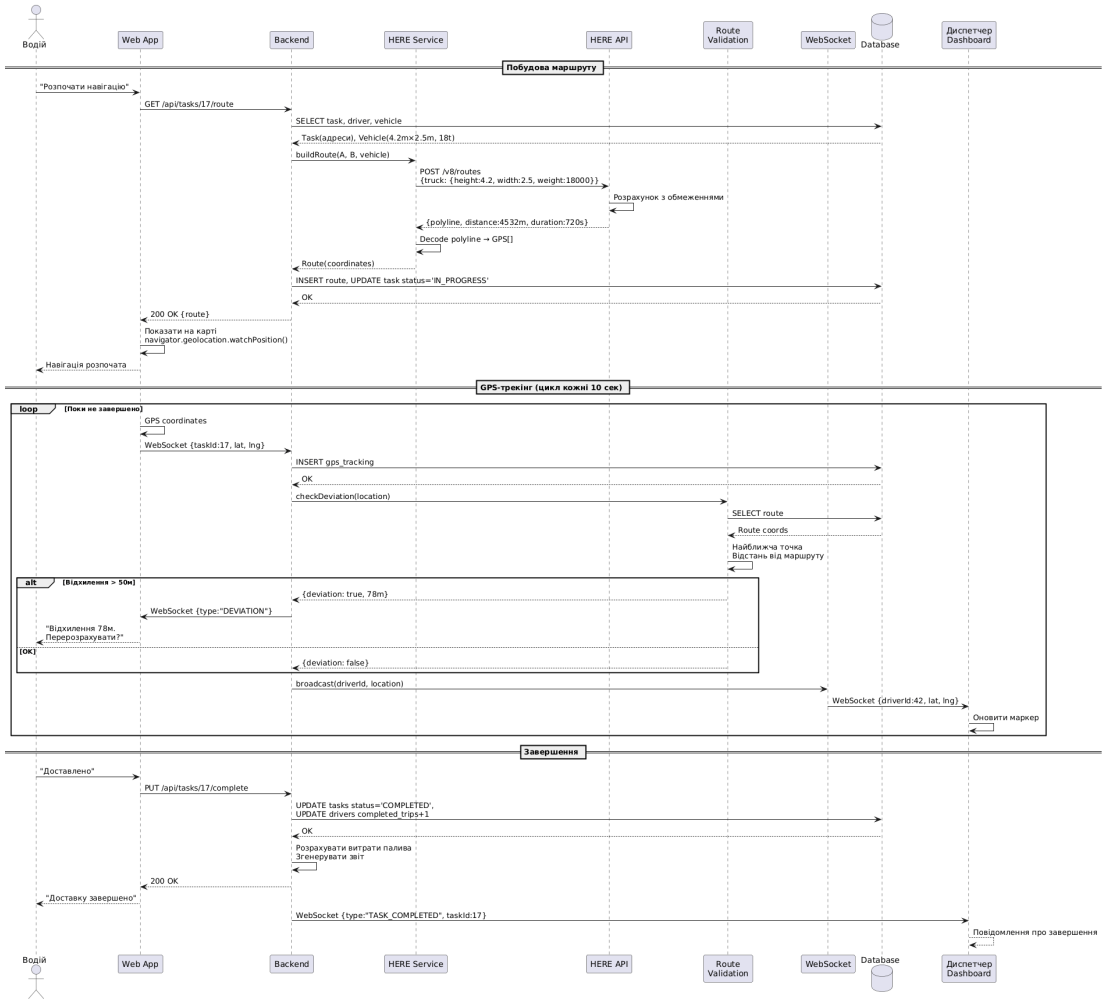

Рисунок 2.6 – Діаграма послідовності (Виконання доставки)

Джерело: розроблено автором

Діаграма складається з трьох логічних блоків: побудови маршруту, відстеження під час доставки та завершення доставки.

Процес розпочинається з натискання водієм кнопки "Розпочати навігацію". Система послідовно отримує дані завдання, інформацію про водія та параметри його вантажівки. Система формує запит до зовнішнього сервісу маршрутизації з адресами та параметрами вантажівки, включаючи висоту, ширину та вагу.

Зовнішній сервіс виконує розрахунок маршруту з урахуванням висоти мостів, вагових обмежень доріг та заборон для небезпечних вантажів. Сервіс повертає маршрут з загальною довжиною та прогнозованою тривалістю. Система зберігає маршрут, оновлює статус завдання на "в процесі" та

відображає маршрут на карті. Розпочинається відстеження фактичного руху водія.

Під час руху система кожні десять секунд отримує поточні координати водія та зберігає їх для подальшого аналізу. Система перевіряє чи водій рухається за побудованим маршрутом. Якщо відхилення перевищує п'ятдесят метрів, система повідомляє водія з пропозицією перерозрахувати маршрут. Незалежно від наявності відхилення система транслює оновлену позицію водія до диспетчерських панелей в реальному часі.

Після прибуття до точки розвантаження водій натискає кнопку "Доставлено". Система оновлює статус завдання на "виконано" та зберігає час завершення. На основі пройденої відстані розраховуються фактичні витрати палива. Оновлюється лічильник виконаних рейсів водія для використання при майбутніх призначеннях. Система генерує звіт з фактичним часом виконання, пройденою відстанню та витратами палива. Диспетчер отримує сповіщення про завершення та може переглянути згенерований звіт.

Діаграми послідовності забезпечують розуміння взаємодії компонентів системи протягом повного життєвого циклу завдання доставки.

2.2 Моделювання архітектури та структури продукту

Діаграма класів відображає доменну модель системи «Delivra», спроектовану з використанням принципів предметно-орієнтованого проєктування.

На рис. 2.7 представлено діаграму класів, згенеровану з фактичної структури коду.

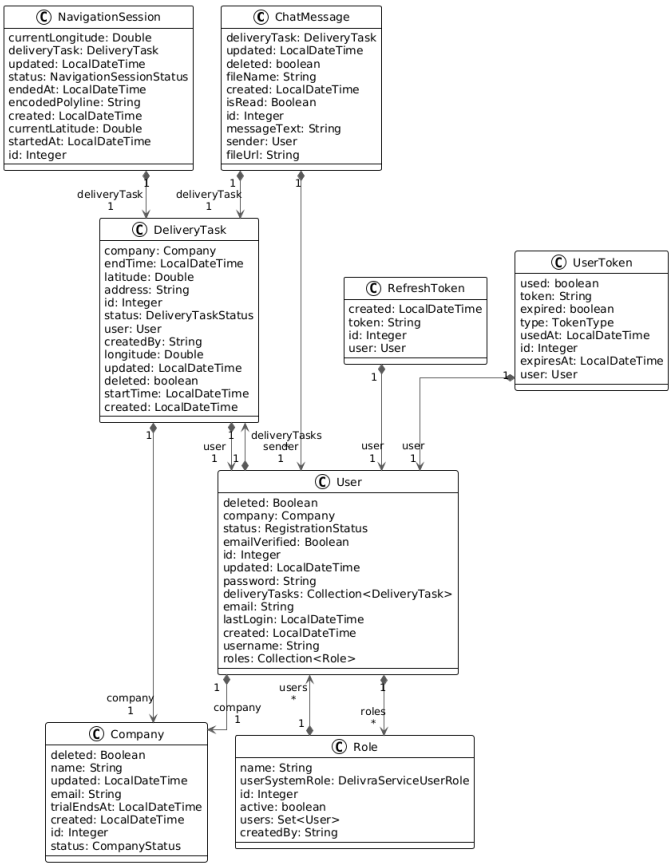

Рисунок 2.7 – Діаграма класів

Джерело: розроблено автором

DeliveryTask є центральною сутністю системи. Містить адресу доставки, координати, статус виконання, час початку та завершення. Пов'язаний з користувачем, компанією, навігаційною сесією та повідомленнями чату.

User представляє користувача системи з обліковими даними. Пароль зберігається у захешованому вигляді. Користувач має множину ролей та належить до компанії. Атрибути підтвердження електронної пошти та часу останнього входу забезпечують безпеку та аудит.

Company представляє організацію-клієнта системи. Містить назву, електронну пошту та статус. Атрибут терміну пробного періоду визначає дату закінчення безкоштовного використання. Зв'язок з користувачами забезпечує ізоляцію даних між компаніями.

Role визначає роль користувача в системі з попередньо визначеними правами доступу. Зв'язок багато до багатьох з користувачами дозволяє призначати множину ролей одному користувачу.

NavigationSession зберігає дані активної навігації для завдання. Містить поточні координати, закодований маршрут та статус сесії. Зв'язок з завданням забезпечує відстеження руху водія під час виконання доставки.

ChatMessage реалізує комунікацію між диспетчерами та водіями. Містить текст повідомлення, посилання на файл та статус прочитання. Зв'язок з завданням прив'язує повідомлення до контексту конкретної доставки.

RefreshToken та UserToken забезпечують автентифікацію. RefreshToken зберігає токен оновлення для тривалих сесій. UserToken використовується для підтвердження електронної пошти та скидання пароля.

Центральний клас DeliveryTask пов'язаний з користувачем як виконавець завдання, з компанією для ізоляції даних, з навігаційною сесією для відстеження та з повідомленнями для комунікації. Користувач має множину ролей та належить до однієї компанії. Компанія містить множину користувачів.

Така структура забезпечує ізоляцію даних між організаціями та гнучке управління доступом.

Діаграма компонентів відображає високорівневу архітектуру системи «Delivra», організовану за принципом багатошарової архітектури з чітким розмежуванням відповідальності між компонентами.

На рис. 2.8 представлено діаграму компонентів системи.

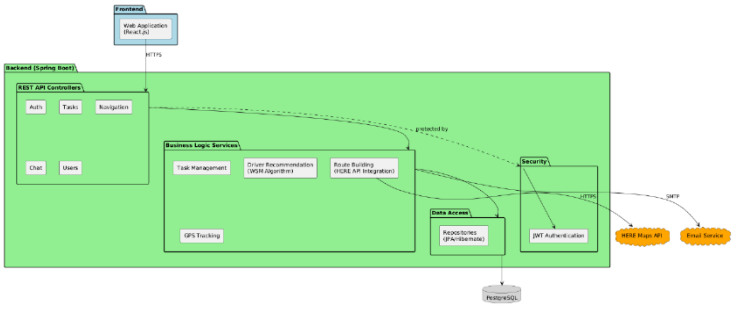

Рисунок 2.8 – Діаграма компонентів

Джерело: розроблено автором

Архітектура системи. Система складається з трьох основних частин: клієнтського додатку, серверної частини та бази даних.

Frontend представлений веб-додатком, що надає інтерфейс для диспетчерів та адміністраторів. Додаток взаємодіє з серверною частиною через захищене з'єднання, відправляючи запити та отримуючи відповіді. Також використовується двостороннє з'єднання для отримання оновлень позиції водіїв в реальному часі під час виконання доставок.

Backend організований у чотири шари. Перший шар обробляє вхідні запити через п'ять основних контролерів. Контролер автентифікації керує входом користувачів. Контролер завдань надає операції для управління завданнями доставки. Контролер навігації забезпечує відстеження водіїв. Контролер чату реалізує обмін повідомленнями між диспетчерами та водіями. Контролер користувачів надає операції управління користувачами та компаніями.

Другий шар містить основну бізнес-логіку системи. Сервіс управління завданнями керує життєвим циклом завдань від створення до завершення. Сервіс рекомендації водіїв реалізує алгоритм вибору оптимального водія на основі трьох факторів: відстані до точки завантаження, поточної завантаженості та досвіду виконання рейсів. Сервіс відстеження забезпечує моніторинг позиції водіїв в реальному часі з частотою оновлення кожні десять секунд та виконує детекцію відхилення від побудованого маршруту. Сервіс побудови маршрутів інтегрується з зовнішнім сервісом для побудови маршрутів з урахуванням габаритних обмежень вантажівок.

Третій шар абстрагує деталі роботи з базою даних та надає методи для операцій створення, читання, оновлення та видалення даних.

Четвертий шар забезпечує захист через перевірку токенів автентифікації. Компонент перехоплює всі вхідні запити та перевіряє наявність та валідність токену. При успішній валідації запит передається до контролерів, при невдалій повертається помилка доступу.

База даних зберігає всі дані системи, включаючи користувачів, завдання доставки, навігаційні сесії, повідомлення чату та історію координат.

Зовнішні сервіси включають сервіс маршрутизації для геокодування адрес та побудови маршрутів з урахуванням габаритних обмежень. Сервіс електронної пошти відправляє листи для підтвердження реєстрації, скидання пароля та повідомлень про зміну статусу завдань.

Взаємодія між компонентами відбувається через чітко визначені інтерфейси. Така архітектура забезпечує низьку зв'язаність компонентів, що полегшує тестування, підтримку та розширення системи.

Модель C4 [11] є методологією візуалізації архітектури програмних систем через чотири рівні деталізації: Context (контекст), Containers (контейнери), Components (компоненти) та Code (код). Кожен рівень надає різний ступінь деталізації від загального контексту системи до структури коду окремих компонентів.

Перший рівень Context показує систему як єдине ціле в контексті зовнішніх користувачів та систем.

Другий рівень Containers деталізує внутрішню структуру на окремі контейнери, такі як веб-додаток, серверна частина та база даних.

Третій рівень Components розкриває внутрішню будову окремих контейнерів до рівня програмних компонентів.

Четвертий рівень Code показує деталі реалізації окремих компонентів, але зазвичай не включається в документацію, оскільки може бути згенерований автоматично з коду.

Для системи «Delivra» розроблено чотири (один з рівнів вже був наведений вище) рівні моделі C4, що забезпечують повне розуміння архітектури від взаємодії з зовнішніми системами до внутрішньої структури компонентів.

Контекстна діаграма відображає систему як єдине ціле в контексті зовнішніх акторів та систем. Цей рівень показує хто використовує систему та з якими зовнішніми сервісами вона інтегрується без деталізації внутрішньої структури.

На рис. 2.9 представлено діаграму контексту системи «Delivra».

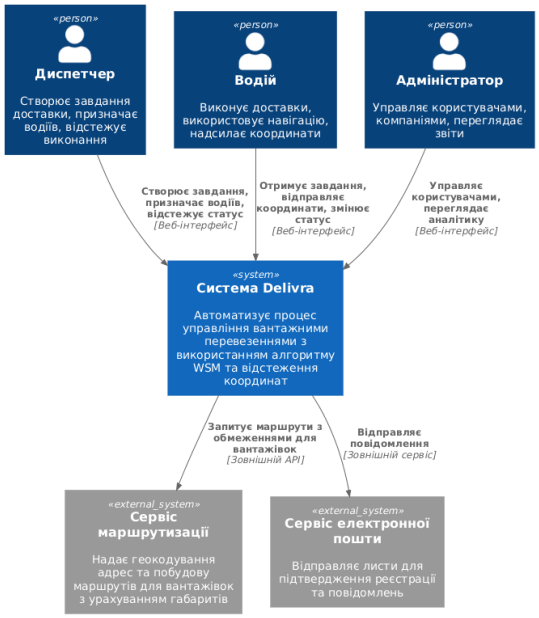

Рисунок 2.9 – Контекстна діаграма

Джерело: розроблено автором

Система має трьох основних типів користувачів, кожен з яких виконує специфічні функції в процесі управління доставками.

Диспетчер створює завдання доставки з вказанням адрес, параметрів вантажу та терміну виконання. Призначає водіїв до завдань на основі автоматичних рекомендацій системи та відстежує статус виконання в реальному часі на інтерактивній карті. Взаємодіє з системою через веб-інтерфейс з використанням захищеного з'єднання.

Водій отримує призначені завдання через мобільний додаток, використовує вбудовану навігацію з урахуванням габаритів вантажівки та відправляє координати кожні десять секунд для відстеження позиції. Змінює статус завдання при досягненні контрольних точок та обмінюється повідомленнями з диспетчером через вбудований чат.

Адміністратор управляє користувачами системи, створюючи облікові записи водіїв та диспетчерів, призначаючи їм ролі та права доступу. Управляє компаніями клієнтами, переглядає аналітику виконання доставок та формує

звіти за різними періодами. Доступ здійснюється через веб-інтерфейс з підвищеними привілеями.

Система «Delivra» інтегрується з двома зовнішніми сервісами для забезпечення повного функціоналу.

Сервіс маршрутизації надає функціонал геокодування для перетворення текстових адрес в координати та побудови маршрутів з урахуванням габаритних обмежень вантажівок. Сервіс враховує висоту мостів, вагові обмеження доріг, ширину проїжджої частини та заборони руху для небезпечних вантажів. Запити виконуються через зовнішній інтерфейс з автентифікацією.

Сервіс електронної пошти використовується для відправки листів користувачам. Система відправляє листи підтвердження при реєстрації, листи з посиланням для скидання пароля, повідомлення про призначення нових завдань водіям та звіти про виконані доставки.

Контекстна діаграма забезпечує високорівневе розуміння місця системи «Delivra» в екосистемі логістичної компанії та її взаємодії з зовнішніми сервісами для надання повного функціоналу управління вантажними перевезеннями.

Діаграма контейнерів деталізує внутрішню структуру системи «Delivra», показуючи основні контейнери та їх взаємодію. Контейнер в термінології C4 означає окремий процес або середовище виконання, таке як веб-додаток, серверна частина або база даних.

На рис. 2.10 представлено діаграму контейнерів системи «Delivra».

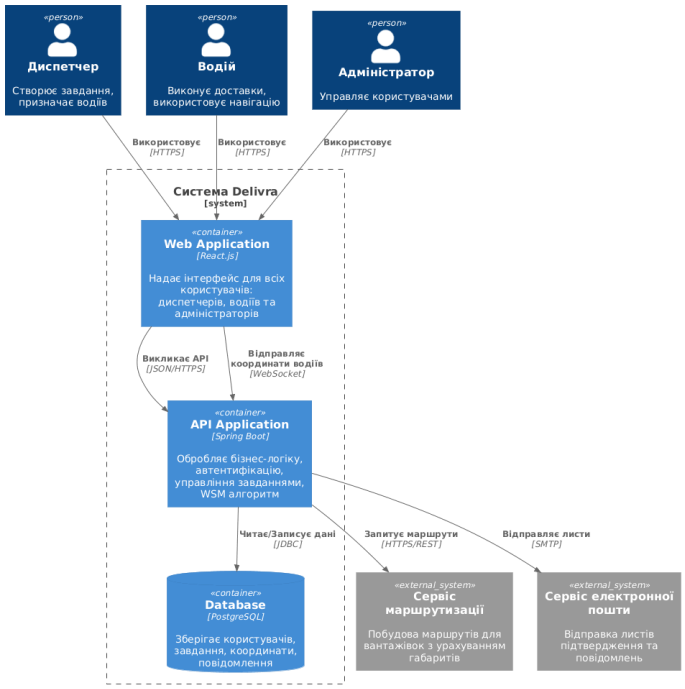

Рисунок 2.10 – Діаграма контейнерів

Джерело: розроблено автором

Веб-додаток є єдиною точкою доступу для всіх типів користувачів системи та надає диференційований інтерфейс залежно від ролі.

Диспетчери використовують додаток для створення завдань доставки з вказанням адрес, параметрів вантажу та терміну виконання. Отримують автоматичні рекомендації щодо призначення оптимального водія на основі алгоритму WSM. Відстежують виконання доставок на інтерактивній карті в реальному часі з можливістю перегляду історії руху та обміну повідомленнями з водіями.

Водії через веб-інтерфейс переглядають призначені завдання з деталями доставки. Розпочинають навігацію з автоматичною побудовою маршруту з урахуванням габаритів вантажівки. Система автоматично відправляє координати водія кожні десять секунд для відстеження позиції диспетчером. Водій змінює статус завдання при досягненні контрольних точок та може обмінюватися повідомленнями з диспетчером.

Адміністратори отримують доступ до управління користувачами з можливістю створення облікових записів, призначення ролей та прав доступу.

Управляють компаніями клієнтами для забезпечення багатокомпанійності системи. Переглядають аналітику виконання доставок та формують звіти за різними періодами.

Веб-додаток взаємодіє з серверною частиною через захищене з'єднання, відправляючи запити та отримуючи відповіді. Для відстеження координат водіїв використовується двосторонній канал обміну даними в реальному часі.

Серверна частина є центральним компонентом системи, що обробляє всю бізнес-логіку та координує взаємодію між веб-додатком, базою даних та зовнішніми сервісами.

Компонент обробляє автентифікацію користувачів з перевіркою облікових даних та генерацією токенів доступу. Забезпечує розмежування прав доступу на основі ролей користувачів. Управляє повним життєвим циклом завдань доставки від створення до завершення з валідацією даних та зміною статусів.

Реалізує алгоритм WSM для автоматичної рекомендації оптимального водія. Для кожного доступного водія розраховує три критерії: відстань до точки завантаження, поточну завантаженість та досвід виконання рейсів. Нормалізує критерії та обчислює загальний бал для ранжування водіїв.

Забезпечує відстеження координат водіїв в реальному часі з частотою оновлення кожні десять секунд. Перевіряє відхилення від побудованого маршруту та повідомляє водія при відхиленні більше п'ятдесяти метрів з пропозицією перерозрахувати маршрут. Зберігає історію координат для подальшого аналізу ефективності виконання доставок.

Серверна частина взаємодіє з базою даних для зберігання та отримання всіх даних системи. Інтегрується з сервісом маршрутизації для побудови оптимальних маршрутів з урахуванням висоти, ширини, ваги вантажівки та класифікації небезпечних вантажів. Використовує сервіс електронної пошти для відправки листів підтвердження реєстрації, скидання пароля та повідомлень про призначення нових завдань.

База даних зберігає всі дані системи в структурованому вигляді з забезпеченням цілісності та швидкого доступу.

Зберігаються облікові записи користувачів з паролями у захешованому вигляді, ролями та належністю до компаній для багатокомпанійності. Завдання доставки містять адреси завантаження та розвантаження, параметри вантажу, статуси виконання та призначених водіїв. Навігаційні сесії зберігають поточні маршрути та закодовані координати для відображення на карті.

Історія координат водіїв зберігається з прив'язкою до завдань для аналізу фактичних маршрутів та розрахунку витрат палива. Повідомлення чату прив'язані до конкретних завдань для забезпечення контекстної комунікації між диспетчерами та водіями. Токени автентифікації забезпечують безпечний доступ користувачів з можливістю оновлення сесії.

Всі користувачі системи взаємодіють з єдиним веб-додатком, який надає різний функціонал залежно від ролі користувача. Веб-додаток ініціює запити до серверної частини через захищене з'єднання. Для водіїв додатково підтримується двосторонній канал для автоматичної передачі координат з частотою один раз на десять секунд.

Серверна частина обробляє запити, виконує бізнес-логіку та взаємодіє з базою даних для збереження та отримання даних. При необхідності серверна частина викликає зовнішні сервіси для побудови маршрутів або відправки повідомлень користувачам.

Третій рівень моделі C4 деталізує внутрішню структуру окремих контейнерів до рівня програмних компонентів та їх взаємодії. Для системи «Delivra» найбільш складним контейнером є серверна частина, яка містить множину взаємодіючих компонентів, організованих у чотири шари.

Детальна діаграма компонентів серверної частини системи представлена вище (рис. 2.8). Діаграма показує структуру та взаємодію компонентів чотирьох шарів архітектури: контролерів для обробки вхідних запитів, сервісів для реалізації бізнес-логіки, репозиторіїв для доступу до даних та компонентів безпеки для автентифікації користувачів.

2.3 Опис алгоритмів та математичних моделей

Система «Delivra» використовує набір математичних моделей та алгоритмів для забезпечення оптимального функціонування процесів управління вантажними перевезеннями. Алгоритми поділяються на три категорії: академічні для прийняття рішень, класичні для геометричних обчислень та прикладні для навігації в реальному часі.

Для однієї з найважливіших задач логістичної платформи «Delivra» - автоматичного призначення водія до завдання можна використовувати декілька алгоритмів, найкращими з них є алгоритми k найближчих сусідів та алгоритм суми зважених критеріїв (WSM).

Алгоритм k найближчих сусідів є класичним методом машинного навчання, що також може використовуватися для вибору водія на основі історичних даних про успішні призначення. Однак для системи «Delivra» алгоритм WSM є більш придатним з наступних причин.

По-перше, алгоритм k найближчих сусідів вимагає значного обсягу історичних даних для навчання моделі. Для нових компаній або компаній з невеликою кількістю водіїв недостатньо даних для побудови надійної моделі. WSM працює з поточними даними без потреби в історії.

По-друге, алгоритм k найближчих сусідів не дозволяє явно контролювати пріоритети критеріїв. Алгоритм автоматично визначає важливість факторів на основі навчальних даних, що може не відповідати стратегії конкретної компанії. WSM дає можливість налаштувати вагові коефіцієнти відповідно до бізнес-вимог.

По-третє, алгоритм k найближчих сусідів має складність обчислень O(n × m), де n - кількість історичних записів, m - кількість ознак. Для великої історії це призводить до повільних обчислень при кожному запиті. WSM має лінійну складність O(k), де k - кількість доступних водіїв, що забезпечує швидку роботу навіть для великих парків.

По-четверте, результати алгоритм k найближчих сусідів важко пояснити диспетчеру. Алгоритм надає рекомендацію на основі схожості з історичними випадками, але не пояснює чому саме цей водій обраний. WSM надає прозорі бали з розбивкою по критеріям, що дозволяє диспетчеру зрозуміти логіку рекомендації та прийняти обґрунтоване рішення. Таким чином, алгоритм WSM краще відповідає вимогам системи завдяки простоті реалізації, швидкості роботи, можливості налаштування та прозорості результатів.

Метод зваженої суми критеріїв є академічним алгоритмом з галузі дослідження операцій та теорії прийняття рішень. Використовується для автоматичного вибору оптимального водія при створенні нового завдання доставки. Метод дозволяє врахувати декілька різнорідних факторів шляхом їх нормалізації та зважування згідно з пріоритетами компанії. Система використовує три критерії для оцінки придатності водія до виконання конкретного завдання:

Перший критерій, відстань до точки завантаження, визначає наскільки далеко знаходиться водій від місця початку доставки. Менша відстань означає швидше прибуття до точки завантаження, економію палива та зменшення загального часу виконання завдання.

Другий критерій, завантаженість водія, визначається як кількість активних завдань, що на даний момент призначені водію. Критерій дозволяє уникнути перевантаження окремих водіїв та забезпечити рівномірний розподіл завдань між всіма доступними виконавцями.

Третій критерій, досвід водія, розраховується на основі кількості успішно виконаних рейсів. Досвідченіші водії краще знають дороги та ефективніше долають непередбачені ситуації.

Оскільки критерії мають різні одиниці виміру та діапазони значень, необхідна нормалізація до єдиної шкали для коректного порівняння. Для критеріїв відстані та завантаженості, де менші значення є кращими, застосовується інверсна нормалізація:

$$n\_i = (max - x\_i) / (max - min)$$

де n_i - нормалізоване значення для водія i, x_i - фактичне значення критерію, max та min - максимальне та мінімальне значення серед всіх водіїв. Для критерію досвіду, де більші значення є кращими, застосовується пряма нормалізація:

$$n\_i = (x\_i - min) / (max - min)$$

Результатом є значення від нуля до одиниці, де одиниця відповідає найкращому показнику. Розрахунок загального балу. Після нормалізації обчислюється загальний бал для кожного водія за формулою зваженої суми:

$$S\_i = w_1 \times n_1 + w_2 \times n_2 + w_3 \times n_3$$

де S_i - загальний бал водія i, n_1, n_2, n_3 - нормалізовані значення критеріїв відстані, завантаженості та досвіду, w_1, w_2, w_3 - вагові коефіцієнти. Вагові коефіцієнти визначають відносну важливість кожного критерію та задовольняють умову:

$$w_1 + w_2 + w_3 = 1$$

Для стандартних доставок система використовує коефіцієнти: п'ять десятих для відстані, три десятих для завантаженості та два десятих для досвіду. Така конфігурація надає найбільшу вагу відстані, оскільки близькість водія до точки завантаження є критичним фактором для своєчасного початку доставки. Після розрахунку балів система ранжує водіїв за спаданням та відбирає трьох найкращих кандидатів для представлення диспетчеру.

Формула Haversine [12] є класичним алгоритмом обчислювальної геометрії та геодезії для визначення відстані між двома точками на поверхні

сфери за їх географічними координатами. Використовується в системі для розрахунку відстані від поточної локації водія до точки завантаження в рамках алгоритму WSM. Формула має вигляд:

$$a = \sin^2(\Delta\varphi/2) + \cos(\varphi_1) \times \cos(\varphi_2) \times \sin^2(\Delta\lambda/2)$$
$$c = 2 \times \operatorname{atan2}(\sqrt{a}, \sqrt{(1-a)})$$
$$d = R \times c$$

де φ_1, φ_2 - широта першої та другої точки в радіанах, $\Delta\varphi$ - різниця широт, $\Delta\lambda$ - різниця довгот, R - радіус Землі (шість тисяч триста сімдесят один кілометр), d - відстань між точками. Формула враховує кривизну Землі, що забезпечує точність обчислення відстаней для координат, розташованих на значній відстані одна від одної. Для коротких відстаней до ста кілометрів похибка не перевищує п'ятсот метрів, що є прийнятним для задач логістики.

Алгоритм проєкції точки на відрізок є прикладним алгоритмом обчислювальної геометрії, що використовується для детекції відхилення водія від побудованого маршруту під час навігації в реальному часі. Маршрут представлений як послідовність відрізків між послідовними точками координат. Для поточної позиції водія система знаходить найближчу точку на маршруті шляхом проєкції координат водія на кожен відрізок маршруту. Для відрізка AB та точки P проєкція обчислюється як:

$$t = ((P - A) \cdot (B - A)) / |B - A|^2$$

де · позначає скалярний добуток векторів, t - параметр проєкції, обмежений діапазоном від нуля до одиниці. Якщо t менше нуля, найближча точка є A. Якщо t більше одиниці, найближча точка є B. Інакше, проєкція обчислюється як:

$$Q = A + t \times (B - A)$$

Відстань від водія до маршруту розраховується як відстань від P до Q. Якщо відстань перевищує п'ятдесят метрів, система повідомляє водія про відхилення від маршруту та пропонує перерозрахунок.

Алгоритм декодування гнучкої полілінії є прикладним алгоритмом з галузі бінарних протоколів та кодування даних. Використовується для розпакування стислого представлення маршруту, отриманого від зовнішнього сервісу маршрутизації.

Полілінія є послідовністю координат, закодованих у компактному текстовому форматі для зменшення обсягу переданих даних. Алгоритм декодування виконує зворотне перетворення стислого рядка в список географічних координат для відображення маршруту на карті. Декодування відбувається поетапно. Кожна координата представлена як різниця з попередньою координатою, помножена на коефіцієнт масштабування. Значення кодується послідовністю п'ятибітових блоків, де шостий біт вказує на продовження числа. Результатом є масив координат з точністю до п'яти знаків після коми, що відповідає точності близько одного метра.

Алгоритм детекції відхилення з періодом охолодження є прикладним алгоритмом для навігації в реальному часі, що запобігає надмірному повідомленню водія про відхилення від маршруту в умовах нестабільного сигналу або складної дорожньої ситуації. Після виявлення відхилення система активує період охолодження тривалістю тридцять секунд, протягом якого повторні повідомлення про відхилення не відправляються. Це дозволяє водію виконати маневр повернення на маршрут або об'їзду перешкоди без постійних переривань від системи. Період охолодження скидається після повернення водія на маршрут або після підтвердження перерозрахунку маршруту. Така логіка забезпечує баланс між інформуванням водія про проблеми та уникненням надмірної кількості сповіщень.

Алгоритм обмеження швидкості запитів з ковзним вікном є системним алгоритмом з галузі розподілених систем та захисту від перевантаження.

Використовується для запобігання зловживанням та забезпечення стабільної роботи серверної частини під навантаженням.

Алгоритм відстежує кількість запитів від кожного користувача протягом ковзного часового вікна. На відміну від фіксованого вікна, ковзне вікно забезпечує більш рівномірний розподіл навантаження та запобігає сплескам на межах інтервалів.

Для кожного запиту система перевіряє кількість попередніх запитів за останню хвилину. Якщо кількість перевищує встановлений ліміт, запит відхиляється з відповідним кодом помилки. Ліміти налаштовані окремо для різних типів операцій: шістдесят запитів на хвилину для звичайних операцій, триста для відправки координат водіями.

Висновок до розділу 2

У другому розділі виконано повне проєктування системи управління вантажними перевезеннями «Delivra» з використанням сучасних методологій та нотацій моделювання програмних систем.

На етапі аналізу вимог сформовано дев'ять користувацьких історій, що охоплюють основну функціональність системи для трьох ролей користувачів: водія, логіста та адміністратора.

Застосування методології Use Case 3.0 через карту користувацьких історій дозволило організувати функціональність у п'ять інкрементів розробки з чіткою пріоритизацією від критично необхідних функцій до додаткових можливостей.

Проєктування архітектури виконано з використанням шести типів діаграм. Діаграми діяльності деталізують три ключові бізнес-процеси: призначення водія через алгоритм WSM, побудову маршруту з урахуванням габаритів вантажівки та повний цикл виконання доставки. Діаграми послідовності показують взаємодію компонентів системи протягом життєвого циклу завдання від автентифікації користувача до формування звіту про

завершену доставку. Діаграма класів відображає доменну модель системи з використанням принципів предметно-орієнтованого проєктування.

Центральною сутністю є завдання доставки, пов'язане з користувачем, компанією, навігаційною сесією та повідомленнями чату. Така структура забезпечує ізоляцію даних між організаціями та гнучке управління доступом на основі ролей.

Діаграма компонентів представляє багатошарову архітектуру серверної частини, організовану в чотири шари: контролери для обробки запитів, сервіси для бізнес-логіки, репозиторії для доступу до даних та компоненти безпеки для автентифікації.

Модель C4 надає багаторівневу візуалізацію архітектури системи. Context Diagram показує систему в контексті трьох типів користувачів та двох зовнішніх сервісів для маршрутизації та електронної пошти. Container Diagram деталізує структуру на веб-додаток для всіх користувачів, серверну частину та базу даних. Component Diagram розкриває внутрішню структуру серверної частини до рівня окремих компонентів. Описано сім алгоритмів та математичних моделей системи.

Алгоритм WSM для автоматичного вибору водія використовує три критерії з ваговими коефіцієнтами п'ять десятих для відстані, три десятих для завантаженості та два десятих для досвіду.

Формула Haversine забезпечує точний розрахунок відстаней між географічними координатами з урахуванням кривизни Землі.

Алгоритми детекції відхилення від маршруту, декодування полілінії та обмеження швидкості запитів забезпечують надійну роботу навігації в реальному часі.

Обґрунтовано вибір алгоритму WSM замість методу k найближчих сусідів через чотири переваги: відсутність потреби в історичних даних, можливість явного контролю пріоритетів критеріїв, лінійну складність обчислень та прозорість результатів для диспетчера.

Результати проєктування забезпечують повну специфікацію системи для переходу до етапу реалізації та слугують основою для розробки серверної і клієнтської частин системи «Delivra».

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ «DELIVRA»

3.1 Реалізація та конструювання програмного продукту

Реалізація системи «Delivra» виконана з використанням сучасних технологій та інструментів для забезпечення високої продуктивності, масштабованості та зручності підтримки програмного продукту.

Серверна частина системи реалізована на мові програмування Java версії сімнадцять з використанням фреймворку Spring Boot версії 3.3. Spring Boot надає готову інфраструктуру для швидкої розробки веб-додатків з вбудованим сервером, автоматичним налаштуванням компонентів та широкою екосистемою бібліотек.

Для забезпечення безпеки використовується Spring Security з реалізацією автентифікації на основі JWT токенів.

Доступ до даних організовано через Spring Data JPA, що надає високорівневі абстракції для роботи з базою даних та автоматичну генерацію запитів на основі методів репозиторіїв.

Клієнтська частина реалізована на бібліотеці React версії 18.2 з використанням TypeScript для статичної типізації коду. TypeScript виявляє помилки на етапі компіляції та покращує підтримку коду через автодоповнення та рефакторинг в середовищі розробки.

Для стилізації інтерфейсу використовується Tailwind CSS, що дозволяє швидко створювати адаптивні компоненти без написання власних стилів.

База даних PostgreSQL версії 15 забезпечує надійне зберігання всіх даних системи. PostgreSQL підтримує складні запити з об'єднаннями та підзапитами, транзакції для забезпечення цілісності даних, індекси для оптимізації продуктивності та розширення для геопросторових даних.

Для побудови маршрутів вантажівок система інтегрується з HERE Maps Truck Routing API, що враховує габаритні обмеження транспортних засобів

при розрахунку оптимальних маршрутів. API надає інформацію про висоту мостів, вагові обмеження доріг та заборони руху для небезпечних вантажів.

Середовище розробки IntelliJ IDEA Ultimate надає потужні інструменти для роботи з Java, Spring та React проєктами.

Система контролю версій Git з хостингом на GitHub забезпечує відстеження змін коду та співпрацю в команді.

Інструменти збірки Maven для серверної частини та npm для клієнтської частини автоматизують управління залежностями та процес компіляції.

Вибір Java та Spring Boot для серверної частини обґрунтовується наступними факторами. Java є зрілою платформою з великою екосистемою бібліотек для вирішення різноманітних задач. Spring Boot значно спрощує конфігурацію додатків та надає готові рішення для типових задач веб-розробки. Продуктивність Java достатня для обробки великої кількості одночасних запитів від користувачів системи.

React обрано для клієнтської частини через компонентний підхід до розробки інтерфейсів, що дозволяє створювати переповторювані елементи та зменшувати дублювання коду. Віртуальний DOM забезпечує ефективне оновлення інтерфейсу при зміні даних. Велика спільнота розробників та наявність готових бібліотек для типових задач прискорюють розробку.

PostgreSQL обрано як систему управління базами даних через підтримку складних типів даних та запитів, надійність транзакцій та можливості масштабування. На відміну від MySQL, PostgreSQL краще підходить для складних аналітичних запитів та має більш строгу відповідність стандарту SQL.

HERE Maps API обрано для маршрутизації через спеціалізовану підтримку вантажівок з урахуванням габаритних обмежень. Альтернативні сервіси, такі як Google Maps, не надають такого детального врахування параметрів вантажних транспортних засобів при побудові маршрутів.

Основною перевагою HERE Maps Truck Routing API є можливість врахування чотирьох категорій обмежень при побудові маршруту. Висота

транспортного засобу дозволяє уникнути маршрутів через низькі мости та тунелі. Ширина враховується при виборі вузьких доріг та проїздів через історичні центри міст. Вага впливає на вибір мостів та естакад з обмеженням навантаження. Класифікація небезпечних вантажів за системою ADR визначає можливість руху через населені пункти та тунелі.

Альтернативні сервіси мають суттєві обмеження. Google Maps Directions API надає тільки базову підтримку режиму вантажівки без можливості вказати конкретну висоту, ширину або вагу транспортного засобу. Mapbox Directions API також не підтримує детальні параметри вантажівок та класифікацію небезпечних вантажів. OpenStreetMap з бібліотекою OSRM є безкоштовним рішенням, але якість даних про обмеження для вантажівок значно нижча через залежність від волонтерського внесення інформації.

Таким чином, HERE Maps API є оптимальним вибором для системи «Delivra» завдяки детальному врахуванню специфіки вантажних перевезень та високій якості даних про дорожні обмеження.

Алгоритм WSM [13] реалізовано в класі (рис. 3.1 та 3.2) DriverRecommendationService. Метод recommendDrivers приймає ідентифікатор завдання та повертає список рекомендованих водіїв з розрахованими балами. Спочатку отримується список всіх доступних водіїв та дані завдання з координатами точки завантаження. Для кожного водія розраховуються три критерії:

```java
@Override  8 usages  Talvin22
public DelivraResponse<ArrayList<DriverRecommendationDTO>> recommendDrivers(Integer taskId, int limit) {
    DeliveryTask task = deliveryTaskRepository.findByIdAndDeletedFalse(taskId)
            .orElseThrow(() -> new NotFoundException(ApiErrorMessage.DELIVERY_NOT_FOUND_BY_ID.getMessage(taskId)));

    List<User> availableDrivers = task.getCompany() != null
            ? userRepository.findAvailableDriversByCompany(task.getCompany().getId())
            : userRepository.findAvailableDrivers();
    log.debug("Found {} available drivers for task {}", availableDrivers.size(), taskId);

    if (availableDrivers.isEmpty()) {
        return DelivraResponse.createSuccessful(new ArrayList<>());
    }

    List<RawMetrics> metrics = availableDrivers.stream() Stream<User>
            .map( User driver -> collectMetrics(driver, task)) Stream<RawMetrics>
            .toList();

    ArrayList<DriverRecommendationDTO> result = applyWeightedScoring(metrics).stream()
            .sorted(Comparator.comparingDouble(DriverRecommendationDTO::getTotalScore).reversed())
            .limit(limit)
            .collect(Collectors.toCollection(ArrayList::new));

    return DelivraResponse.createSuccessful(result);
}
```

Рисунок 3.1 – Метод автоматичної рекомендації водіїв

Джерело: розроблено автором

```java
List<DriverRecommendationDTO> result = new ArrayList<>();
for (RawMetrics m : allMetrics) {
    double proximityScore = computeProximityScore(m.distanceMeters(), maxDistance);
    double workloadScore = computeWorkloadScore(m.activeCount(), maxActive);
    double successRateScore = m.successRate();
    double recencyScore = computeRecencyScore(m.hoursSinceActivity());

    double totalScore = (WEIGHT_PROXIMITY * proximityScore
            + WEIGHT_WORKLOAD * workloadScore
            + WEIGHT_SUCCESS_RATE * successRateScore
            + WEIGHT_RECENCY * recencyScore) * 100.0;

    result.add(DriverRecommendationDTO.builder()
            .driverId(m.driver().getId())
            .driverUsername(m.driver().getUsername())
            .driverEmail(m.driver().getEmail())
            .busy(m.busy())
            .totalScore(Math.round(totalScore * 100.0) / 100.0)
            .proximityScore(Math.round(proximityScore * 10000.0) / 100.0)
            .workloadScore(Math.round(workloadScore * 10000.0) / 100.0)
            .successRateScore(Math.round(successRateScore * 10000.0) / 100.0)
            .recencyScore(Math.round(recencyScore * 10000.0) / 100.0)
            .distanceMeters(m.distanceMeters())
            .pendingTasksCount(m.activeCount())
            .successRate(Math.round(m.successRate() * 10000.0) / 10000.0)
            .hoursSinceLastActivity(m.hoursSinceActivity())
            .build());
}
return result;
```

Рисунок 3.2 – Фрагмент реалізації алгоритму WSM

Джерело: розроблено автором

Відстань обчислюється через метод calculateDistance (Рис. 3.3), що реалізує формулу Haversine для визначення відстані між поточною позицією водія та точкою завантаження. Завантаженість визначається через підрахунок активних завдань водія з статусами призначено та в процесі. Досвід

розраховується на основі кількості завдань зі статусом виконано. Після розрахунку фактичних значень виконується нормалізація до діапазону від нуля до одиниці. Для відстані та завантаженості застосовується інверсна нормалізація, для досвіду пряма. Загальний бал обчислюється як зважена сума нормалізованих критеріїв з коефіцієнтами нуль крапка п'ять для відстані, нуль крапка три для завантаженості та нуль крапка два для досвіду. Результуючий список сортується за спаданням балу та обрізається до трьох найкращих кандидатів. Кожен елемент списку містить ідентифікатор водія, ім'я, розрахований бал та релевантну інформацію для відображення диспетчеру.

```
public static double calculateDistance(double lat1, double lng1, double lat2, double lng2) {  8 usages  Talvin22 *
    double dLat = Math.toRadians(lat2 - lat1);
    double dLng = Math.toRadians(lng2 - lng1);

    double a = Math.sin(dLat / 2) * Math.sin(dLat / 2)
            + Math.cos(Math.toRadians(lat1)) * Math.cos(Math.toRadians(lat2))
            * Math.sin(dLng / 2) * Math.sin(dLng / 2);

    return EARTH_RADIUS_METERS * 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
}
```

Рисунок 3.3 – Реалізація формули Haversine

Джерело: розроблено автором

Побудова маршруту реалізована в класі HereApiService через метод buildRoute (Рис. 3.4). Метод формує запит до HERE Maps API з координатами початку та кінця маршруту, параметрами вантажівки та режимом транспорту. Відповідь містить закодовану полілінію маршруту, загальну відстань та прогнозований час руху. Декодування полілінії виконується методом decodePolyline, що перетворює стислий текстовий формат в масив координат для відображення на карті. Маршрут зберігається в базі даних разом з навігаційною сесією для можливості відстеження відхилення водія від побудованого шляху.

```
    URI uri = UriComponentsBuilder
            .fromHttpUrl(hereApiConfig.getRoutingBaseUrl())
            .queryParam( name: "transportMode",  ...values: "truck")
            .queryParam( name: "origin",  ...values: originLat + "," + originLng)
            .queryParam( name: "destination",  ...values: destLat + "," + destLng)
            .queryParam( name: "return",  ...values: "polyline,summary,actions,instructions")
            .queryParam( name: "truck[grossWeight]", effectiveWeight)
            .queryParam( name: "truck[height]", effectiveHeight)
            .queryParam( name: "truck[width]", effectiveWidth)
            .queryParam( name: "truck[length]", effectiveLength)
            .queryParam( name: "apiKey", hereApiConfig.getApiKey())
            .build()
            .toUri();

    try {
        log.debug("Calculating truck route: ({},{}) -> ({},{})", originLat, originLng, destLat, destLng);
        HereRoutingResponse response = hereRestTemplate.getForObject(uri, HereRoutingResponse.class);

        if (response == null || response.getRoutes() == null || response.getRoutes().isEmpty()) {
            throw new HereApiException(
                    ApiErrorMessage.ROUTING_NO_RESULTS.getMessage(originLat, originLng, destLat, destLng));
        }

        return mapToRouteDTO(response.getRoutes().get(0));
    } catch (RestClientException e) {
        log.error("HERE Routing API call failed: ({},{}) -> ({},{})", originLat, originLng, destLat, destLng, e);
        throw new HereApiException(
                ApiErrorMessage.ROUTING_FAILED.getMessage(originLat, originLng, destLat, destLng), e);
    }
}
```

Рисунок 3.4 – Фрагмент реалізації побудови маршруту

Джерело: розроблено автором

Відстеження координат водія реалізовано через WebSocket з'єднання в класі NavigationWebSocketHandler. Кожні десять секунд клієнтський додаток водія відправляє поточні координати на сервер. Сервер зберігає координати в базі даних та перевіряє відхилення від маршруту через алгоритм проєкції точки на відрізок. Якщо відхилення перевищує п'ятдесят метрів, водію відправляється повідомлення з пропозицією перерозрахувати маршрут. Одночасно оновлені координати транслюються всім підключеним диспетчерам для відображення позиції водія на карті в реальному часі.

База даних системи побудована за принципами реляційної моделі з нормалізацією до третьої нормальної форми для забезпечення цілісності даних та уникнення дублювання інформації. Структура складається з дев'яти основних таблиць, організованих у три логічні групи: управління користувачами та доступом, управління завданнями доставки, допоміжні таблиці для автентифікації та комунікації.

На рис. 3.5 представлено діаграму сутність-зв'язок бази даних системи «Delivra».

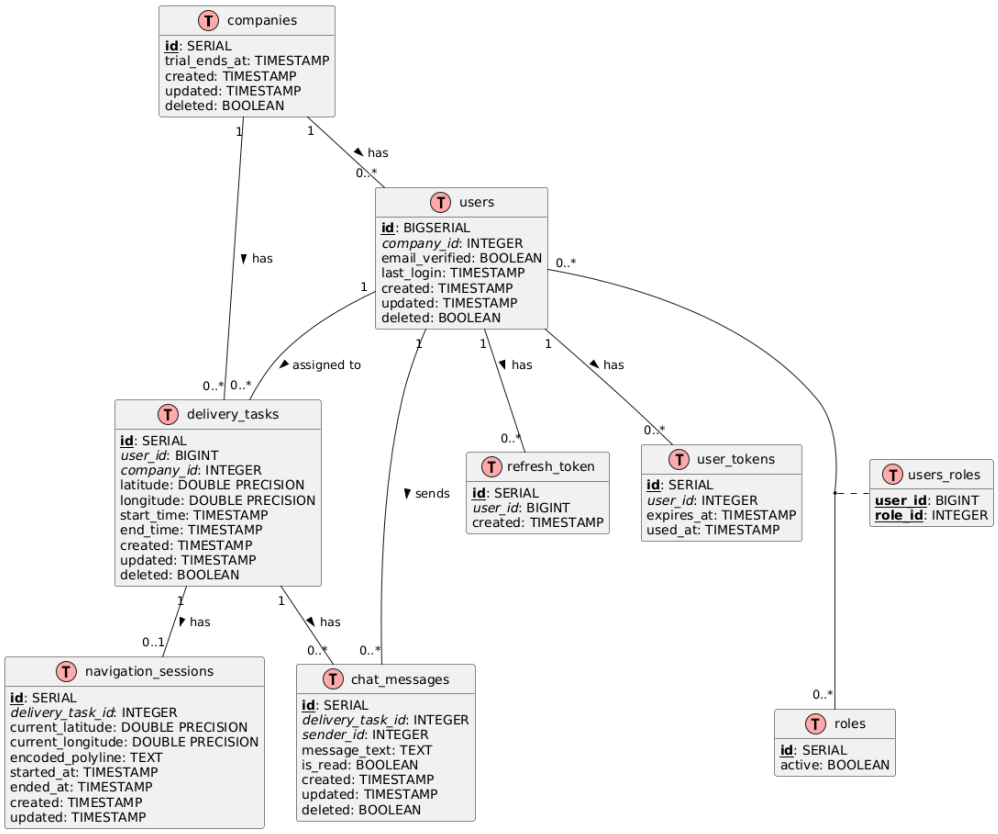

Рисунок 3.5– Діаграма сутність-зв'язок бази даних

Джерело: розроблено автором

Таблиця users зберігає облікові записи користувачів системи. Містить унікальне ім'я користувача, електронну пошту, пароль у захешованому вигляді, статус реєстрації, час останнього входу та прапорець видалення для м'якого видалення записів. Поле company_id пов'язує користувача з компанією для забезпечення багатокомпанійності системи. Поле email_verified вказує чи підтверджено електронну адресу користувача.

Таблиця roles визначає ролі в системі з попередньо заданими правами доступу. Підтримуються чотири ролі: суперадміністратор для управління всією системою, адміністратор для управління компанією, диспетчер для створення завдань та призначення водіїв, водій для виконання доставок. Зв'язок між користувачами та ролями реалізовано через проміжну таблицю users_roles для підтримки множини ролей на одного користувача.

Таблиця companies представляє організації клієнтів системи. Містить назву компанії, електронну пошту, статус підписки та дату закінчення пробного періоду. Статус може бути пробний для безкоштовного використання або активний для платних клієнтів. Ізоляція даних між компаніями забезпечується через зовнішні ключі в таблицях користувачів та завдань.

Таблиця delivery_tasks є центральною сутністю системи та зберігає всі завдання доставки. Містить адресу доставки, координати точки призначення, статус виконання, час початку та завершення, посилання на виконавця через user_id та компанію через company_id. Статус може приймати значення нове, призначено, в процесі або виконано. Поле user_id може бути пустим для нових завдань, що ще не призначені жодному водію.

Таблиця navigation_sessions зберігає дані активної навігації для завдання. Містить посилання на завдання через delivery_task_id, поточні координати водія, закодовану полілінію маршруту, статус сесії та час початку і завершення. Зв'язок один до одного з завданням забезпечує відстеження руху водія під час виконання доставки. Індекс на полі delivery_task_id прискорює пошук активної сесії для конкретного завдання.

Таблиця chat_messages реалізує обмін повідомленнями між диспетчерами та водіями в контексті конкретного завдання. Містить посилання на завдання через delivery_task_id, відправника через sender_id, текст повідомлення, прапорець прочитання та дату створення. Підтримується прикріплення файлів через поля file_url та file_name для передачі документів або зображень. Індекс на полях delivery_task_id та created забезпечує швидке отримання історії повідомлень для завдання.

Таблиця refresh_token зберігає токени оновлення для тривалих сесій користувачів. Містить сам токен, час створення та посилання на користувача. Унікальне обмеження на комбінацію ідентифікатора та користувача запобігає дублюванню токенів. Каскадне видалення забезпечує автоматичне очищення токенів при видаленні користувача.

Таблиця user_tokens використовується для підтвердження електронної пошти та скидання пароля. Містить токен, тип операції, час закінчення дії та час використання. Після використання токен помічається як використаний для запобігання повторному застосуванню. Каскадне видалення видаляє токени разом з користувачем.

Всі таблиці містять службові поля created та updated для відстеження часу створення та останньої зміни записів. Поле deleted реалізує м'яке видалення, що дозволяє зберігати історичні дані для аудиту та аналізу без фізичного видалення записів з бази даних.

Структура бази даних побудована на основі чітко визначених зв'язків між сутностями, що забезпечують цілісність даних та підтримку бізнес-процесів системи.

Зв'язок один до багатьох Companies - Users реалізує багатокомпанійність системи. Одна компанія може мати необмежену кількість користувачів, тоді як кожен користувач належить до однієї компанії через зовнішній ключ company_id. Виняток становлять суперадміністратори з NULL в полі company_id та доступом до всіх компаній. Обмеження ON DELETE RESTRICT запобігає видаленню компанії з активними користувачами.

Зв'язок багато до багатьох Users - Roles реалізовано через проміжну таблицю users_roles з композитним первинним ключем, що запобігає дублюванню призначень. Один користувач може мати декілька ролей одночасно, наприклад диспетчер та водій. Обмеження ON DELETE CASCADE для користувачів видаляє всі призначення ролей, ON DELETE RESTRICT для ролей запобігає видаленню ролі призначеної хоча б одному користувачу.

Зв'язок один до багатьох Companies - DeliveryTasks забезпечує ізоляцію даних між організаціями. Кожне завдання належить до однієї компанії через company_id, що автоматично заповнюється ідентифікатором компанії диспетчера при створенні завдання. Користувачі бачать тільки завдання своєї організації.

Зв'язок один до багатьох Users - DeliveryTasks реалізує призначення водіїв до доставок. Особливістю є можливість NULL в полі user_id для нових завдань зі статусом NEW. Після призначення водія поле заповнюється його ідентифікатором. Обмеження ON DELETE SET NULL зберігає історію завдань навіть після видалення користувача-водія.

Зв'язок один до одного DeliveryTasks - NavigationSessions відстежує рух водія під час доставки. Кожне завдання має тільки одну активну сесію через унікальне обмеження на delivery_task_id. Сесія зберігає закодований маршрут та поточні координати що оновлюються кожні десять секунд. Обмеження ON DELETE CASCADE видаляє сесію разом з завданням.

Зв'язок один до багатьох DeliveryTasks - ChatMessages організує комунікацію в контексті доставки. Композитний індекс на полях delivery_task_id та created оптимізує отримання історії повідомлень відсортованої за часом.

Другий зв'язок Users - ChatMessages через sender_id ідентифікує відправника для візуального розмежування повідомлень диспетчера та водія.

Зв'язок один до багатьох Users - RefreshToken реалізує тривалі сесії без повторного введення пароля. Користувач може мати декілька токенів для різних пристроїв.

Зв'язок Users - UserTokens зберігає одноразові токени для підтвердження пошти та скидання пароля з обмеженим терміном дії двадцять чотири години та одна година відповідно.

Всі зовнішні ключі налаштовані з відповідними правилами цілісності. Каскадне видалення ON DELETE CASCADE застосовується для залежних сутностей без самостійного значення. Встановлення NULL через ON DELETE SET NULL зберігає історичні дані. Обмеження ON DELETE RESTRICT запобігає видаленню записів з активними залежностями. Унікальні обмеження на критичних полях запобігають дублюванню даних.

Зовнішні ключі з каскадним видаленням забезпечують автоматичне очищення залежних записів. Наприклад, при видаленні завдання автоматично

видаляються пов'язані навігаційні сесії та повідомлення чату. Індекси на часто використовуваних полях для пошуку та з'єднань прискорюють виконання запитів навіть при великих обсягах даних.

3.2 Тестування програмного продукту

Тестування системи «Delivra» виконано на двох рівнях: функціональне тестування для перевірки відповідності заявленим вимогам та модульне тестування для перевірки коректності роботи окремих компонентів. Обидва рівні тестування забезпечують високу якість програмного продукту та впевненість у стабільності системи.

Функціональне тестування виконано для перевірки реалізації користувацьких історій та основних сценаріїв використання системи. Тестування проведено вручну з перевіркою всіх критичних функцій для трьох ролей користувачів.

На таблиці 3.1 представлено основні тест-кейси функціонального тестування системи «Delivra».

Таблиця 3.1 – Функціональне тестування системи «Delivra»

№	Сценарій тестування	Очікуваний результат	Результат
1	Реєстрація нового користувача через форму з валідними даними	Користувач створений, отримано email з підтвердженням, можливий вхід в систему	Успішно
2	Автентифікація користувача з правильними обліковими даними	Отримано JWT токен, перенаправлення на головну сторінку, відображення ролі	Успішно
3	Створення завдання доставки диспетчером з вказанням адреси та параметрів вантажу	Завдання збережено зі статусом «нове», координати отримано через геокодування	Успішно

4	Отримання рекомендацій водіїв для завдання через алгоритм WSM	Відображено трьох найкращих водіїв з балами WSM, відстанню та завантаженістю	Успішно
5	Призначення водія до завдання диспетчером зі списку рекомендацій	Статус завдання змінено на «призначено» водій отримав email повідомлення	Успішно
6	Розпочинання навігації водієм з урахуванням габаритів вантажівки	Маршрут побудовано з урахуванням висоти та ваги вантажівки, статус «в процесі»	Успішно
7	Відстеження координат водія у реальному часі під час доставки	Позиція водія на карті диспетчера оновлюється кожні 10 секунд	Успішно
8	Детекція відхилення водія від маршруту	Водій отримує повідомлення про відхилення від маршруту з пропозицією перерозрахунку маршруту	Успішно
9	Відправка повідомлення в чаті між диспетчером та водієм	Повідомлення збережено, отримувач бачить непрочитане повідомлення	Успішно
10	Прикріплення файлу до повідомлення чату	Файл завантажено на сервер, отримувач може завантажити файл	Успішно
11	Зміна статусу завдання водієм на «Доставлено»	Статус оновлено до «Виконано», збережено час завершення, диспетчер отримав email	Успішно
12	Формування звітів по виконаних доставках за період	Звіт згенеровано з інформацією про відстань, час та витрати палива	Успішно

Всі дванадцять тест-кейсів виконано успішно, що підтверджує відповідність реалізованої системи заявленим вимогам та користувацьким історіям. Система коректно обробляє основні сценарії використання для всіх трьох ролей користувачів: диспетчера, водія та адміністратора.

Особливу увагу приділено тестуванню критичних функцій відстеження в реальному часі та детекції відхилення від маршруту, оскільки ці функції безпосередньо впливають на безпеку та ефективність доставок. Перевірено коректність роботи алгоритму WSM для різних комбінацій параметрів водіїв та завдань.

Модульне тестування виконано з використанням фреймворку JUnit 5 та бібліотеки Mockito для ізоляції компонентів. Розроблено дев'яносто дев'ять модульних тестів що покривають критичні компоненти системи.

Клас DriverRecommendationServiceTest містить вісім тестів що перевіряють коректність роботи алгоритму WSM для автоматичного вибору оптимального водія. Тести покривають основні сценарії роботи алгоритму.

Тест recommendDrivers_taskNotFound_throwsNotFoundException перевіряє обробку ситуації коли завдання не знайдено в базі даних. Очікується викидання винятку NotFoundException з відповідним повідомленням про помилку.

Тест recommendDrivers_noAvailableDrivers_returnsEmptyList перевіряє випадок коли в системі немає доступних водіїв для призначення. Алгоритм повертає порожній список рекомендацій без помилок.

Тест recommendDrivers_nearerDriverRankedHigher перевіряє що водій який знаходиться ближче до точки завантаження отримує вищий бал та ранжується першим у списку рекомендацій. Перший водій розташований на відстані близько десяти кілометрів від завдання, другий на відстані понад дві тисячі кілометрів. Система коректно визначає ближчого водія як оптимального.

Тест recommendDrivers_driverWithLessWorkloadScoresHigher перевіряє що водій з меншою кількістю активних завдань отримує вищий бал при однаковій відстані до завдання. Перший водій не має активних завдань, другий має п'ять активних завдань. Алгоритм коректно надає перевагу менш завантаженому водію.

Тест recommendDrivers_limitApplied перевіряє що система повертає не більше запитаної кількості рекомендацій навіть якщо доступних водіїв більше. При запиті однієї рекомендації система повертає тільки одного найкращого водія.

Клас NavigationServiceImplTest містить двадцять один тест що покривають функціонал навігації, відстеження координат та детекції відхилення від маршруту.

Тест startNavigation_taskNotFound_throwsNotFoundException перевіряє обробку спроби розпочати навігацію для неіснуючого завдання. Система коректно викидає виняток з повідомленням про відсутність завдання.

Тест startNavigation_notTaskOwner_throwsInvalidDataException перевіряє що тільки призначений водій може розпочати навігацію для завдання. Спроба розпочати навігацію іншим користувачем призводить до помилки доступу.

Тест startNavigation_alreadyActiveSession_throwsInvalidDataException перевіряє що неможливо розпочати другу навігаційну сесію для завдання що вже має активну сесію. Це запобігає конфліктам при одночасній навігації.

Тест startNavigation_missingCoordinates_geocodesOnTheFly перевіряє автоматичне геокодування адреси якщо координати завдання не заповнені. Система викликає зовнішній сервіс геокодування та заповнює координати перед побудовою маршруту.

Тест handlePositionUpdate_driverFarOffRoute_triggersReroute перевіряє детекцію відхилення водія від побудованого маршруту. Коли відстань від поточної позиції до маршруту перевищує двісті метрів, система автоматично перераховує маршрут від поточної локації до точки призначення.

Тест handlePositionUpdate_rerouteCooldownActive_skipsReroute перевіряє механізм періоду охолодження що запобігає надмірній кількості перерозрахунків маршруту. Після першого перерозрахунку система ігнорує наступні відхилення протягом тридцяти секунд.

Клас DeliveryTaskServiceImplTest містить вісімнадцять тестів що покривають створення, оновлення та управління завданнями доставки.

Тест createDeliveryTask_withDriver_isDriver_assignsAndNotifies перевіряє створення завдання з одразу призначеним водієм. Система перевіряє що користувач має роль водія, призначає завдання та відправляє email повідомлення.

Тест createDeliveryTask_withDriver_notDriver_throwsNotDriverException перевіряє що неможливо призначити завдання користувачу без ролі водія. Спроба призначити завдання диспетчеру або адміністратору призводить до помилки.

Тест createDeliveryTask_noCoordinates_geocodesAddress перевіряє автоматичне отримання координат через геокодування адреси. Якщо координати не вказані при створенні завдання, система викликає сервіс геокодування та заповнює широту і довготу автоматично.

Тест updateDeliveryTaskById_setsInProgressAndStartTime перевіряє що при зміні статусу завдання на "в процесі" автоматично зберігається час початку виконання для подальшого розрахунку тривалості доставки.

Тест updateDeliveryTaskById_setsCompletedAndEndTime перевіряє що при зміні статусу на "виконано" зберігається час завершення та відправляється email повідомлення водію про зміну статусу.

Клас ChatServiceImplTest містить тринадцять тестів що покривають обмін повідомленнями та прикріплення файлів в контексті завдань доставки.

Тест sendMessage_valid_savesAndNotifies перевіряє успішне відправлення текстового повідомлення. Система зберігає повідомлення в базі даних з посиланням на відправника та завдання, відправляє email повідомлення отримувачу.

Тест uploadFile_emptyFile_throwsInvalidDataException перевіряє валідацію розміру файлу перед завантаженням. Порожні файли відхиляються з відповідним повідомленням про помилку.

Тест uploadFile_forbiddenExtension_throwsInvalidDataException перевіряє білий список дозволених розширень файлів. Файли з розширеннями exe, bat, sh та інші потенційно небезпечні формати відхиляються системою.

Тест uploadFile_validFile_savesMessageWithFileUrl перевіряє успішне завантаження дозволеного файлу. Система генерує унікальне ім'я файлу, зберігає на диску та створює повідомлення чату з посиланням на файл.

Клас GeoUtilsTest містить дев'ять тестів що перевіряють коректність геометричних обчислень.

Тест calculateDistance_samePoint_returnsZero перевіряє що відстань між однаковими координатами дорівнює нулю.

Тест distanceMeters_knownDistance_withinTolerance перевіряє точність обчислення відстані між Києвом та Львовом. Очікувана відстань близько шестисот тридцяти чотирьох кілометрів, обчислена відстань знаходиться в межах похибки.

Тест minDistanceToPolyline_pointOnSegment_returnsNearZero перевіряє обчислення відстані від точки що лежить на відрізку. Відстань повинна бути близькою до нуля з невеликою похибкою через неточність обчислень з числами з плаваючою комою.

Клас FlexiblePolylineDecoderTest містить сім тестів що перевіряють декодування стислого формату полілінії. Тест decode_validPolyline_returnsWaypoints перевіряє успішне декодування валідного рядка в список координат.

Тест decode_waypointCoordinatesAreInValidRange перевіряє що всі декодовані координати знаходяться в валідних діапазонах широти від мінус дев'яноста до дев'яноста та довготи від мінус ста вісімдесяти до ста вісімдесяти градусів.

Клас PasswordUtilsTest містить дванадцять тестів валідації паролів. Тести перевіряють що пароль повинен містити мінімум вісім символів, хоча б одну велику літеру, одну малу літеру, одну цифру та один спеціальний символ.

Також перевіряється відхилення паролів з заборонених символів поза ASCII діапазоном.

Клас ApiUtilsTest містить десять тестів допоміжних функцій для роботи з автентифікацією.

Тест createAuthCookie_setsExpectedAttributes перевіряє правильне налаштування cookie для збереження JWT токену з прапорцями HttpOnly та Secure для захисту від XSS атак. Тест getUserIdFromAuthentication_parsesTokenAndReturnsUserId перевіряє витягування ідентифікатора користувача з JWT токену з контексту безпеки.

Всі дев'яносто дев'ять модульних тестів виконуються успішно при кожній зміні коду через систему автоматизованого тестування. Модульні тести забезпечують швидке виявлення регресій при внесенні змін в існуючий код.

Покриття коду тестами становить близько вісімдесяти відсотків для критичних компонентів системи. Найвище покриття досягнуто для сервісів бізнес-логіки, включаючи DriverRecommendationService, NavigationService та DeliveryTaskService. Утилітарні класи для геометричних обчислень та декодування полілінії покриті на сто відсотків.

Використання бібліотеки Mockito для створення mock об'єктів дозволяє ізолювати тестовані компоненти від зовнішніх залежностей таких як база даних, зовнішні API та сервіси електронної пошти. Це забезпечує швидке виконання тестів без необхідності підняття повної інфраструктури та робить тести детерміністичними незалежно від зовнішніх факторів.

Інтеграційний тест DelivraApplicationTests перевіряє успішне завантаження контексту Spring Boot з усіма налаштованими компонентами. Цей тест виявляє помилки конфігурації що можуть виникнути при додаванні нових залежностей або зміні налаштувань додатку.

3.3 Використання програмного продукту

Система «Delivra» надає веб-інтерфейс для чотирьох ролей користувачів: диспетчера, водія, адміністратора та супер адміністратора (Адміністратора

всієї системи). Кожна роль має власний набір функцій адаптований до специфіки виконуваних завдань. Інтерфейс розроблено з акцентом на швидкість доступу до критичної інформації та мінімізацію кроків для виконання типових операцій.

На рис. 3.6 зображено головну сторінку диспетчера з картою міста та списком завдань доставки. Ліва панель містить навігаційне меню з двома основними розділами: Map для перегляду завдань на карті та Report для формування звітності. Центральна область відображає інтерактивну карту на базі OpenStreetMap з позначками локацій доставки. Права панель містить список завдань з фільтрами за статусом: All для всіх завдань, In Progress для завдань в процесі виконання, Pending для нових завдань та Completed для завершених доставок. Кожне завдання відображається з номером, адресою доставки та часом створення. Кнопка Create у правому верхньому куті дозволяє швидко створити нове завдання.

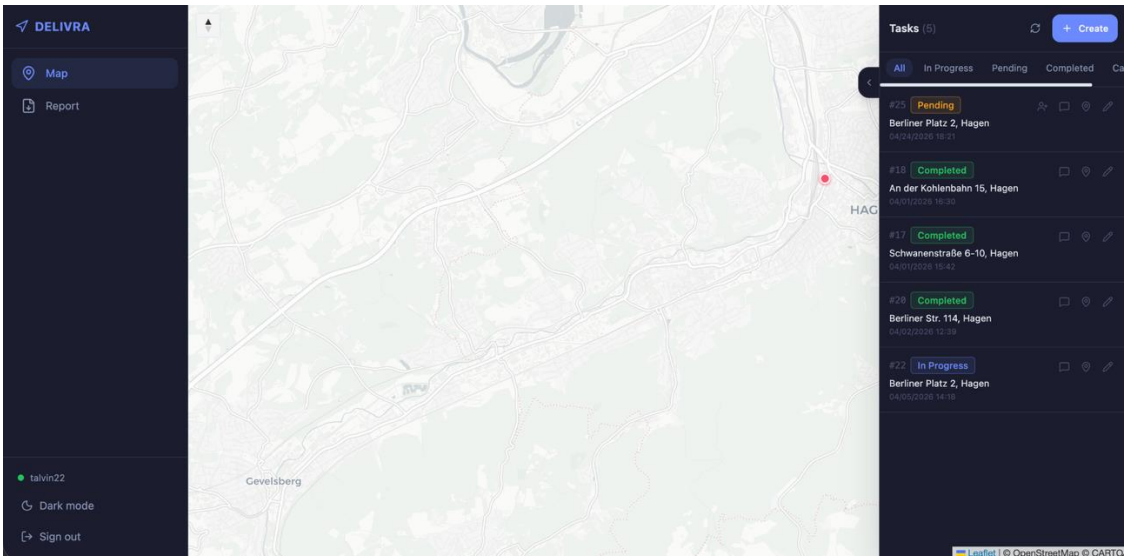

Рисунок 3.6 – Головна сторінка диспетчера

Джерело: розроблено автором

На рис. 3.7 представлено діалогове вікно з рекомендаціями водіїв для завдання згенерованими алгоритмом WSM. Кожна рекомендація відображає ім'я водія, електронну пошту та загальний бал оцінки у великому розмірі

справа. Під основною інформацією показано детальну розбивку балів за чотирма критеріями: Proximity відображає близькість водія до точки завантаження, Workload показує поточну завантаженість водія активними завданнями, Success rate відображає відсоток успішно виконаних доставок, Recency показує як давно водій останній раз виконував завдання. Кожен критерій візуалізовано горизонтальним індикатором з відсотковим значенням. Червоним кольором відзначено відстань від поточної локації водія до точки призначення у кілометрах. Кнопка Assign дозволяє миттєво призначити обраного водія до завдання.

Рисунок 3.7 – Діалогове вікно з рекомендаціями водія для завдання

Джерело: розроблено автором

На рис. 3.8 зображено модальне вікно з деталями завдання. У верхній частині відображається статус, дата та час створення, а також ім'я користувача що створив завдання. Поле Delivery address містить адресу доставки.

Випадаючий список Driver дозволяє обрати або змінити призначеного водія з переліку доступних користувачів з роллю водія. Кнопка Save зберігає внесені зміни та відправляє email повідомлення водію про призначення. Кнопка Cancel task червоного кольору дозволяє скасувати завдання та змінити його статус. Кнопка Delete task видаляє завдання з бази даних через м'яке видалення зі збереженням історії.

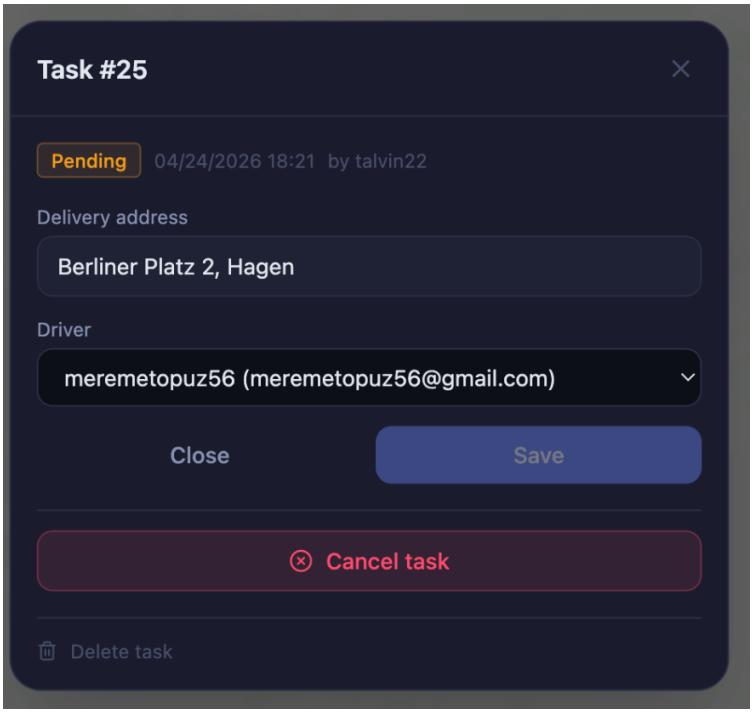

Рисунок 3.8– Модальне вікно з деталями завдання

Джерело: розроблено автором

На рис. 3.9 зображено сторінку Export Report для формування звітів про виконані доставки. Розділ Report contents описує три категорії даних що включаються до звіту. Перша категорія Tasks містить повний перелік завдань з адресою, статусом, призначеним водієм, користувачем що створив завдання та часовими мітками. Друга категорія Summary надає загальний підрахунок та процентний розподіл за статусами: Pending для нових, In Progress для активних, Completed для завершених, Canceled для скасованих завдань. Третя категорія Driver Activity показує кількість завдань по кожному водію

згруповані за статусами. Період звіту фіксовано на останні тридцять днів. Кнопка Download Excel генерує файл формату xlsx з трьома окремими аркушами для кожної категорії даних та автоматично завантажує на комп'ютер користувача.

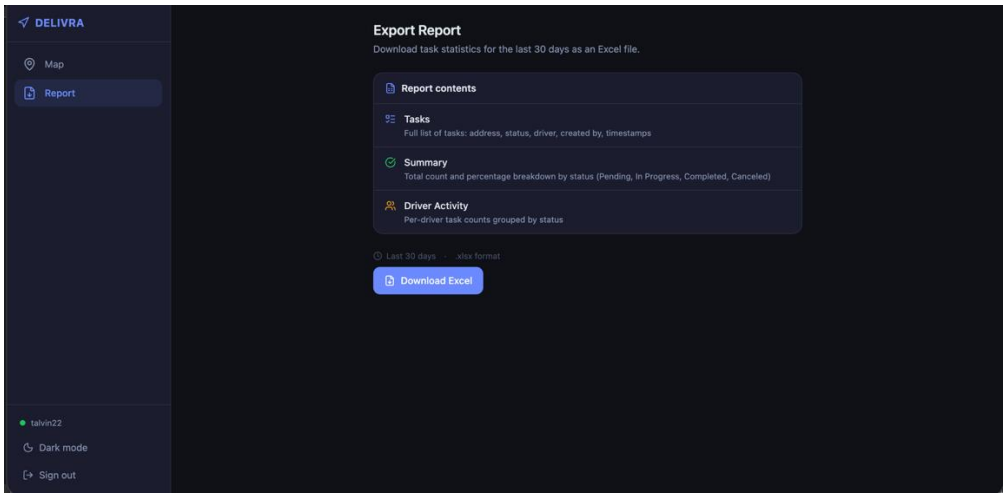

Рисунок 3.9– Сторінка формування звітів

Джерело: розроблено автором

На рис. 3.10 представлено інтерфейс чату між диспетчером та водієм в контексті завдання номер двадцять п'ять. Панель чату відкривається справа поверх карти та містить історію повідомлень з часовими мітками. Повідомлення диспетчера вирівняні праворуч з синім фоном, повідомлення водія вліво з темно-сірим фоном. Під іменем відправника відображається час надсилання у форматі години:хвилини. Поле введення внизу дозволяє написати нове повідомлення, кнопка з іконкою скріпки дає можливість прикріпити файл. Повідомлення відправляються натисканням іконки літака або клавіші Enter. Чат оновлюється в реальному часі через WebSocket з'єднання без необхідності перезавантаження сторінки.

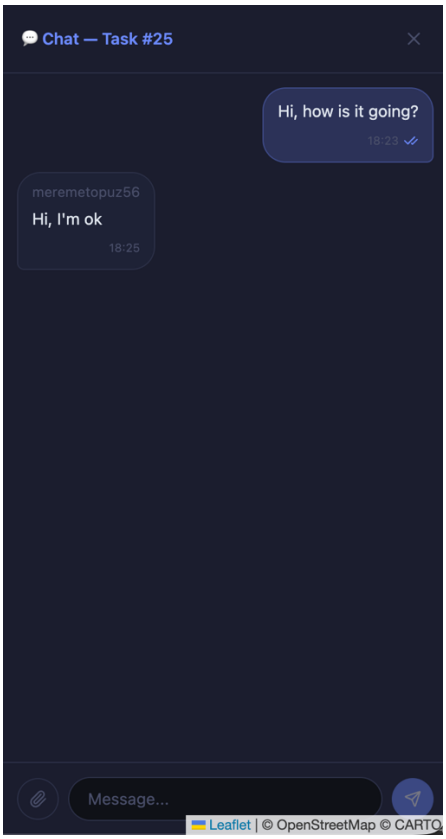

Рисунок 3.10 – Інтерфейс чату між диспетчером та водієм

Джерело: розроблено автором

Веб-інтерфейс водія адаптовано для використання на мобільних пристроях під час виконання доставок. Адаптивний дизайн забезпечує зручність роботи на смартфонах з великими кнопками та спрощеною навігацією.

На рис. 3.11 зображено головну сторінку водія з переліком призначених завдань. Кожне завдання відображається окремою карткою з номером, статусом, адресою та часом створення. Завдання зі статусом "In Progress" мають позначку "Route active" що вказує на активну навігацію. Натискання на картку відкриває деталі завдання.

Рисунок 3.11 – Головна сторінка водія

Джерело: розроблено автором

На рис. 3.12 представлено форму налаштування параметрів транспортного засобу для побудови маршруту з урахуванням габаритних обмежень. Водій вказує загальну вагу, висоту, ширину та довжину вантажівки. Система використовує ці параметри для виключення доріг з низькими мостами, вузькими проїздами або обмеженнями ваги.

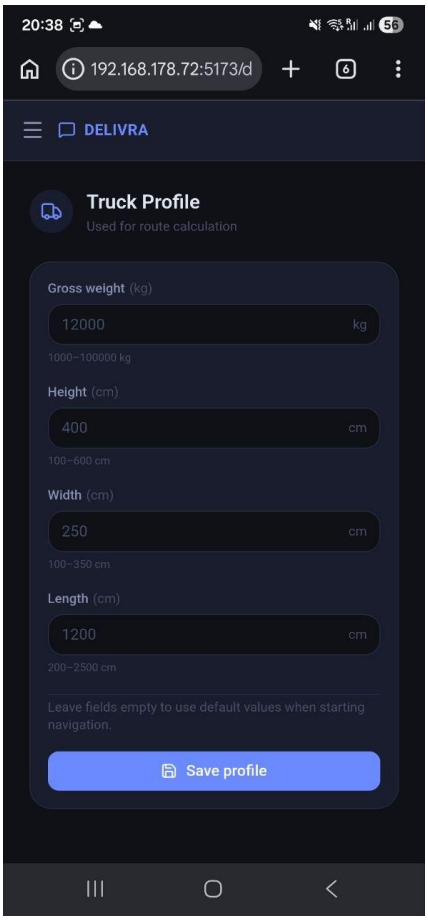

Рисунок 3.12 – Форма налаштування параметрів транспортного засобу

Джерело: розроблено автором

На рис. 3.13 зображено екран деталей завдання з адресою доставки, географічними координатами та часовими мітками створення і початку виконання. Кнопка Continue navigation дозволяє відновити навігацію після перерви. Кнопка Cancel task дозволяє скасувати завдання у виняткових випадках.

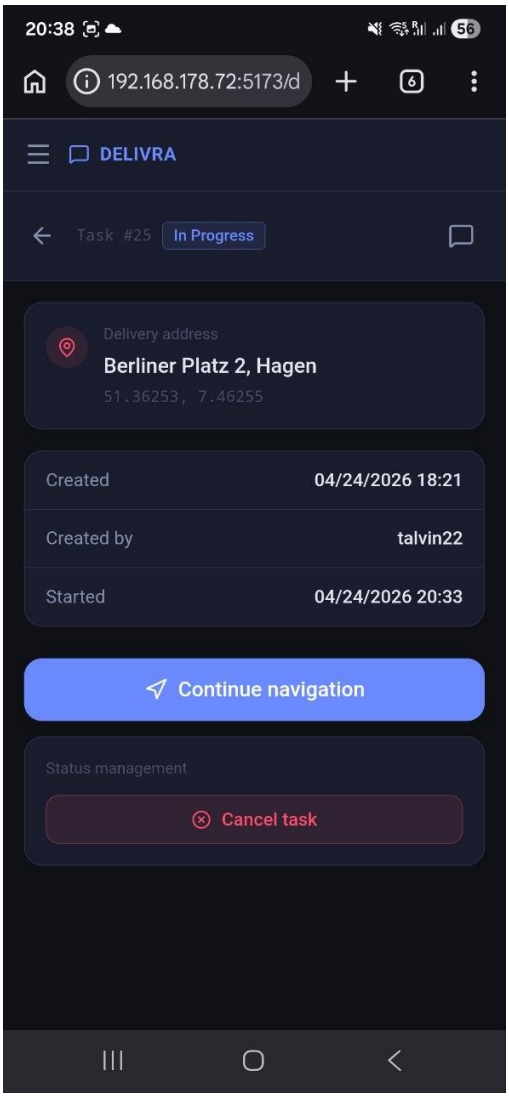

Рисунок 3.13– Екран деталей логістичного завдання

Джерело: розроблено автором

На рис. 3.14 представлено екран активної навігації з побудованим маршрутом на карті. Синя лінія показує оптимальний шлях з урахуванням параметрів вантажівки. Верхня панель містить текстову інструкцію для наступного маневру. Нижня панель відображає залишкову відстань та адресу призначення. Кнопка Chat відкриває чат з диспетчером. Кнопка Delivered стає активною при наближенні до точки призначення та дозволяє підтвердити успішну доставку.

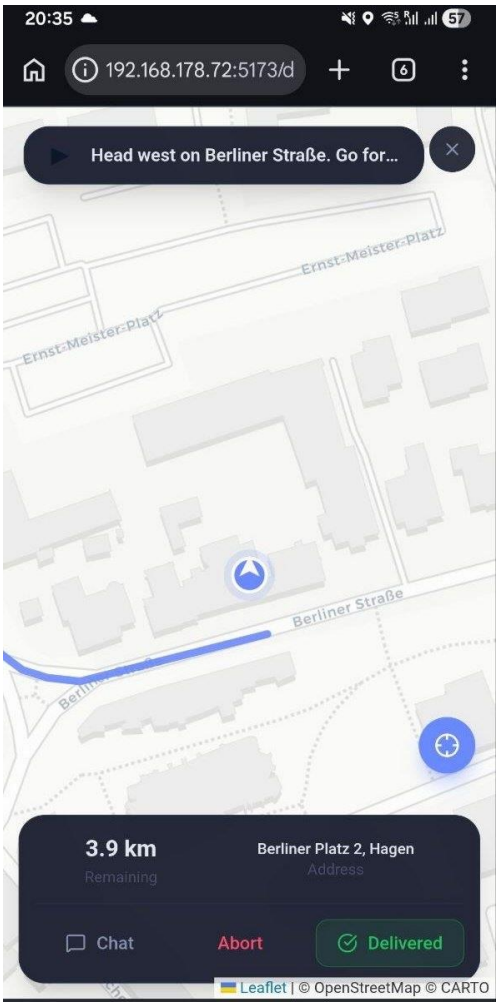

Рисунок 3.14 – Екран активної навігації

Джерело: розроблено автором

Під час навігації веб-додаток автоматично відправляє GPS координати на сервер кожні десять секунд через WebSocket. Коли відстань від поточної позиції до маршруту перевищує двісті метрів водій отримує сповіщення про відхилення з пропозицією перерозрахувати шлях. Механізм періоду охолодження тридцять секунд запобігає надмірним перерозрахункам. Після прибуття водій натискає кнопку Delivered що змінює статус завдання на "виконано" та зберігає час завершення. Диспетчер отримує email повідомлення про успішну доставку.

Супер адміністратор має доступ до глобального управління системою з можливістю перегляду та редагування даних всіх компаній.

На рис. 3.15 зображено головну сторінку суперадміністратора з оглядовою статистикою всієї системи. Чотири картки у верхній частині відображають кількість компаній, користувачів на пробному періоді, загальну кількість користувачів та завдань. Розділ Companies містить список всіх зареєстрованих організацій з назвою, електронною поштою та статусом підписки. Кнопка New company дозволяє створити нову організацію з налаштуванням пробного періоду.

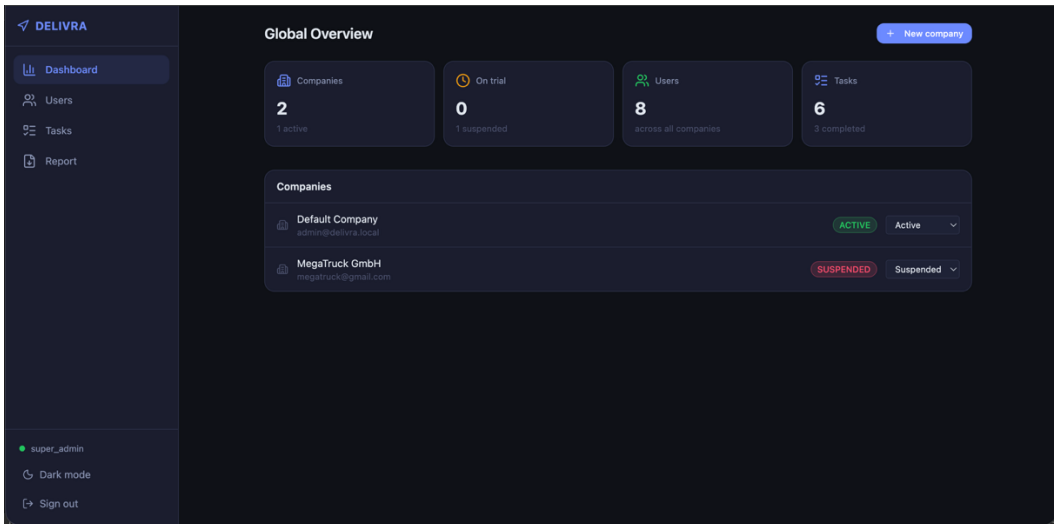

Рисунок 3.15 – Головний екран супер адміністратора

Джерело: розроблено автором

На рис. 3.16 представлено сторінку управління користувачами всіх компаній системи. Таблиця відображає ідентифікатор, ім'я, електронну пошту та призначені ролі кожного користувача. Ролі візуалізовано кольоровими бейджами: SUPER_ADMIN червоного кольору, ADMIN зеленого, DISPATCHER жовтого, DRIVER синього. Поле пошуку дозволяє фільтрувати користувачів за іменем або електронною поштою. Іконки праворуч дозволяють редагувати або видаляти облікові записи.

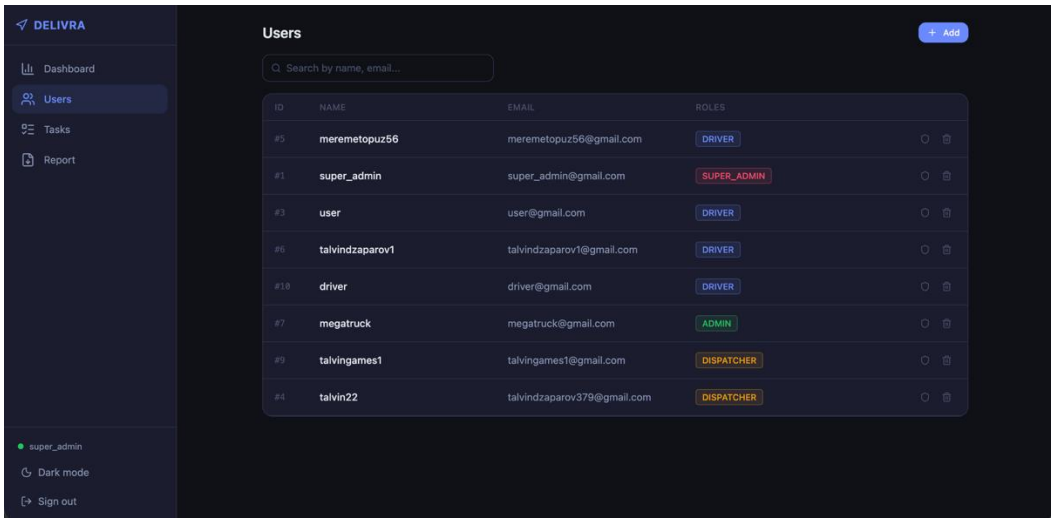

Рисунок 3.16 – Екран управління користувачами

Джерело: розроблено автором

На рис. 3.17 зображено сторінку Tasks з можливістю перегляду завдань доставки всіх організацій. Таблиця містить номер завдання, адресу, статус виконання, призначеного водія та час створення. Фільтри дозволяють відбирати завдання за статусом: All, In Progress, Pending, Completed, Canceled. Суперадміністратор може переглядати деталі будь-якого завдання незалежно від компанії для технічної підтримки.

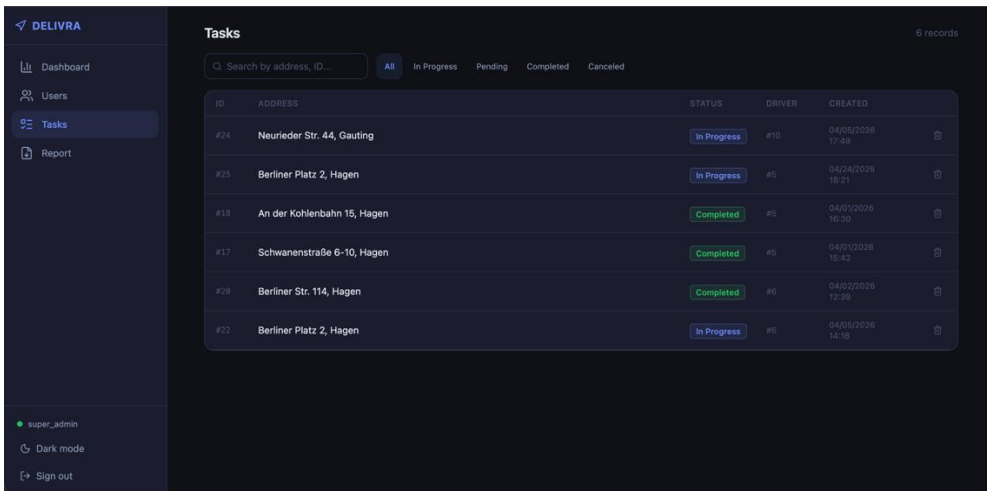

Рисунок 3.17 – Екран управління доставками

Джерело: розроблено автором

На рис. 3.18 представлено сторінку формування звітів з можливістю вибору компанії через випадаючий список Company filter. Опція "All companies" генерує звіт по всіх організаціях з додатковою колонкою Company для ідентифікації належності завдань. Структура звіту містить три розділи: Tasks з повним переліком завдань, Summary з процентним розподілом за статусами, Driver Activity з активністю водіїв. Кнопка Download Excel завантажує файл формату xlsx з трьома аркушами.

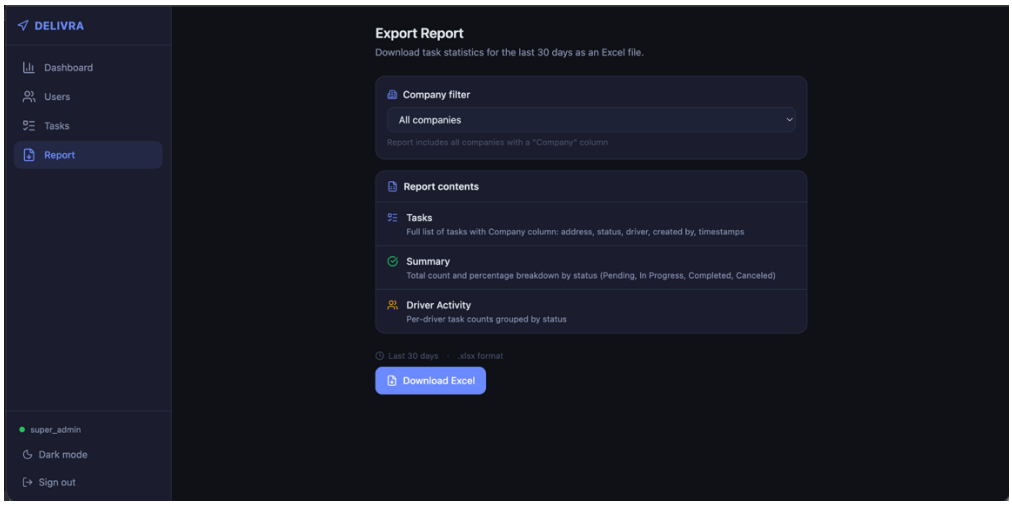

Рисунок 3.18 – Екран формування звітів

Джерело: розроблено автором

Висновок до розділу 3

У третьому розділі представлено результати реалізації системи управління вантажними перевезеннями «Delivra» з детальним описом технологічного стеку, архітектури бази даних та результатів тестування.

Для реалізації серверної частини обрано Java 17 з фреймворком Spring Boot що забезпечує швидку розробку та масштабованість системи. Клієнтська частина побудована на React 18 з TypeScript для типобезпечного коду. База даних PostgreSQL забезпечує надійне зберігання даних з підтримкою складних запитів. Інтеграція з HERE Maps API дозволяє будувати маршрути з

урахуванням габаритних обмежень вантажівок, що є критичним для логістичних компаній.

Структура бази даних складається з дев'яти нормалізованих таблиць з чітко визначеними зв'язками та обмеженнями цілісності. Реалізовано багатокомпанійність через ізоляцію даних на рівні компанії, що дозволяє використовувати систему множиною організацій одночасно. М'яке видалення записів забезпечує збереження історичних даних для аудиту та аналізу.

Тестування системи виконано на двох рівнях. Функціональне тестування підтвердило коректність реалізації всіх дванадцяти основних сценаріїв використання для трьох ролей користувачів. Модульне тестування охопило дев'яносто дев'ять тестових випадків з покриттям близько вісімдесяти відсотків критичних компонентів системи. Використання Mockito для ізоляції залежностей забезпечує швидке виконання тестів та раннє виявлення регресій.

Веб-інтерфейс системи розроблено з акцентом на зручність використання для кожної ролі. Диспетчер отримує повний контроль над завданнями з можливістю відстеження водіїв в реальному часі та комунікації через вбудований чат. Адаптивний веб-інтерфейс водія оптимізовано для мобільних пристроїв з великими елементами управління та навігацією з урахуванням габаритів транспортного засобу. Суперадміністратор має глобальний доступ до даних всіх компаній для технічної підтримки та адміністрування системи.

Реалізована система повністю відповідає заявленим вимогам та готова до впровадження в логістичних компаніях для автоматизації процесів управління доставками та підвищення ефективності операцій.

ВИСНОВКИ

У дипломній роботі виконано повний цикл розробки веб-орієнтованої системи управління вантажними перевезеннями «Delivra» від аналізу вимог до реалізації та тестування програмного продукту.

У першому розділі проведено аналіз предметної області управління вантажними перевезеннями та виявлено основні проблеми існуючих рішень. Встановлено що комерційні системи мають високу вартість впровадження та складність налаштування, тоді як безкоштовні альтернативи не надають функціонал для оптимізації призначення водіїв та побудови маршрутів з урахуванням габаритів транспортних засобів. Визначено функціональні та нефункціональні вимоги до розроблюваної системи з акцентом на автоматизацію процесів призначення водіїв та навігації в реальному часі.

У другому розділі розроблено архітектуру системи на основі клієнт-серверної моделі з REST API для взаємодії компонентів. Обрано тришарову архітектуру серверної частини з розділенням на рівні представлення, бізнес-логіки та доступу до даних. Розроблено дев'ять користувацьких історій що покривають основні сценарії роботи диспетчерів, водіїв та адміністраторів. Створено діаграми прецедентів, активностей, послідовностей, класів та компонентів для документування архітектурних рішень. Спроектовано алгоритм WSM для автоматичної рекомендації водіїв на основі трьох критеріїв: близькості до точки завантаження, поточної завантаженості та недавності виконання завдань.

У третьому розділі реалізовано систему з використанням Java 17, Spring Boot, React 18 та PostgreSQL. Інтегровано HERE Maps API для побудови маршрутів з урахуванням чотирьох категорій обмежень вантажівок: висоти, ширини, ваги та класифікації небезпечних вантажів за системою ADR. Розроблено базу даних з дев'яти нормалізованих таблиць з підтримкою багатокомпанійності через ізоляцію даних. Виконано тестування на двох рівнях: дванадцять функціональних тест-кейсів підтвердили відповідність

всім користувацьким історіям, дев'яносто дев'ять модульних тестів забезпечили покриття близько вісімдесяти відсотків критичних компонентів. Розроблено чотири веб-інтерфейси адаптовані під потреби кожної ролі користувачів з підтримкою мобільних пристроїв для водіїв.

Основні результати дипломної роботи:

- розроблено веб-систему управління вантажними перевезеннями з функціоналом створення завдань, автоматичної рекомендації водіїв, навігації в реальному часі та комунікації між диспетчерами і водіями;
- реалізовано алгоритм WSM для автоматичного вибору оптимального водія з урахуванням трьох критеріїв що скорочує час призначення завдань з кількох хвилин до кількох секунд;
- інтегровано HERE Maps Truck Routing API для побудови маршрутів з урахуванням габаритних обмежень що запобігає потраплянню вантажівок під низькі мости або на дороги з обмеженням ваги;
- розроблено систему відстеження водіїв в реальному часі з автоматичною детекцією відхилення від маршруту більше двохсот метрів та пропозицією перерозрахунку шляху;
- реалізовано багатокомпанійну архітектуру з повною ізоляцією даних що дозволяє використовувати систему множиною організацій одночасно;
- створено адаптивні веб-інтерфейси оптимізовані для роботи на десктопних комп'ютерах диспетчерів та мобільних пристроях водіїв.

Розроблена система повністю відповідає заявленим вимогам та готова до впровадження в логістичних компаніях. Функціонал автоматичної рекомендації водіїв та навігації з урахуванням габаритів транспортних засобів є унікальною комбінацією можливостей що відсутня в безкоштовних альтернативах на ринку.

Перспективи подальшого розвитку системи включають інтеграцію з телематичними пристроями вантажівок для автоматичного отримання даних про пальне та стан транспортного засобу, додавання модуля планування

маршрутів з множиною точок доставки для оптимізації послідовності виконання завдань, розробку мобільних додатків для iOS та Android з офлайн-режимом навігації, впровадження машинного навчання для прогнозування часу доставки на основі історичних даних.

Вихідний код системи опубліковано у відкритому репозиторії GitHub за адресою https://github.com/Talvin22/delivra.

Апробація результатів дослідження здійснювалась під час проведення Міжнародної конференції «Держава, регіони, підприємництво: інформаційні, суспільно-правові, соціально-економічні аспекти розвитку» (2025)та Міжнародної науково-практичної конференції здобувачів вищої освіти і молодих вчених «Інформаційні технології: теорія та практика» (м. Харків 25-27 березня 2026 року)

ПЕРЕЛІК ДЖЕРЕЛ ТА ПОСИЛАННЯ

1. TomTom Live Services URL: https://www.nintendo.com/us/store/products/hollow-knight-switch/ (дата звернення 20.02.2026).
2. Sygic Truck & Caravan GPS Navigation URL: https://www.sygic.com/truckb (дата звернення 21.02.2026).
3. Onfleet URL: https://onfleet.com/ (дата звернення 21.02.2026)
4. HERE Maps Truck Routing API URL: https://www.here.com/ (дата звернення 21.02.2026).
5. Гринченко М., Куценко Д. Моделі бізнес-процесів медичного центру для підвищення ефективності прийняття рішень. Сучасний стан наукових досліджень та технологій в промисловості, 2025 №34 URL: https://doi.org/10.30837/2522-9818.2025.4.005 (дата звернення 21.02.2026).
6. Бульба С. С., Соловйова О. І., Семеренко Ю. О. Дослідження алгоритмів пошуку оптимального шляху. Системи управління, навігації та зв'язку. Збірник наукових праць. 2024. Т. 2, № 76. С. 67–69. URL: https://doi.org/10.26906/sunz.2024.2.067 (дата звернення 21.02.2026).
7. Qu L., Li H. Analysis of distribution path optimization algorithm based on big data technology. Journal of King Saud University - Science. 2022. P. 102019. URL: https://doi.org/10.1016/j.jksus.2022.102019 (дата звернення 21.02.2026).
8. Моделювання за допомогою DDD та архітектурні патерни. Міністерство освіти і науки України. Національний університет «києво-могилянська академія» Гавришко Я. О. URL: https://ekmair.ukma.edu.ua/server/api/core/bitstreams/8c7db7ac-2781-4002-b0df-9b11c92cb5c2/content (Дата звернення 07.04.2026).

9. Що таке User Stories і навіщо вони насправді потрібні? Олександр Кононенко. URL: https://iampm.club/ua/blog/yak-pisati-user-stories-shhob-bulo-zrozumilo-vsim/ (Дата звернення 11.04.2026).
10. Use-case 3.0, The guide to succeeding with use cases. Keith de Mendonca, Dr. Ivar Jacobson. URL: https://www.ivarjacobson.com/publications/books/use-case-30-ebook (Дата звернення 14.04.2026).
11. C4 Model: a Research Guide for Designing Software Architectures. Argyro Mavrogiorgou. URL: https://ieeexplore.ieee.org/document/11087935/authors#authors (Дата звернення 11.04.2026).
12. Measure distance locating nearest public facilities using Haversine. Medi Taruk, Evi Maria, Edy Budiman. URL: https://iopscience.iop.org/article/10.1088/1742-6596/1450/1/012080 (Дата звернення 14.04.2026).
13. Insights into Weighted Sum Sampling Approaches for multi criteria decision making problems. Alled Williams, Yilun Cai URL: https://www.researchgate.net/publication/391391617_Insights_into_Weighted_Sum_Sampling_Approaches_for_Multi-Criteria_Decision_Making_Problems (Дата звернення 14.04.2026).