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

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

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Комп'ютерна система керування процесами вантажних перевезень із використанням платформи n8n

Виконав: студент 4 курсу, групи СІ-41
спеціальності 123 «Комп'ютерна інженерія»
(шифр і назва спеціальності)

(підпис) *Кармазин О.Б.* (прізвище та ініціали)

Керівник (підпис) *Стадник Н.Б.* (прізвище та ініціали)

Нормоконтроль (підпис) *Луцик Н.С.* (прізвище та ініціали)

Завідувач кафедри (підпис) *Осухівська Г.М.* (прізвище та ініціали)

Рецензент (підпис) *Деркач М.В.* (прізвище та ініціали)

Тернопіль
2026

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

Факультет *комп'ютерно-інформаційних систем і програмної інженерії*
(повна назва факультету)

Кафедра *комп'ютерних систем та мереж*
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ *Осухівська Г.М.*
(підпис) (прізвище та ініціали)
«25» квітня 2026 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня *бакалавр*
(назва освітнього ступеня)

за спеціальністю *123 «Комп'ютерна інженерія»*
(шифр і назва спеціальності)

студента *Кармазина Олександра Богдановича*
(прізвище, ім'я, по батькові)

1. Тема роботи Комп'ютерна система керування процесами вантажних перевезень із використанням платформи n8n

Керівник роботи *Стадник Наталія Богданівна, к.т.н. ст. викл.*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від
« 24 » квітня 2026 року № 4/9-188
2. Термін подання студентом завершеної роботи *« 19 » червня 2026 року*
3. Вихідні дані до роботи *Технічне завдання*

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1. Аналіз технічного завдання
2. Проєктна частина
3. Практична частина
4. Безпека життєдіяльності, основи охорони праці.
Висновки
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Сруктурна схема
2. Схема електрична принципова
3. Макет системи
4. Схема електрична функціональна

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	*доц. каф. МТ Сенчишин В.С.*		

7. Дата видачі завдання *25.04.2026 р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	*Розробка технічного завдання*	*26.01 – 02.02*	
2.	*Робота над першим розділом «Аналіз технічного завдання»*	*03.02 – 15.02*	
3.	*Робота над другим розділом «Проєктна частина»*	*20.04 – 02.05*	
4.	*Робота над третім розділом «Практична частина»*	*02.05 – 20.05*	
5.	*Робота над четвертим розділом «Безпека життєдіяльності, основи охорони праці»*	*20.05 – 25.05*	
6.	*Оформлення пояснювальної записки і графічного матеріалу*	*26.05 – 7.06*	
7.	*Перевірка на академічний плагіат, перевірка керівником та консультантами*	*8.06 – 14.06*	
8.	*Попередній захист кваліфікаційної роботи бакалавра*	*15.06 – 21.06*	
9.	*Захист кваліфікаційної роботи бакалавра*	*26.06*	

Студент _____ *Кармазин О.Б*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Стадник Н. Б.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кармазин О.Б. Комп’ютерна система керування процесами вантажних перевезень із використанням платформи n8n: робота на здобуття кваліфікаційного ступеня бакалавра: спец. 123 — комп’ютерна інженерія. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2026.

Ключові слова: комп'ютерна система, вантажні перевезення, n8n, автоматизація бізнес-процесів, GPS-трекер, мікроконтролер, моніторинг місцезнаходження, webhook, Spring Boot.

У даній роботі спроєктовано та розроблено комп'ютерну систему керування процесами вантажних перевезень із використанням платформи автоматизації бізнес-процесів n8n, яка забезпечує координацію водіїв і замовлень та автоматичне сповіщення про ключові події диспетчерського процесу. Спроєктовано багатошарову архітектуру, до складу якої входять серверна частина на основі Spring Boot, база даних PostgreSQL, платформа оркестрації подій n8n, клієнтський веб-застосунок на React та апаратний модуль моніторингу місцезнаходження водія. Серверну частину реалізовано з автентифікацією на основі JWT, рольовим розмежуванням доступу та аспектно-орієнтованим (AOP) механізмом аудиту й генерації webhook-подій, які передаються до n8n і маршрутизуються до сповіщень у Telegram та електронною поштою. Апаратний модуль на базі мікроконтролера ESP32 із приймачем NEO-6M періодично передає координати транспортного засобу до серверної частини, де виконується перевірка наближення водія до пункту призначення замовлення з автоматичним сповіщенням. У результаті отримано цілісне програмно-апаратне рішення, що поєднує серверну логіку, оркестрацію подій та апаратний рівень збору геоданих.

ANNOTATION

Karmazyn O.B. Computer System for Managing Freight Transportation Processes Using the n8n Platform: Bachelor’s Graduation Thesis: speciality 123 — computer engineering. Ternopil: Ternopil Ivan Puluj National Technical University, 2026.

Keywords: computer system, freight transportation, n8n, business process automation, GPS tracker, microcontroller, location monitoring, webhook, Spring Boot.

This thesis presents the design and development of a computer system for managing freight transportation processes using the n8n business process automation platform, which provides coordination of drivers and orders and automatic notification of key dispatching events. A multi-layered architecture was designed, comprising a Spring Boot server side, a PostgreSQL database, the n8n event orchestration platform, a React web client, and a hardware module for monitoring the driver's location. The server side was implemented with JWT-based authentication, role-based access control, and an aspect-oriented (AOP) mechanism for auditing and generating webhook events, which are passed to n8n and routed to notifications via Telegram and email. The hardware module, based on an ESP32 microcontroller with a NEO-6M receiver, periodically transmits the vehicle's coordinates to the server side, where the driver's proximity to the order's destination is checked with automatic notification. As a result, an integrated software-hardware solution was obtained that combines server-side logic, event orchestration, and a hardware layer for geodata collection.

ЗМІСТ

СПИСОК СКОРОЧЕНЬ

AOP– Aspect-Oriented Programming

API – Application Programming Interface

BPMN – Business Process Model and Notation

CRUD – Create, Read, Update, Delete

DBMS – Database Management System

DTO – Data Transfer Object

GPIO – General-Purpose Input/Output

GPS – Global Positioning System

HMAC – Hash-based Message Authentication Code

HTTP – HyperText Transfer Protocol

I²C – Inter-Integrated Circuit

JPA – Java Persistence API

JSON – JavaScript Object Notation

JWT – JSON Web Token

NVS – Non-Volatile Storage

ORM – Object-Relational Mapping

RBAC – Role-Based Access Control

REST – Representational State Transfer

SPA – Single Page Application

TMS – Transport Management System

UART – Universal Asynchronous Receiver/Transmitter

UUID – Universally Unique Identifier

ВСТУП

Логістична галузь є однією з ключових для сучасної економіки, а ефективна організація вантажних перевезень безпосередньо впливає на конкурентоспроможність підприємств і своєчасність постачань. В умовах зростання обсягів перевезень традиційні методи ручного управління диспетчерськими процесами стають менш ефективними, що зумовлює необхідність впровадження автоматизованих систем для оперативного розподілу замовлень між водіями, відстеження їх виконання та сповіщення відповідальних осіб.

Розроблення подібних комп'ютерних систем у межах кваліфікаційної роботи бакалавра за спеціальністю 123 «Комп'ютерна інженерія» виконується відповідно до загальних вимог до структури та змісту таких робіт [1].

Актуальність теми обумовлена зростанням ринку TMS-систем (Transport Management Systems) та розвитком low-code платформ автоматизації, зокрема n8n, які дозволяють гнучко інтегрувати сервіси без значних витрат на розробку. Поєднання серверних технологій (Spring Boot, JWT, PostgreSQL) із платформою n8n та апаратним рівнем збору геоданих відкриває можливості для побудови практичних, масштабованих систем керування вантажними перевезеннями.

Метою кваліфікаційної роботи є проєктування та розробка комп'ютерної системи керування процесами вантажних перевезень із використанням платформи n8n, що забезпечує ефективну координацію водіїв і замовлень шляхом інтеграції серверної платформи на основі Spring Boot, реляційної DBMS PostgreSQL, платформи автоматизації бізнес-процесів n8n та апаратного модуля моніторингу місцезнаходження водіїв на базі мікроконтролера ESP32.

Для досягнення поставленої мети необхідно вирішити такі задачі:

– провести аналіз предметної галузі та існуючих програмних рішень для диспетчеризації вантажних перевезень;

– визначити вимоги до системи та обрати відповідний технологічний стек;

– спроєктувати архітектуру серверної частини та схему бази даних;

– реалізувати REST API з підтримкою рольової моделі доступу та JWT-автентифікації;

– розробити модуль автоматизації на базі платформи n8n;

– реалізувати систему аудиту та журналювання подій;

– розробити клієнтський інтерфейс для диспетчерів та водіїв;

– розробити апаратний модуль моніторингу місцезнаходження водія (GPS-трекер на базі ESP32) та інтегрувати його із серверною частиною;

– виконати тестування ключових компонентів системи та розгорнути її у контейнерному середовищі Docker.

У роботі використано такі методи дослідження: об'єктно-орієнтований аналіз і проєктування, UML- та BPMN-моделювання, методи розробки RESTful API, методи забезпечення безпеки веб-застосунків (JWT, RBAC) та технологію контейнеризації Docker.

Практична цінність результатів полягає у тому, що розроблена система може бути використана малими та середніми транспортно-логістичними підприємствами для автоматизації диспетчерських процесів, скорочення часу призначення водіїв до замовлень та забезпечення прозорості операцій. Вихідний код розміщено у відкритому репозиторії на платформі GitHub (https://github.com/AveEg0/logistics-dispatch-system).

РОЗДІЛ 1 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ

1.1 Аналіз предметної галузі

Логістика — це комплекс заходів з планування, організації та управління переміщенням матеріальних цінностей від місця їх виробництва або зберігання до кінцевого споживача. Одним із ключових елементів логістичної системи є диспетчеризація - процес оперативного управління транспортними засобами та виконавцями з метою своєчасного і якісного виконання замовлень.

Диспетчерська служба виконує такі основні функції:

– прийом та реєстрація замовлень на перевезення;

– призначення відповідального водія та транспортного засобу;

– відстеження поточного стану виконання замовлень;

– координація дій водіїв та оповіщення зацікавлених сторін;

– ведення журналу операцій та формування звітності.

Традиційно диспетчерські функції виконувалися вручну - телефонними дзвінками, записами в журналах або у таблицях Microsoft Excel. Такий підхід має суттєві недоліки: низька оперативність реагування, висока ймовірність помилок при ручному введенні даних, відсутність прозорості та наскрізного журналювання операцій. Зі зростанням кількості замовлень і розширенням географії перевезень ручне управління стає неефективним і потребує автоматизації.

Сучасний стан ринку програмного забезпечення для управління транспортом (TMS) характеризується широким спектром рішень - від великих корпоративних платформ (SAP TM, Oracle TMS) до спеціалізованих хмарних

сервісів (Wialon, LogiNext). Проте більшість таких рішень є комерційними, мають закритий код і не дозволяють гнучко адаптувати функціонал під специфічні потреби малих та середніх підприємств.

Паралельно з розвитком TMS-рішень активно розвивається ринок low-code платформ автоматизації бізнес-процесів: Zapier, Make (Integromat), n8n. Ці платформи дозволяють з'єднувати різні сервіси через візуальний інтерфейс без написання складного коду, що суттєво скорочує час впровадження автоматизованих сповіщень та інтеграцій. Використання n8n дозволяє реалізувати обробку webhook-подій та інтеграцію із зовнішніми сервісами без розробки окремих модулів автоматизації.

1.2 Аналіз існуючих програмних рішень

Було проведено аналіз існуючих програмних систем для управління логістичними та диспетчерськими процесами. Розглянуто чотири основні рішення, що широко використовуються на ринку.

1С Логістика: Управління перевезеннями — вітчизняна система на платформі 1С, орієнтована на автоматизацію транспортної логістики підприємств. Забезпечує планування маршрутів, оформлення транспортних документів, розрахунок вартості перевезень. Серед недоліків — висока вартість впровадження та обслуговування, залежність від платформи 1С, відсутність відкритого API для зовнішніх інтеграцій.

Wialon — хмарна GPS-платформа для моніторингу транспорту. Платформа надає розширений функціонал геолокації та аналітики, однак орієнтована насамперед на моніторинг місцезнаходження, а не на управління замовленнями та диспетчерський документообіг.

SAP Transportation Management (SAP TM) — корпоративна система планування та управління транспортом у складі SAP ERP. Забезпечує наскрізну автоматизацію логістики на рівні великих підприємств. Основним

обмеженням є висока складність впровадження та необхідність значних інвестицій, що робить систему недоступною для малого та середнього бізнесу.

LogiNext Mile — хмарна SaaS-платформа для управління доставкою. Використовує алгоритми штучного інтелекту для оптимізації маршрутів. Основні недоліки — закритий код, відсутність можливості розгортання на власній інфраструктурі, висока вартість підписки.

В таблиці 1.1 наведено порівняльний аналіз розглянутих систем із розробленою в рамках цієї роботи.

Таблиця 1.1 — Порівняльний аналіз програмних рішень для диспетчеризації

Система	Платформа	Рольова модель	Автоматизація	Відкритий код
1С Логістика	Desktop/Web	Так	Часткова	Ні
Wialon	Web/SaaS	Так	GPS-моніторинг	Ні
SAP TM	Enterprise	Розширена	Так	Ні
LogiNext Mile	SaaS	Так	AI-маршрути	Ні
Розроблена система	Web (Spring Boot)	ADMIN DISPATCHER DRIVER	n8n сервіси	Так (GitHub)

Аналіз показує, що жодна з розглянутих систем не поєднує відкритий код, рольову модель доступу, нативну підтримку webhook-автоматизації через відкриту low-code платформу та можливість розгортання на власній інфраструктурі. Це обґрунтовує доцільність розробки власного рішення.

1.3 Аналіз вимог до системи

1) Функціональні вимоги. На підставі аналізу предметної галузі та потреб потенційних користувачів визначено такі функціональні вимоги до системи:

– Управління користувачами: реєстрація нових користувачів адміністратором, призначення ролей (ADMIN, DISPATCHER, DRIVER), вмикання та вимикання облікових записів.

– Автентифікація та авторизація: вхід в систему за email та паролем, видача JWT-токенів доступу та refresh-токенів, rotation refresh-токенів при кожному оновленні сесії, вихід із системи з анулюванням токенів.

– Управління водіями: реєстрація профілю водія, прив'язка до облікового запису, управління статусом (AVAILABLE, BUSY, OFFLINE, RESERVED), відстеження поточного місцезнаходження.

– Управління замовленнями: створення замовлення диспетчером, призначення водія, прийняття/відхилення замовлення водієм, виконання доставки, завершення або скасування замовлення.

– Автоматизація сповіщень: надсилання webhook-подій при настанні чотирьох типів бізнес-подій (призначення водія, завершення, скасування замовлення, зміна статусу водія) та їх обробка платформою n8n з відправкою сповіщень у Telegram та по електронній пошті.

– Аудит та журналювання: ведення журналу бізнес-дій користувачів (user_logs), журналу безпекових подій (security_logs: LOGIN, LOGOUT, REFRESH_TOKEN, UNAUTHORIZED, ACCESS_DENIED).

– Фільтрація та пагінація: сортування та постачання даних у вигляді сторінок для всіх спискових ресурсів (замовлення, водії, користувачі).

– Клієнтський інтерфейс: веб-інтерфейс для диспетчерів та окрема панель для водіїв.

Наведений перелік функціональних вимог охоплює повний життєвий цикл замовлення — від його створення диспетчером до завершення доставки

водієм — і слугує основою для подальшого проєктування архітектури системи.

2) Нефункціональні вимоги:

– Безпека: хешування паролів (BCrypt), JWT-підпис HMAC-SHA256, httpOnly cookie для refresh-токену, захист від несанкціонованого доступу через Spring Security [2].

– Масштабованість: модульна архітектура, можливість горизонтального масштабування серверного застосунку, пул потоків для асинхронних задач [3].

– Надійність: використання транзакцій бази даних, pessimistic locking для запобігання гонкам даних при призначенні водія, Flyway-міграції для версіонування схеми.

– Розгортання: контейнеризація Docker, конфігурація через змінні середовища, підтримка docker-compose.

– Супровідність: документація API через OpenAPI/Swagger, версіонування коду у репозиторії GitHub [4].

– Продуктивність: асинхронна обробка подій аудиту та webhook-сповіщень, індексування ключових полів у БД [5], пагінація результатів запитів.

Дотримання зазначених нефункціональних вимог забезпечує стабільну роботу системи під навантаженням та зручність її подальшого супроводу.

3) Рольова модель доступу. Система реалізує рольову модель доступу (RBAC — Role-Based Access Control). Визначено три ролі користувачів, що відповідають реальним посадам у транспортній компанії. В таблиці 1.2 наведено опис ролей та перелік доступних дій.

Розмежування доступу реалізовано за допомогою анотацій @PreAuthorize на рівні контролерів через механізми Spring Security.

Такий підхід реалізує принцип найменших привілеїв, за якого користувач отримує лише той набір прав, що необхідний для виконання його посадових обов’язків.

Таблиця 1.2 — Рольова модель доступу системи

Роль	Опис	Основні дії
ADMIN	Адміністратор системи	Управління користувачами, водіями, замовленнями, перегляд аудит-журналів
DISPATCHER	Диспетчер	Створення замовлень, призначення водіїв, зміна статусів
DRIVER	Водій, що виконує доставки	Перегляд поточного замовлення, прийняття / відхилення / завершення

1.4 Вибір та обґрунтування технологічного стеку

1) Вибір мови програмування та фреймворку. Для розробки серверної частини розглянуто три альтернативи: Spring Boot (Java) [6], Django (Python) та Express.js (Node.js). Порівняння наведено у таблиці 1.3.

Таблиця 1.3 — Порівняння серверних технологій

Критерій	Spring Boot (Java)	Django (Python)	Node.js (Express)
Типізація	Строга (Java)	Динамічна	Динамічна / TypeScript
ORM	Hibernate / JPA	Django ORM	Sequelize / Prisma
Безпека	Spring Security	django-allauth	Passport.js
Масштабованість	Висока	Середня	Висока
Обрано	Так	Ні	Ні

Обрано Java 26 + Spring Boot 4 [7, 8, 9]. Строга статична типізація Java дозволяє виявляти помилки на етапі компіляції; Spring Security забезпечує

потужний інструментарій для реалізації автентифікації та авторизації [10]; Spring Data JPA спрощує роботу з реляційними базами даних. Обрана зв'язка є зрілим і широко підтримуваним рішенням з великою спільнотою та готовою екосистемою бібліотек, що знижує ризики під час розробки та подальшого супроводу.

2) Вибір системи управління базами даних. Для зберігання даних розглядалися три DBMS: PostgreSQL, MySQL та MongoDB. Порівняння наведено у таблиці 1.4.

Таблиця 1.4 — Порівняння DBMS

Критерій	PostgreSQL	MySQL	MongoDB
Модель	Реляційна	Реляційна	Документна
ENUM-типи	Нативна підтримка	Часткова	Відсутня
ACID-транзакції	Так	Так	Обмежено
Відкритий код	Так	Обмежено (Oracle)	Так
Обрано	Так	Ні	Ні

Обрано PostgreSQL [11]. Ключові переваги: нативна підтримка типів ENUM (user_role, driver_status, order_status), яку активно використовує система; повна підтримка ACID-транзакцій; відкритий код без обмежень ліцензування. Версіонування схеми реалізовано через Flyway. Поєднання надійності транзакційної моделі та гнучкості ENUM-типів робить PostgreSQL оптимальним вибором для предметної галузі вантажних перевезень.

3) Вибір платформи автоматизації. Для реалізації автоматизованих сповіщень при настанні бізнес-подій розглянуто три платформи. Порівняння наведено у таблиці 1.5.

Таблиця 1.5 — Порівняння платформ автоматизації

Критерій	n8n	Zapier	Make (Integromat)
Розгортання	Self-hosted / Cloud	SaaS	SaaS
Відкритий код	Так	Ні	Ні
Вартість (self-hosted)	Безкоштовно	Платно	Платно
Docker-інтеграція	Так	Ні	Ні
Обрано	Так	Ні	Ні

Обрано n8n. Платформа є відкритою та підтримує розгортання у власній інфраструктурі через Docker [12], що узгоджується з архітектурою системи. n8n надає вбудований webhook-вузол та готові інтеграції з Telegram і SMTP без написання коду, а на відміну від Zapier та Make, не потребує підписки при self-hosted розгортанні.

4) Інструменти контейнеризації та розгортання. Для розгортання обрано Docker та Docker Compose [13]. Docker ізолює кожен компонент системи (backend, frontend, PostgreSQL, n8n) у власному контейнері, забезпечуючи відтворюваність середовища, а Docker Compose спрощує оркестрацію та дозволяє запустити всі сервіси однією командою. Конфігурація реалізована через змінні середовища.

5) Клієнтська частина. Клієнтський інтерфейс розроблено з використанням React + TypeScript + Vite [14, 15]. React обрано за широку екосистему та компонентний підхід, TypeScript додає строгу типізацію, Vite — швидку збірку та гарячу заміну модулів. Для HTTP-запитів використовується Axios з інтерсепторами для автоматичного оновлення токенів, маршрутизація — через React Router v7.

РОЗДІЛ 2 ПРОЄКТНА ЧАСТИНА

2.1 Розробка узагальненої структури системи

Розроблена система є монолітним веб-застосунком з чітко визначеними шарами відповідальності [16]. Система складається з чотирьох основних компонентів, що взаємодіють між собою у межах єдиного середовища Docker Compose:

– Серверна частина (Backend) — Spring Boot 4 застосунок, що реалізує всю бізнес-логіку, обробку REST-запитів, автентифікацію, аудит та взаємодію з базою даних.

– Клієнтська частина (Frontend) — React 19 / TypeScript SPA, що надає графічний інтерфейс диспетчерам і водіям для роботи із системою.

– База даних (PostgreSQL) — реляційна DBMS, що зберігає всі дані системи. Версіонування схеми здійснюється через Flyway.

– Платформа автоматизації (n8n) — low-code середовище, що обробляє webhook-події від серверної частини та ініціює автоматизовані сповіщення у Telegram та по електронній пошті.

Серверна частина реалізує класичну тришарову архітектуру [17,18]: Controller layer приймає HTTP-запити, делегує виконання service-шару, повертає DTO-відповіді; Service layer містить бізнес-логіку; Repository layer — Spring Data JPA-репозиторії з підтримкою Specification API та pessimistic locking.

Наскрізно через усі шари працює подвійний AOP-механізм [19]: AuditAspect перехоплює методи, анотовані @AuditAction, та асинхронно зберігає бізнес-лог; WebhookAspect перехоплює методи, анотовані

@WebhookEvent, та асинхронно надсилає події до n8n. Окремий пул потоків (ThreadPoolTaskExecutor) обробляє аудит і webhook-виклики, не блокуючи основний потік запиту.

Зовнішня взаємодія між серверною частиною та n8n реалізована через єдиний HTTP POST webhook. При настанні бізнес-події WebhookAspect через WebhookService асинхронно надсилає структурований об'єкт WebhookEventDto (з полями eventType та data) на зареєстрований n8n endpoint. n8n маршрутизує подію через вузол Switch та виконує налаштовані дії.

2.2 Проєктування бази даних

База даних складається з шести основних таблиць, зв'язаних зовнішніми ключами. В таблиці 2.1 наведено перелік таблиць та їх призначення. UML діаграма бази даних наведена у додатку В.

Таблиця 2.1 — Таблиці бази даних системи

Таблиця	Призначення	Ключові поля
users	Облікові записи користувачів	id, email, password_hash, role, enabled
drivers	Профілі водіїв	id, name, status, current_location, user_id
orders	Замовлення на доставку	id, pickup_location, delivery_location, status, driver_id, created_by
refresh_tokens	Refresh-токени сесій	id, token, user_id, expiry_date, revoked
user_logs	Журнал бізнес-дій	id, user_id, email, action, entity, entity_id, ip_address, http_method
security_logs	Журнал безпекових подій	id, user_id, email, action, success, ip_address, user_agent

Схема даних включає три власних ENUM-типи PostgreSQL, що описані у таблиці 2.2. Використання нативних ENUM-типів забезпечує цілісність даних на рівні DBMS та дозволяє ORM-фреймворку Hibernate автоматично виконувати маппінг між Java-перерахуваннями та колонками таблиць [20].

Таблиця 2.2 — ENUM-типи PostgreSQL

ENUM-тип	Таблиця	Значення
user_role	users	ADMIN, DISPATCHER, DRIVER
driver_status	drivers	AVAILABLE, BUSY, OFFLINE, RESERVED
order_status	orders	CREATED, ASSIGNED, IN_PROGRESS, COMPLETED, CANCELLED

Між таблицями визначено такі зв’язки: drivers → users (ManyToOne, UNIQUE — один користувач має не більше одного профілю водія); orders → drivers (ManyToOne, ON DELETE SET NULL); orders → users (ManyToOne — автор замовлення); refresh_tokens → users (ManyToOne, ON DELETE CASCADE).

Версіонування схеми реалізовано через Flyway. Кожна міграція виконується рівно один раз у визначеному порядку [21].

Для прискорення запитів у ключових таблицях створено індекси: idx_user_logs_user_id, idx_user_logs_action, idx_user_logs_created_at (таблиця user_logs); idx_security_logs_user_id, idx_security_logs_email, idx_security_logs_created_at (таблиця security_logs); idx_refresh_tokens_user_id (таблиця refresh_tokens).

2.3 Проєктування серверної частини

1) Структура пакетів. Серверна частина організована за принципом package-by-feature. Кожен домен (user, driver, order, security) містить власні підпакети: controller, service, repository, entity, dto, mapper, specification. Спільна інфраструктура розміщена у пакеті common (audit, exception, util), конфігурація — у пакеті config (AppConfig, AsyncConfig, OpenApiConfig, WebConfig, WebhookProperties), webhook-компоненти — у пакеті n8n (annotation, aspect, dto, entity, service). Такий поділ забезпечує слабку зв'язаність між доменами та полегшує масштабування й супровід коду.

2) Специфікація REST API. Система надає REST-ендпоінти, згруповані за доменами [22]. Повна специфікація наведена у таблиці 2.3. Контроль доступу реалізовано через анотації @PreAuthorize Spring Security.

Таблиця 2.3 — Специфікація REST API системи

Метод	Endpoint	Роль	Опис
POST	/auth/login	—	Вхід, видача JWT та refresh-токену
POST	/auth/refresh	—	Оновлення токену (rotation)
POST	/auth/logout	—	Вихід, анулювання токенів
POST	/users	ADMIN	Створення нового користувача
GET	/users	ADMIN	Список користувачів
GET	/users/me	AUTH	Дані поточного користувача
GET	/users/{id}	ADMIN	Користувач за ID

Продовж. табл. 2.3

Метод	Endpoint	Роль	Опис
GET	/users/search	ADMIN	Пошук користувача за email
PUT	/users/me/password	AUTH	Зміна пароля
POST	/drivers	ADMIN	Реєстрація профілю водія
GET	/drivers	ADMIN/DISP	Список водіїв
GET	/drivers/{id}	ADMIN/DISP	Водій за ID
GET	/drivers/search/name	ADMIN/DISP	Пошук водіїв за іменем
GET	/drivers/search/email	ADMIN/DISP	Пошук водіїв за email
PUT	/drivers/{id}/status	ADMIN	Оновлення статусу водія
PUT	/drivers/{id}/location	ADMIN\ DRIVER	Оновлення місцезнаходження
POST	/orders	ADMIN/DISP	Створення замовлення
GET	/orders	ADMIN/DISP	Список замовлень
GET	/orders/{id}	ADMIN/DISP	Замовлення за ID
PUT	/orders/{id}/assign	ADMIN/DISP	Призначення водія до замовлення
PUT	/orders/{id}/accept	DRIVER	Прийняття замовлення
PUT	/orders/{id}/reject	DRIVER	Відхилення замовлення
PUT	/orders/{id}/complete	DRIVER	Завершення замовлення
PUT	/orders/{id}/cancel	ADMIN/DISP	Скасування замовлення
GET	/orders/my-current	DRIVER	Поточне замовлення
GET	/audit/user-logs	ADMIN	Журнал бізнес-дій
GET	/audit/security-logs	ADMIN	Журнал безпекових подій

Централізована обробка помилок реалізована через GlobalExceptionHandler (@RestControllerAdvice): 400 Bad Request (InvalidPasswordException, InvalidPrincipalException), 401 Unauthorized (UserNotAuthenticatedException), 403 Forbidden (InvalidUserRoleException), 404 Not Found (UserNotFoundException, DriverNotFoundException, OrderNotFoundException), 409 Conflict (DriverNotAvailableException, EmailAlreadyExistsException).

3) Модель безпеки: JWT та Refresh Token Rotation. Автентифікація реалізована на основі двох токенів: короткоживучого JWT access-токену та довгоживучого refresh-токену з ротацією [23]. Процес функціонування:

– Користувач надсилає POST /auth/login з email та паролем.

– AuthService верифікує пароль через BCryptPasswordEncoder.matches().

– JwtService.generateToken() підписує JWT (HMAC-SHA256) з claims: subject (email), roles та expiration. RefreshTokenService.create() зберігає новий refresh-токен у БД.

– Клієнт отримує accessToken у тілі відповіді та refreshToken у httpOnly cookie (path=/auth).

– Для захищених запитів клієнт додає заголовок Authorization: Bearer <accessToken>.

– JwtAuthFilter перехоплює кожен запит, валідує підпис та expiration, завантажує User з БД та встановлює SecurityContext.

– При закінченні терміну дії accessToken клієнт надсилає POST /auth/refresh. RefreshTokenService.rotate() атомарно анулює старий токен та видає новий — забезпечується механізм token reuse detection.

2.4 Проєктування системи автоматизації (n8n)

Платформа n8n інтегрується з серверною частиною через єдиний уніфікований webhook-endpoint, що обробляє всі типи бізнес-подій системи.

Архітектурний принцип інтеграції — слабка зв’язність (loose coupling): серверна частина надсилає HTTP POST запит і не очікує результату. Асинхронний виклик через @Async не блокує відповідь клієнту.

Механізм ініціювання webhook-подій реалізований через AOP-аспект WebhookAspect. Анотація @WebhookEvent застосовується до методів контролерів та сервісів. Після успішного виконання методу WebhookAspect автоматично викликає WebhookService.send(), передаючи тип події та результат операції як payload. Такий підхід повністю відокремлює логіку автоматизації від бізнес-коду.

У системі визначено типи webhook-подій перерахування WebhookEventType, наведені у таблиці 2.4.

Таблиця 2.4 — Типи webhook-подій системи

Тип події (WebhookEventType)	Метод-ініціатор	Опис
ORDER_ASSIGNED	OrderController .assignDriver()	Призначення водія до замовлення
ORDER_COMPLETED	OrderController .completeOrder()	Завершення замовлення водієм
ORDER_CANCELLED	OrderController .cancelOrder()	Скасування замовлення диспетчером
DRIVER_STATUS_ CHANGED	DriverService .changeDriverStatus()	Зміна статусу водія
DRIVER_LOCATION_ NEAR_DESTINATION	DriverService .updateDriverCurrentLocation() (умовно, прямий виклик)	Наближення водія до пункту призначення (< 2 км)

Перші чотири події генеруються декларативно через анотацію @WebhookEvent, п’ята (DRIVER_LOCATION_NEAR_DESTINATION) — умовно, прямим викликом WebhookService.send() у логіці перевірки відстані.

WebhookService формує об’єкт WebhookEventDto, що містить поля eventType (тип події WebhookEventType) та data (payload результату операції). Об’єкт серіалізується у JSON та надсилається через RestTemplate.postForEntity() на URL, визначений у WebhookProperties.getLogisticsEventUrl() (властивість app.webhooks.logistics-event-url, змінна середовища WEBHOOK_LOGISTICS_EVENT_URL). При недоступності n8n виключення перехоплюється та записується попередження, основний потік виконання не переривається.

Workflow у n8n (файл n8n/workflows/logistics-event.json) побудований за схемою маршрутизації подій. Опис вузлів наведено у таблиці 2.5.

Таблиця 2.5 — Вузли n8n workflow logistics-event

Вузол	Тип	Функція
Webhook logistics-event	Webhook (POST)	Приймає вхідний HTTP POST від серверної частини
Switch	Switch	Маршрутизує подію
ORDER_ASSIGNED	Set	Повідомлення про призначення
ORDER_COMPLETED	Set	Повідомлення про виконання
ORDER_CANCELLED	Set	Повідомлення про скасування
DRIVER_STATUS_CHANGED	Set	Повідомлення про зміну статусу водія
DRIVER_LOCATION_NEAR_ DESTINATION	Code + Set	Дедуплікація повідомлення про наближення
Send a text message	Telegram	Telegram повідомлення
Send an Email	emailSend SMTP	Надсилає повідомлення електронною поштою

Кожна з п'яти гілок Switch з'єднана паралельно з вузлами Send a text message (Telegram) та Send an Email.

Гілка DRIVER_LOCATION_NEAR_DESTINATION додатково містить Code-вузол дедуплікації, розміщений перед вузлами сповіщення: оскільки ця подія надходить повторно, поки водій перебуває в межах порогової зони, вузол

пропускає лише перше повідомлення для кожного замовлення. Workflow версіонується у репозиторії Git та монтується до контейнера n8n через Docker Volume (bind mount ./n8n/workflows:/workflows). Налаштування credentials Telegram Bot API та SMTP здійснюється через захищене сховище credentials n8n.

2.5 Проєктування системи аудиту

Система реалізує двошарову архітектуру аудиту. Перший шар — бізнес-аудит (user_logs) — журналює дії користувачів у предметній галузі через AOP-аспект AuditAspect (@Around), що перехоплює методи, анотовані @AuditAction(UserAction.VALUE). Другий шар — безпековий аудит (security_logs) — журналює автентифікаційні та авторизаційні події: LOGIN, LOGOUT, REFRESH_TOKEN, UNAUTHORIZED, ACCESS_DENIED. Обидва шари працюють асинхронно, тому журналювання не впливає на час відгуку основних бізнес-операцій.

AuditAspect при перехопленні: визначає IP, User-Agent, URI та HTTP-метод з HttpServletRequest; читає email та userId через SecurityUtils; при успіху зберігає лог зі значенням SUCCESS, при виключенні — з повідомленням помилки; якщо результат реалізує інтерфейс Identifiable, витягує entityId для лінкування з конкретним об’єктом.

Такий підхід на основі AOP дозволяє централізувати логіку аудиту в одному аспекті та уникнути дублювання коду журналювання в бізнес-методах: для додавання нової контрольованої дії достатньо анотувати метод @AuditAction із відповідним значенням UserAction. Розділення на два шари (user_logs та security_logs) забезпечує чітке відмежування подій предметної галузі від подій безпеки, що спрощує подальший аналіз та фільтрацію журналів. В таблиці 2.6 наведено перелік значень перерахування UserAction.

Таблиця 2.6 — Значення UserAction та їх семантика

Значення UserAction	Опис події
CREATE_ORDER / UPDATE_ORDER / DELETE_ORDER	Операції зі створення, редагування та видалення замовлень
ASSIGN_DRIVER / ACCEPT_ORDER / REJECT_ORDER	Процес призначення та прийняття/відхилення замовлень
COMPLETE_ORDER / CANCEL_ORDER	Завершення або скасування замовлення
CREATE_USER / UPDATE_USER / DELETE_USER	Управління обліковими записами
UPDATE_USER_PASSWORD	Зміна пароля користувача
CREATE_DRIVER / UPDATE_DRIVER / DELETE_DRIVER	Управління профілями водіїв
UPDATE_DRIVER_STATUS / UPDATE_DRIVER_LOCATION	Зміна стану та місцезнаходження водія

2.6 Проєктування клієнтського інтерфейсу

Клієнтська частина реалізована як SPA на основі React 19 з TypeScript та React Router v7. Архітектура побудована навколо трьох концепцій: AuthProvider для управління автентифікацією, RoleGuard для захисту маршрутів та API-клієнта Axios з автоматичним оновленням токенів.

Схема маршрутизації застосунку: /login — публічна сторінка входу; / (MainLayout) — захищена зона для ADMIN та DISPATCHER: Dashboard, /orders, /orders/create, /users, /drivers; /driver (DriverLayout) — захищена зона для DRIVER: панель водія (DriverDashboard).

AuthProvider при ініціалізації виконує POST /auth/refresh. При успіху записує accessToken у пам’ять (inMemoryAccessToken) та завантажує GET /users/me. Зберігання токену виключно в пам’яті унеможливлює його зчитування через XSS-атаки. Після успішного оновлення токену всі відкладені у failedQueue запити повторюються автоматично з новим accessToken, що робить процес прозорим для користувача та не перериває його роботу. Компонент RoleGuard перевіряє роль із завантаженого профілю /users/me і за невідповідності перенаправляє користувача на дозволену для його ролі зону, унеможливлюючи доступ до чужих маршрутів навіть за прямого введення URL. Така архітектура чітко відокремлює логіку автентифікації та авторизації від презентаційного шару, завдяки чому проєктування візуальних компонентів зводиться до композиції захищених маршрутів і не дублює перевірки безпеки.Axios-інтерсептор відповідей перехоплює HTTP 401, виконує token refresh та повторює запит. Паралельні запити ставляться у чергу (failedQueue).

2.7 Проєктування апаратного модуля моніторингу (GPS-трекер)

1) Призначення та структура модуля. Для забезпечення збору геоданих про фактичне місцезнаходження транспортних засобів до складу системи введено апаратний модуль — GPS-трекер на базі мікроконтролера ESP32 [24]. Модуль виконує роль автономного клієнта серверної частини: періодично визначає координати засобами супутникової навігації та надсилає їх до backend через наявний REST-ендпоінт PUT /drivers/{id}/location, використовуючи Bearer-автентифікацію. Такий підхід не потребує змін у моделі безпеки: трекер є звичайним HTTP-клієнтом і взаємодіє з API на тих самих умовах, що й веб-інтерфейс. Завдяки автономності модуля його можна встановити на будь-який транспортний засіб без доопрацювання серверної частини системи.

Узагальнену структуру системи з урахуванням апаратного рівня наведено на рис. 2.1 (структурна схема С1). Апаратний модуль об’єднує мікроконтролер ESP32, GPS-приймач NEO-6M, OLED-дисплей, світлодіодну індикацію та кнопку керування і взаємодіє із серверною частиною (Spring Boot, PostgreSQL, n8n) поверх мережі WiFi.

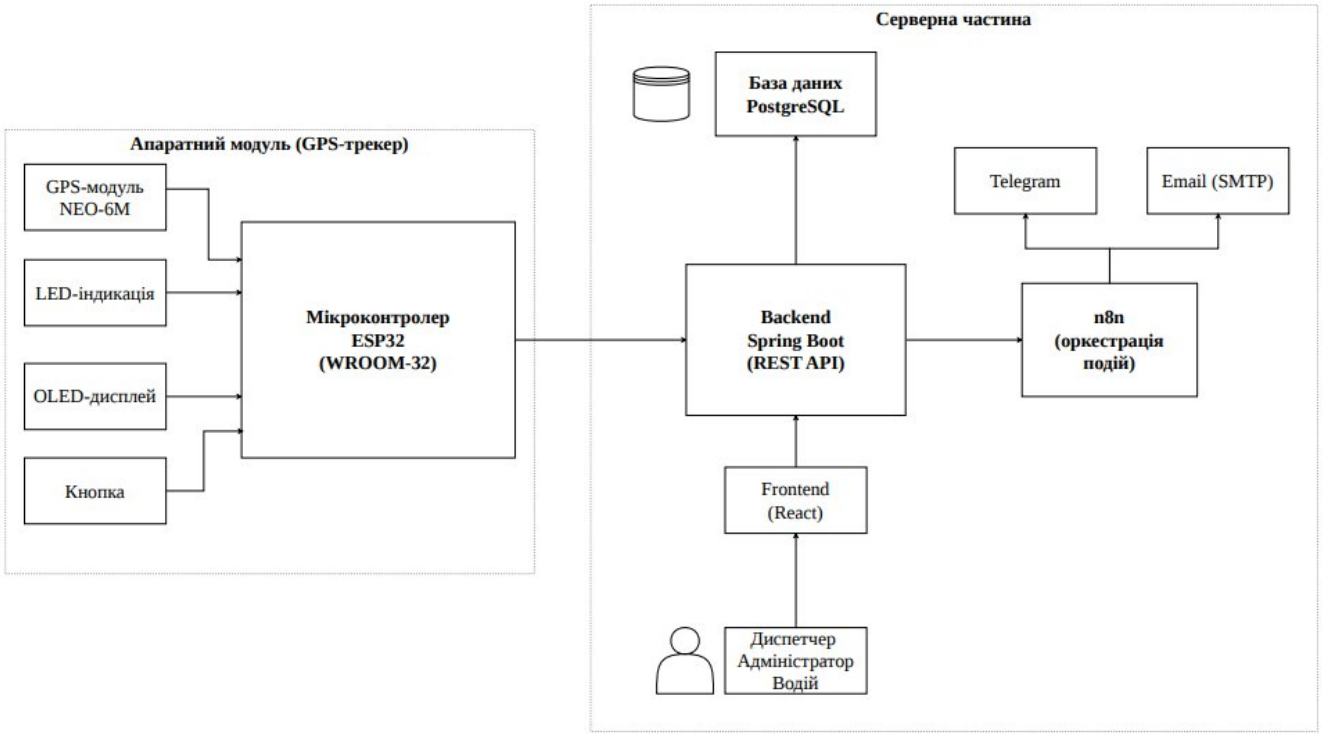

Рисунок 2.1 — Структурна схема системи з апаратним модулем (С1)

Функціональну організацію трекера та потоки даних між його блоками показано на рис. 2.2 (функціональна схема Е2). Приймач GPS передає координати до мікроконтролера по інтерфейсу UART у форматі NMEA 0183; блок індикації (OLED + LED) отримує дані статусу по I²C та GPIO; блок зв’язку (WiFi) формує HTTP-запит до сервера; блок живлення забезпечує напруги 5 В та 3,3 В.

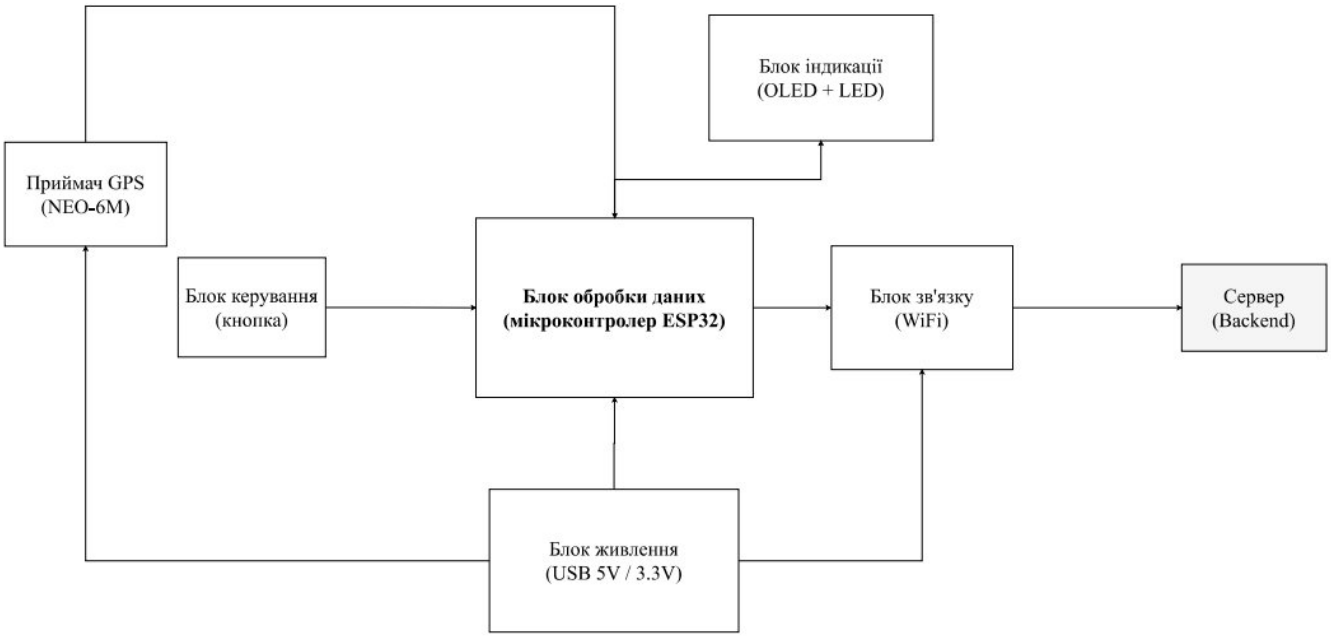

Рисунок 2.2 — Функціональна схема апаратного модуля (Е2)

2) Вибір елементної бази. Елементну базу модуля підібрано за критеріями доступності, низької вартості та достатньої функціональності для демонстраційного зразка. Перелік елементів наведено у таблиці 2.7. та у додатку Б.

Таблиця 2.7 — Елементна база апаратного модуля

Компонент	Модель	Інтерфейс	Призначення
Мікроконтролер	ESP32-WROOM-32 (DevKit V1)	—	Обробка даних, WiFi-зв’язок
GPS-приймач	NEO-6M (GY-GPS6MV2)	UART NMEA 0183	Визначення координат
Дисплей	OLED 0,91" (SSD1306)	I²C	Відображення статусу
Світлодіод	LED + резистор R220	GPIO	Індикація стану зв’язку
Кнопка	Тактова SW_Push	GPIO	Ручне керування
Живлення	USB 5 В / LDO 3,3 В	—	Електроживлення

Вибір ESP32 зумовлений вбудованим модулем WiFi, достатньою обчислювальною потужністю та підтримкою енергонезалежної пам’яті (NVS) для збереження токена автентифікації між циклами живлення. Приймач NEO-6M є поширеним недорогим рішенням із підтримкою стандарту NMEA 0183. Періодичність опитування (полінг) обрано рівною одній хвилині — це простий і надійний патерн для трекера, що забезпечує прийнятну точність відстеження за мінімального мережевого навантаження. За потреби інтервал опитування можна змінити програмно без втручання в апаратну частину, що дозволяє гнучко балансувати між точністю відстеження та споживанням трафіку.

3) Принципова схема. Принципову електричну схему модуля наведено на рис. 2.3 (Е3). Мікроконтролер ESP32-WROOM-32 (U1) з’єднано з GPS-приймачем NEO-6M (J1) по апаратному UART2; для узгодження логічних рівнів на лінії використано резистивний дільник R1k/R2k. OLED-дисплей (U2) підключено по шині I²C (лінії SDA/SCL). Світлодіод D1 під’єднано через струмообмежувальний резистор R220, кнопка SW1 — між виводом GPIO та спільним проводом. Розподіл виводів мікроконтролера наведено у таблиці 2.8. Живлення компонентів організовано від двох напруг: мікроконтролер ESP32-WROOM-32 та OLED-дисплей живляться напругою +3,3 В, а GPS-приймач NEO-6M — напругою +5 В, при спільному проводі GND. Оскільки логічні рівні NEO-6M (5 В) перевищують допустимі для входів ESP32 (3,3 В), резистивний дільник R1k/R2k увімкнено саме на лінії передачі даних від приймача до мікроконтролера, що знижує рівень сигналу до безпечного. Світлодіодна та кнопкова частини під'єднані до спільного проводу GND, утворюючи замкнені кола індикації та керування.

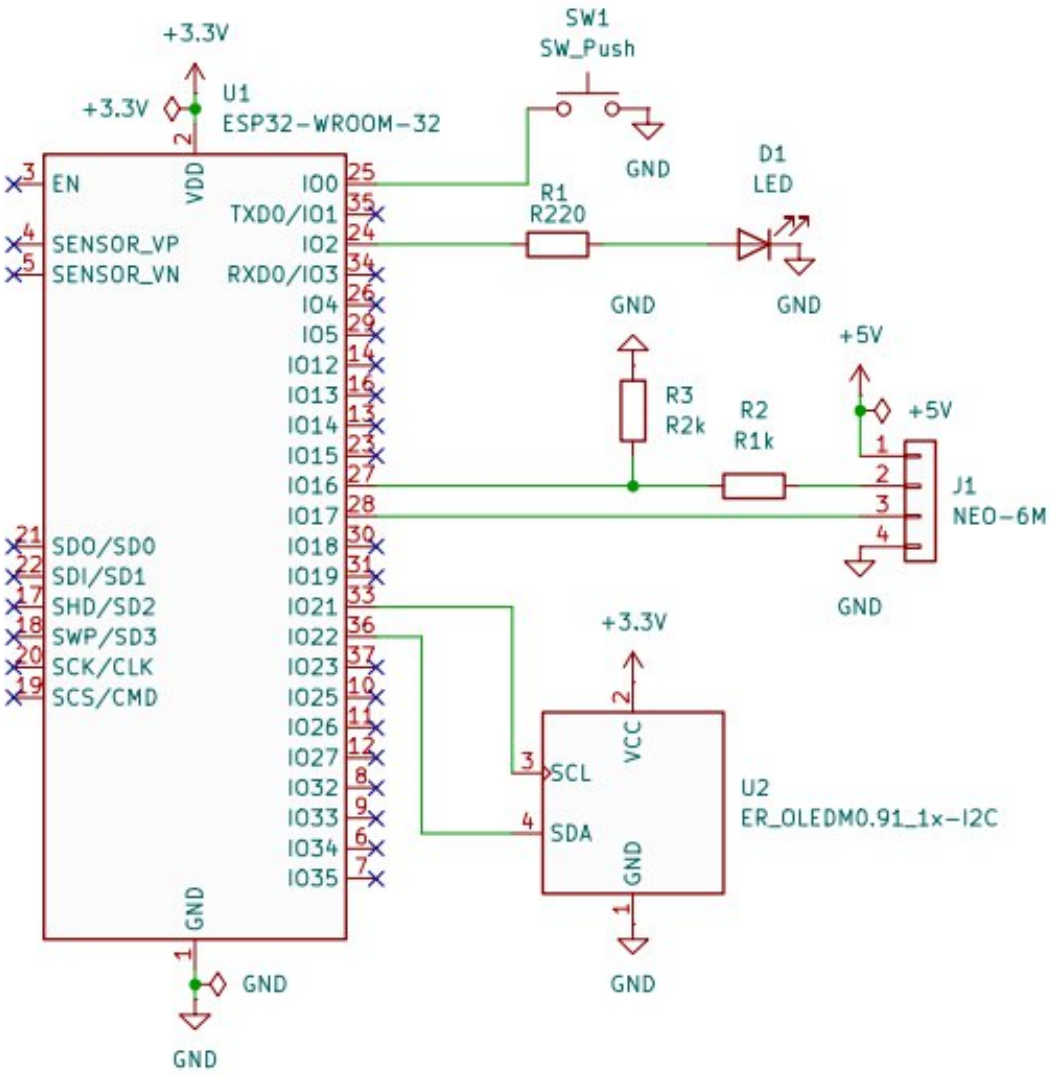

Рисунок 2.3 — Принципова електрична схема апаратного модуля (E3)

Таблиця 2.8 — Розподіл виводів мікроконтролера ESP32

Вивід ESP32	Підключення	Призначення
GPIO16 (RX2)	NEO-6M TX (через дільник R1k/R2k)	Прийом NMEA-даних GPS
GPIO17 (TX2)	NEO-6M RX	Передача команд GPS
GPIO21 (SDA)	OLED SDA	Лінія даних I²C
GPIO22 (SCL)	OLED SCL	Лінія тактування I²C
GPIO2	LED (через R220)	Індикація стану
GPIO0	Кнопка → GND	Ручне керування
3V3 / 5V / GND	Живлення NEO-6M, OLED	Електроживлення

4) Макет системи. Макет системи з'єднань компонентів модуля наведено на рис. 2.4. Він відображає фактичне розташування плати ESP32 DevKit, GPS-модуля GY-GPS6MV2 з антеною, OLED-дисплея та світлодіода, а також трасування з'єднувальних провідників відповідно до принципової схеми.

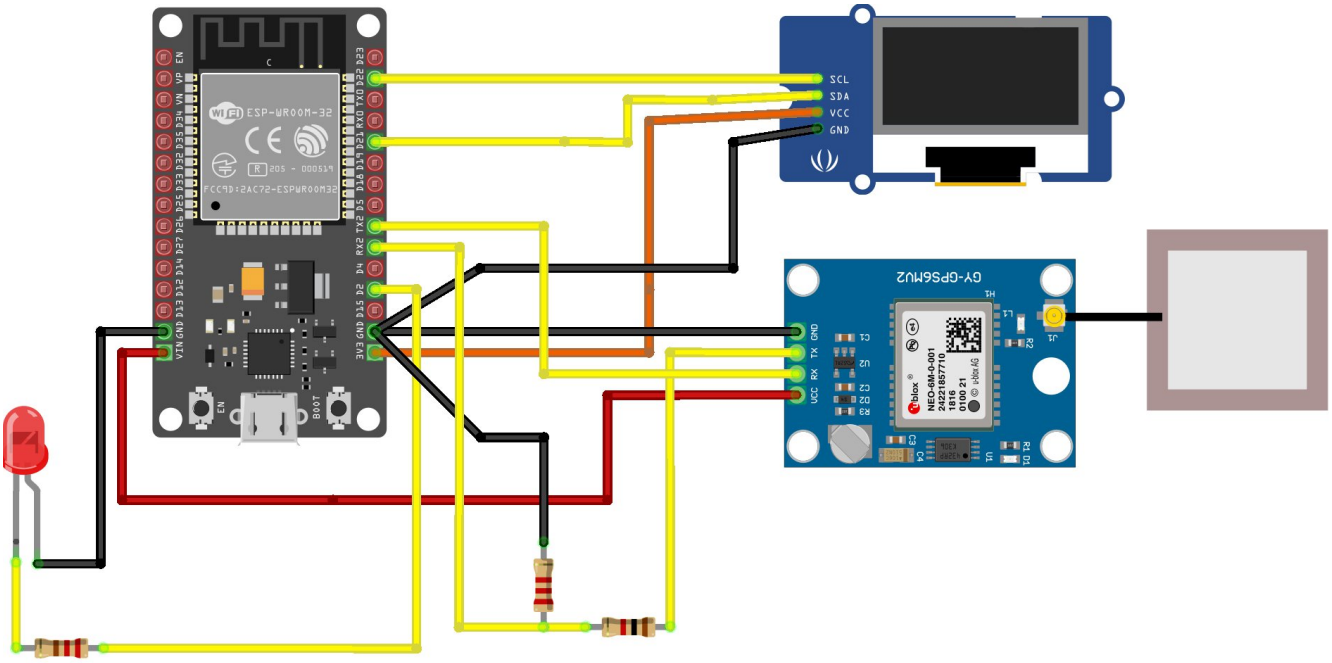

Рисунок 2.4 — Макет системи

5) Подія наближення до пункту призначення. Апаратний модуль розширює перелік бізнес-подій системи подією DRIVER_LOCATION_NEAR_DESTINATION — «водій наблизився до пункту призначення замовлення». Ця подія є умовною: вона генерується не на кожен виклик методу оновлення локації, а лише за виконання просторової умови. Тому її реалізовано не через AOP-анотацію @WebhookEvent (яка спрацьовує на кожен виклик анотованого методу), а через прямий виклик WebhookService.send() усередині логіки перевірки відстані. Це дозволяє зберегти декларативність аспектного підходу для безумовних подій і водночас

коректно обробити умовний сценарій. Такий гібридний підхід поєднує переваги обох механізмів: безумовні події залишаються повністю декларативними, а умовна подія отримує контроль над моментом свого генерування. Це уникає зайвих викликів webhook та навантаження на оркестратор у випадках, коли просторова умова не виконується.

Перевірку відстані виконує серверна частина, а не трекер. Таке рішення обґрунтоване тим, що апаратний модуль є «тонким» сенсором і не зберігає інформацію про замовлення та його пункт призначення — цими даними володіє backend. При надходженні нових координат серверна частина визначає активне замовлення водія (статус IN_PROGRESS), обчислює відстань між поточним місцезнаходженням і координатами пункту призначення за формулою гаверсинусів і, якщо відстань менша за поріг 2 км, генерує подію. Оскільки водій може перебувати в межах порогової зони протягом кількох циклів опитування, така подія надсилатиметься повторно. Запобігання дублюванню сповіщень винесено на рівень оркестратора n8n: у відповідній гілці workflow перед вузлами сповіщення розміщено вузол дедуплікації, що за допомогою власного сховища стану (getWorkflowStaticData) пропускає лише перше повідомлення для кожного замовлення, а наступні відхиляє. Винесення дедуплікації на рівень n8n, а не серверної частини, дозволяє не ускладнювати бізнес-логіку backend та зберігати стан безпосередньо в межах workflow. Завдяки цьому кожне замовлення породжує рівно одне сповіщення про наближення незалежно від кількості циклів опитування, проведених водієм у пороговій зоні.

РОЗДІЛ 3 ПРАКТИЧНА ЧАСТИНА

3.1 Реалізація серверної частини

1) Модуль автентифікації та авторизації. Центральним компонентом безпеки є JwtAuthFilter — фільтр, що успадковує OncePerRequestFilter і виконується для кожного HTTP-запиту рівно один раз. Фільтр витягує Bearer-токен з заголовку Authorization, перевіряє його дійсність через JwtService.isTokenValid() та, у разі успіху, завантажує користувача з БД і встановлює UsernamePasswordAuthenticationToken у SecurityContextHolder. Виключення JwtAuthenticationException перехоплюються і логуються без переривання ланцюжка фільтрів.

JwtService реалізує підписання та верифікацію JWT за алгоритмом HMAC-SHA256. Токен містить claims: subject (email), roles та expiration. Секретний ключ зчитується з змінної середовища JWT_SECRET_KEY. RefreshTokenService управляє life cycle refresh-токенів: create() генерує UUID, validate() перевіряє існування та термін дії, rotate() атомарно анулює поточний токен і видає новий, revokeAllByUser() при logout інвалідує всі активні сесії.

Очищення застарілих токенів виконується щодобово о 03:00 через RefreshTokenCleanupService (@Scheduled(cron = "0 0 3 * * *")). SecurityConfig налаштовує SecurityFilterChain: вимикає CSRF, налаштовує CORS, дозволяє публічний доступ до /auth/** та /swagger-ui/**; SessionCreationPolicy.STATELESS забезпечує відсутність HTTP-сесій. Фрагмент фільтру JwtAuthFilter наведено на рис. 3.1.

```
Optional<String> jwt = jwtAuthService.extractToken(request);
if (jwt.isPresent() && jwtService.isTokenValid(jwt.get())) {
    String email = jwtService.extractEmail(jwt.get());
    if (email != null && SecurityContextHolder.getContext()
                                                .getAuthenticatio
    n() == null) {
        jwtAuthService.authenticateUser(request, email);
    }
}
```

Рисунок 3.1 — Лістинг фрагмента фільтра JwtAuthFilter

2) Модуль управління замовленнями. OrderService реалізує повний life cycle замовлення. В таблиці 3.1 наведено матрицю допустимих переходів між статусами.

Таблиця 3.1 — Матриця переходів статусів замовлення

Поточний статус	Дія	Виконавець	Наступний статус
CREATED	assignDriver()	ADMIN /DISPATCHER	ASSIGNED
ASSIGNED	acceptOrder()	DRIVER	IN_PROGRESS
ASSIGNED	rejectOrder()	DRIVER	CREATED
IN_PROGRESS	completeOrder()	DRIVER	COMPLETED
CREATED / ASSIGNED / IN_PROGRESS	cancelOrder()	ADMIN /DISPATCHER	CANCELLED

Метод assignDriver() є найбільш критичним з точки зору конкурентності. Він послідовно набуває pessimistic lock на Order та на Driver через findByIdForUpdate() (@Lock(PESSIMISTIC_WRITE)). Лише після перевірки order.getStatus() == CREATED та driver.getStatus() == AVAILABLE виконується оновлення стану обох сутностей в межах єдиної @Transactional транзакції. Фрагмент методу наведено на рис. 3.2.

```
@Transactional
public OrderResponseDto assignDriver(Long orderId,
    AssignDriverRequestDto dto) {
    Order order = orderRepository.findByIdForUpdate(orderId)
        .orElseThrow(() -> new OrderNotFoundException("Order
    not found"));
    if (order.getStatus() != OrderStatus.CREATED)
        throw new IllegalStateException("Invalid order state");
    Driver driver =
    driverRepository.findByIdForUpdate(dto.getDriverId())
        .orElseThrow(() -> new DriverNotFoundException("Driver
    not found"));
    if (driver.getStatus() != DriverStatus.AVAILABLE)
        throw new DriverNotAvailableException("Driver not
    available");
    order.setDriver(driver);
    order.setStatus(OrderStatus.ASSIGNED);
    driver.setStatus(DriverStatus.RESERVED);
    return orderMapper.toDto(orderRepository.save(order));
}
```

Рисунок 3.2 — Лістинг фрагмента методу OrderService.assignDriver()

Фільтрація замовлень реалізована через OrderSpecification (JPA Specification API) з підтримкою фільтрів: status (рівність по ENUM), driverId (рівність по FK), from/to (порівняння createdAt), search (LIKE-предикати по pickupLocation, deliveryLocation, description).

3) Модуль управління водіями. DriverService реалізує CRUD-операції для профілів водіїв. При створенні профілю перевіряється, що пов’язаний користувач має роль UserRole.DRIVER — інакше кидається InvalidUserRoleException. Оновлення статусу та місцезнаходження виконуються з pessimistic lock (findByIdForUpdate). DriverSpecification підтримує текстовий пошук по name, currentLocation та user.email через LIKE-предикати та фільтр по status через точний збіг. DriverMapper (MapStruct) виконує маппінг Driver → DriverResponseDto з вкладеним маппінгом user.email → email. Такий підхід забезпечує чітке розділення шарів застосунку та спрощує підтримку бізнес-логіки.

4) Модуль аудиту. AuditAspect реалізує перехоплення бізнес-дій через Spring AOP з pointcut-виразом @Around("@annotation(auditAction)"). У

системі 12 методів у UserController, DriverController та OrderController анотовані для аудиту. Аспект визначає IP, User-Agent, URI та HTTP-метод; читає email та userId через SecurityUtils; при успіху зберігає лог; при виключенні — логує помилку. При цьому після запису в журнал виключення повторно прокидається (re-throw), тому аудит фіксує невдалу спробу, але не приховує помилку від штатного механізму її обробки. Якщо результат реалізує Identifiable, витягується entityId. Зафіксований запис зберігається у таблицю журналу дій користувачів (user_logs) через AuditService, а назву сутності, до якої належить дія, визначено за іменем класу-джерела виклику. Це дозволяє централізовано контролювати історію дій користувачів без дублювання логіки в контролерах.

SecurityLogService.log() анотований @Async — виклик відбувається у окремому потоці ThreadPoolTaskExecutor, що унеможливлює затримку основного потоку запиту. Це підвищує загальну продуктивність системи при великому навантаженні.

3.2 Реалізація автоматизації через n8n

Реалізація інтеграції з n8n побудована на двох компонентах: AOP-аспекті WebhookAspect та сервісі WebhookService. WebhookAspect перехоплює всі методи, анотовані @WebhookEvent(WebhookEventType.X), використовуючи pointcut @Around("@annotation(webhookEvent)"). Після успішного виконання методу аспект викликає webhookService.send(), передаючи тип події та результат операції як payload. Такий механізм є узагальненим: щоб система генерувала нову подію, достатньо позначити відповідний метод анотацією @WebhookEvent — код самого аспекту змінювати не потрібно. Реалізацію аспекту наведено на рис. 3.3:

```
@Slf4j
@Aspect
@Component
@RequiredArgsConstructor
public class WebhookAspect {
    private final WebhookService webhookService;
    @Around("@annotation(webhookEvent)")
    public Object handleEvent(ProceedingJoinPoint joinPoint,
    WebhookEvent webhookEvent) throws Throwable {
        log.info(">>> WebhookAspect triggered: {}",
    webhookEvent.value());
        Object result = joinPoint.proceed();
        try {
            log.info(">>> WebhookAspect firing send for: {}",
    webhookEvent.value());
            webhookService.send(webhookEvent.value(), result);
        } catch (Exception e) {
            log.warn("Webhook event error [{}]: {}",
    webhookEvent.value(), e.getMessage());
        }
        return result;
    }
}
```

Рисунок 3.3 — Лістинг класу WebhookAspect

WebhookService.send() анотований @Async("taskExecutor"): виклик відбувається у фоновому потоці з пулу. Метод формує об’єкт WebhookEventDto (поля eventType та data), після чого надсилає його через RestTemplate.postForEntity() на URL, визначений у WebhookProperties.getLogisticsEventUrl() (властивість app.webhooks.logistics-event-url, змінна середовища WEBHOOK_LOGISTICS_EVENT_URL). У разі будь-якої помилки виключення перехоплюється та записується log.warn() — основний потік не переривається. Оскільки send() виконується асинхронно у пулі taskExecutor за принципом «fire-and-forget», а будь-яке виключення лише журналюється, недоступність n8n чи помилка доставки не впливають на виконання основної бізнес-операції. Такий підхід забезпечує слабке зв’язування між основною системою та системою автоматизації n8n. Це дозволяє продовжувати роботу бізнес-логіки навіть при частковій деградації зовнішніх сервісів. Реалізацію сервісу наведено на рис. 3.4.

```
@Slf4j
@Service
@RequiredArgsConstructor
public class WebhookService {
    private final RestTemplate restTemplate;
    private final WebhookProperties webhookProperties;
    @Async("taskExecutor")
    public void send(WebhookEventType webhookEventType, Object
    payload) {
        String url = webhookProperties.getLogisticsEventUrl();
        if (url == null || url.isBlank()) { log.warn("Webhook
    URL is blank"); return; }
        try {
            WebhookEventDto webhookEvent = new
    WebhookEventDto();
            webhookEvent.setEventType(webhookEventType);
            webhookEvent.setData(payload);
            restTemplate.postForEntity(url, webhookEvent,
    String.class);
        } catch (Exception e) {
            log.warn("Failed to send webhook [{}]: {}", url,
    e.getMessage());
        }
    }
}
```

Рисунок 3.4 — Лістинг класу WebhookService

Подієва модель системи поєднує два механізми надсилання. Більшість бізнес-подій генерується декларативно: анотацією @WebhookEvent позначено відповідні методи контролерів і сервісів (призначення, завершення та скасування замовлення, зміна статусу водія), після успішного виконання яких аспект автоматично надсилає подію. Окремо реалізовано подію наближення водія до пункту призначення (DRIVER_LOCATION_NEAR_DESTINATION). Вона є умовною — генерується не на кожен виклик методу оновлення локації, а лише за виконання просторової умови, тому її надсилання виконується прямим викликом WebhookService.send() усередині логіки перевірки відстані (див. п. 3.6). Поєднання декларативного підходу для безумовних подій з імперативним для умовної дозволяє зберегти чистоту аспектної моделі й водночас коректно обробити подію, що залежить від даних.

Workflow logistics-event у n8n (n8n/workflows/logistics-event.json) реалізований за схемою рис. 3.5. Webhook-вузол приймає POST-запит. Switch-вузол порівнює поле EventType зі значеннями типів подій та направляє потік до відповідної гілки. Кожна гілка формує поле message Set-вузлом і паралельно з'єднана з вузлом Send a text message (Telegram Bot API) та вузлом Send an Email (emailSend, SMTP), тобто подія генерує одночасно сповіщення в Telegram та електронною поштою. Гілка події наближення додатково містить Code-вузол дедуплікації, що пропускає лише перше повідомлення для кожного замовлення (реалізацію наведено у п. 3.6), оскільки backend надсилає цю подію повторно, поки водій перебуває в межах порогової зони. Workflow зберігається у репозиторії та монтується до контейнера n8n через Docker Volume (bind mount ./n8n/workflows:/workflows); налаштування credentials Telegram Bot API та SMTP здійснюється через захищене сховище credentials n8n. Контейнер запускається з офіційного образу n8nio/n8n і публікує веб-інтерфейс на порту 5678. Окрім каталогу з workflow, до контейнера підключено іменований том n8n_data, змонтований у /home/node/.n8n, у якому зберігаються внутрішні дані та credentials платформи — це забезпечує їх збереження між перезапусками контейнера.

Вхідний webhook-вузол налаштовано на HTTP-метод POST зі шляхом logistics-event, на який серверна частина надсилає подію за адресою зі змінної середовища WEBHOOK_LOGISTICS_EVENT_URL. Вузли Telegram та Email використовують спільне поле message, сформоване у відповідному Set-вузлі: повідомлення надходить у заданий Telegram-чат та на електронну пошту з темою «Logistics Update», а доступ до Telegram Bot API і SMTP-сервера забезпечують збережені у n8n облікові записи credentials.

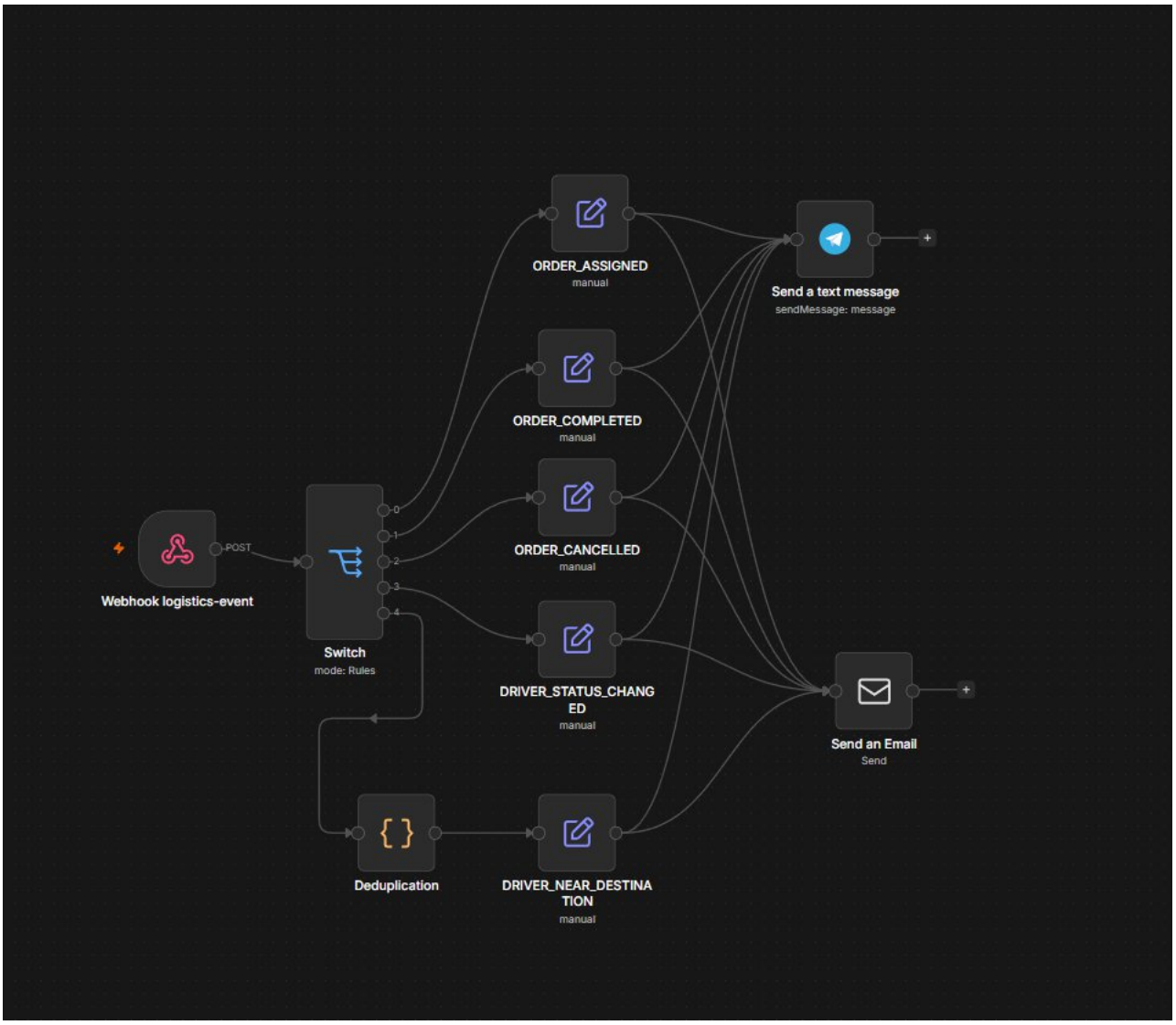

Рисунок 3.5 — Workflow logistics-event у платформі n8n

3.3 Реалізація клієнтського інтерфейсу

AuthProvider — React Context Provider, що при монтуванні надсилає POST /auth/refresh; при успіху записує accessToken у пам’ять через setAccessToken() та завантажує GET /users/me. Стан user та isLoading доступні через useAuth(). RoleGuard перевіряє роль автентифікованого користувача: DRIVER перенаправляється на /driver, неавторизовані — на /login.

API-клієнт (Axios) має два інтерсептори. Request-інтерсептор додає заголовок Authorization. Response-інтерсептор перехоплює HTTP 401: виконує POST /auth/refresh, отримує новий токен та повторює оригінальний запит.

Паралельні запити ставляться у failedQueue і відновлюються після успішного refresh.

Основні сторінки застосунку:

Orders — таблиця замовлень з пагінацією, фільтрами (пошук, статус, дата від/до) та призначенням водія через AssignDriverAction;

Drivers — таблиця водіїв з фільтром по статусу (DriverStatusBadge з кольоровим кодуванням);

Users — таблиця користувачів з фільтром по email та ролі (EnabledBadge);

DriverDashboard — панель водія з DriverOrderCard; кнопки Accept/Reject (при ASSIGNED) або Complete (при IN_PROGRESS);

CreateOrderPage — форма створення замовлення (Pickup Location, Delivery Location, Description).

Зовнішній вигляд основних сторінок клієнтського інтерфейсу наведено на рис. 3.6–3.8.

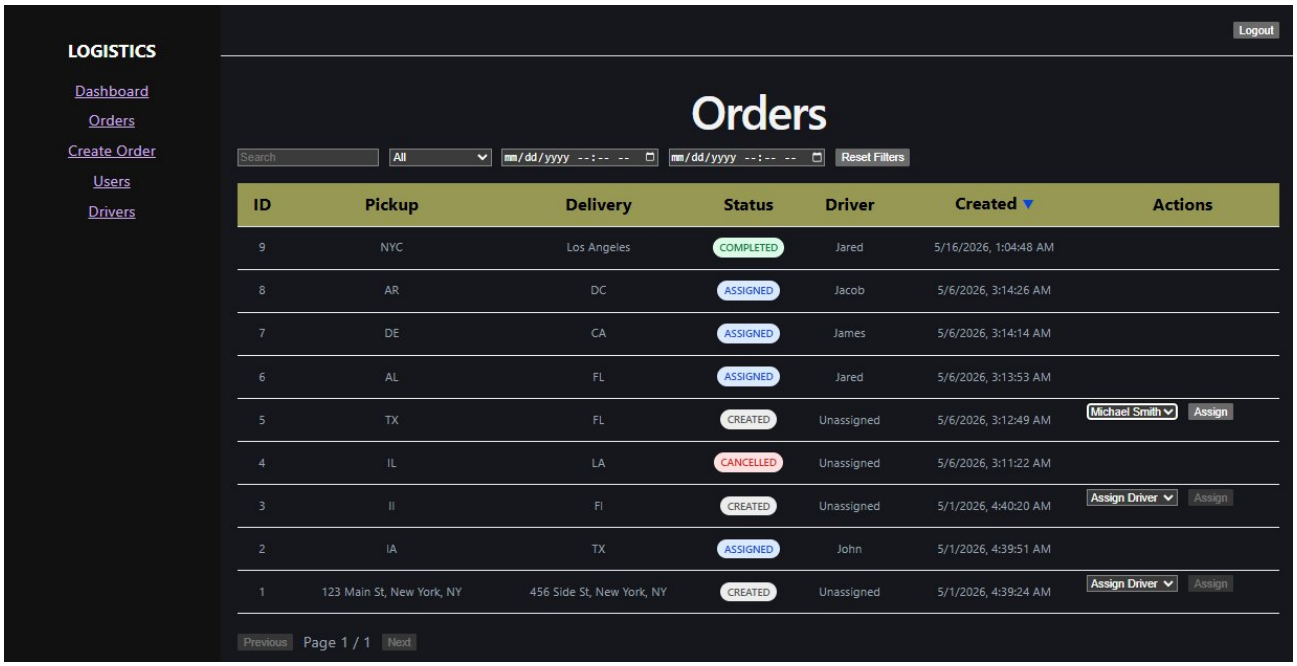

Рисунок 3.6 — Сторінка управління замовленнями

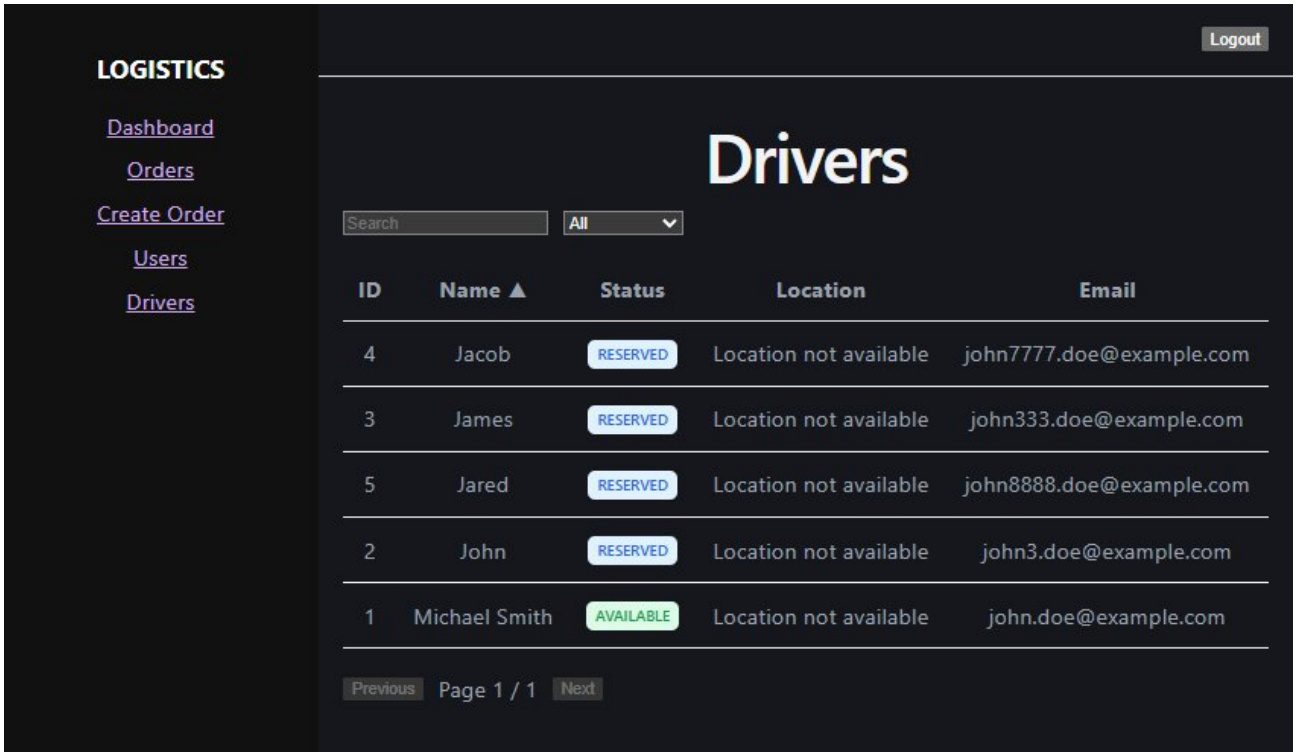

Рисунок 3.7 — Сторінка управління водіями

Рисунок 3.8 — Панель водія

3.4 Розгортання системи

Система розгортається через Docker Compose, що визначає чотири сервіси (таблиця 3.2).

Таблиця 3.2 — Сервіси Docker Compose

Сервіс	Образ / Збірка	Порт хост:контейнер	Опис
postgres	postgres:16	5433:5432	База даних PostgreSQL
backend	Збірка з Dockerfile (eclipse-temurin:26)	8080:8080	Spring Boot REST API
frontend	Збірка з frontend/Dockerfile (nginx:alpine)	3000:80	React SPA через nginx
n8n	n8nio/n8n	5678:5678	Платформа автоматизації

Backend-сервіс залежить від postgres (depends_on), схема БД мігрується через Flyway при кожному запуску. Dockerfile для backend реалізує двоетапну збірку: перший етап gradle:9.5.0-jdk26 виконує ./gradlew clean bootJar; другий етап eclipse-temurin:26 запускає зібраний JAR. Dockerfile для frontend збирає статику через node:20 (npm run build), потім nginx:alpine роздає її; nginx.config налаштовує SPA-маршрутизацію через try_files $uri /index.html на порту 80.

Конфігурація системи передається через змінні середовища:

DB_URL, DB_USERNAME, DB_PASSWORD — параметри підключення до PostgreSQL;

JWT_SECRET_KEY, JWT_ACCESS_EXPIRATION_TIME, JWT_REFRESH_EXPIRATION_TIME — налаштування JWT;

WEBHOOK_LOGISTICS_EVENT_URL — URL n8n webhook для всіх типів бізнес-подій (app.webhooks.logistics-event-url).

n8n workflow монтуються через named volume n8n_data та bind mount ./n8n/workflows:/workflows. Для розгортання необхідно виконати: `./gradlew clean build -x test`, після чого `docker compose up --build -d`.

3.5 Тестування

Тестування системи здійснюється на двох рівнях: автоматизовані unit- та інтеграційні тести, та функціональне тестування через Swagger UI [25, 26, 27].

3.5.1 Unit-тестування

Unit-тести реалізовані з використанням JUnit 5 та Mockito (@ExtendWith(MockitoExtension.class)). Всі залежності тестованих класів замінено mock-об'єктами через @Mock; сам клас інстанціюється через @InjectMocks без підняття Spring-контексту.

UserControllerTest перевіряє метод updatePassword(): успішне оновлення з коректним SecurityContext; кидання UserNotAuthenticatedException при відсутності Authentication; кидання InvalidPrincipalException при невалідному principal. SecurityContextHolder замінюється mock-реалізацією у @BeforeEach.

OrderServiceTest охоплює метод assignDriver() та суміжні операції: успішне призначення водія (перехід CREATED → ASSIGNED, AVAILABLE → RESERVED); кидання DriverNotAvailableException при статусі BUSY; кидання IllegalStateException при спробі призначити водія до замовлення не у статусі CREATED; успішне відхилення замовлення водієм (повернення у CREATED); успішне завершення (IN_PROGRESS → COMPLETED); кидання OrderNotFoundException при відсутньому замовленні.

DriverServiceTest перевіряє метод createDriver(): успішне створення профілю водія зі статусом AVAILABLE; кидання UserNotFoundException при відсутньому користувачі; кидання InvalidUserRoleException при некоректній ролі. Також перевірено updateDriverStatus(): успішне оновлення та кидання DriverNotFoundException.

Фрагмент тесту OrderServiceTest наведено на рис. 3.9.

```
    @Test
    void
assignDriver_Success_OrderAssignedAndDriverReserved() {
        Order order = new Order();
        order.setId(1L);
        order.setStatus(OrderStatus.CREATED);
        Driver driver = new Driver();
        driver.setId(2L);
        driver.setStatus(DriverStatus.AVAILABLE);
        AssignDriverRequestDto dto = new
AssignDriverRequestDto();
        dto.setDriverId(2L);
    when(orderRepository.findByIdForUpdate(1L)).thenReturn
(Optional.of(order));
    when(driverRepository.findByIdForUpdate(2L)).thenRetur
n(Optional.of(driver));
    when(orderRepository.save(order)).thenReturn(order);
    when(orderMapper.toDto(order)).thenReturn(new
OrderResponseDto());
        orderService.assignDriver(1L, dto);
        assertEquals(OrderStatus.ASSIGNED,
order.getStatus());
        verify(driverService).changeDriverStatus(driver,
DriverStatus.RESERVED); }
```

Рисунок 3.9 — Лістинг фрагмента OrderServiceTest

3.5.2 Інтеграційне тестування

Інтеграційні тести реалізовані у класі AssignDriverIntegrationTest (@SpringBootTest, @Testcontainers). Тестуванню підлягає endpoint PUT /orders/{id}/assign крізь повний стек: JwtAuthFilter → SecurityFilterChain → OrderController → OrderService → PostgreSQL ← GlobalExceptionHandler. Жодна компонента не замінюється mock-об'єктами.

Для тестової бази даних використовується Docker-контейнер postgres:16 (Testcontainers). Координати підключення передаються через @DynamicPropertySource до старту Spring-контексту; Flyway автоматично виконує всі міграції. Перед кожним тестом таблиці очищаються через JdbcTemplate (TRUNCATE ... RESTART IDENTITY CASCADE). JWT-токен генерується реальним JwtService.generateToken() і передається у заголовку Authorization: Bearer; JwtAuthFilter валідує підпис та встановлює SecurityContext через userRepository.findByEmail().

Визначено три сценарії: успішне призначення (HTTP 200, перевірка JSON-відповіді та стану БД); призначення недоступного водія (HTTP 409, відповідь GlobalExceptionHandler, БД без змін); конкурентне призначення — перевірка pessimistic locking.

У третьому сценарії два потоки через ExecutorService одночасно надсилають запити на призначення одного замовлення до різних AVAILABLE водіїв. Синхронізацію забезпечує CountDownLatch startGate (обидва потоки стартують одночасно) та CountDownLatch doneLatch (основний потік чекає завершення обох). Після виконання перевіряється: рівно один запит отримав HTTP 200, рівно один — HTTP 409; в БД рівно один Driver у статусі RESERVED, інший — AVAILABLE. Якщо pessimistic locking (SELECT ... FOR UPDATE) відсутній, обидві транзакції зчитали б status = CREATED одночасно і записали різних водіїв до одного замовлення, що тест виявив би як порушення бізнес-інваріанту.

Фрагмент перевірочних тверджень наведено на рис. 3.10.

```
        javastartGate.countDown();
        boolean      finished      =      doneLatch.await(10,
    TimeUnit.SECONDS);
        assertThat(statusCodes.stream().filter(s    ->    s    ==
    200).count()).isEqualTo(1);
        assertThat(statusCodes.stream().filter(s    ->    s    ==
    409).count()).isEqualTo(1);
        assertThat(finalOrder.getStatus()).isEqualTo(OrderStat
    us.ASSIGNED);
        assertThat(reservedDriversCount).isEqualTo(1);
```

Рисунок 3.10 — Лістинг assertions тесту pessimistic locking

3.5.3 Функціональне тестування

Функціональне тестування API виконується через Swagger UI (http://localhost:8080/swagger-ui/index.html). Інтерфейс дозволяє надсилати запити до всіх ендпоінтів безпосередньо з браузера з JWT-автентифікацією (кнопка Authorize). В таблиці 3.3 наведено перелік ключових тест-кейсів.

Таблиця 3.3 — Тест-кейси функціонального тестування API

Сценарій	Очікуваний результат
Успішний вхід з коректними обліковими даними	HTTP 200, accessToken у тілі відповіді, refreshToken у httpOnly cookie
Вхід із некоректним паролем	HTTP 400, код помилки INVALID_PASSWORD
Запит до /users без заголовку Authorization	HTTP 401 Unauthorized
DRIVER надсилає GET /users з валідним токеном	HTTP 403 Access Denied

Продовж. табл. 3.3

Сценарій	Очікуваний результат
DISPATCHER створює замовлення (POST /orders)	HTTP 200, поле status=CREATED у відповіді
Призначення водія зі статусом BUSY	HTTP 409 Conflict (DriverNotAvailableException)
Призначення доступного водія до замовлення CREATED	HTTP 200, order.status=ASSIGNED, driver.status=RESERVED
Водій завершує замовлення IN_PROGRESS	HTTP 200, order.status=COMPLETED, driver.status=AVAILABLE
Оновлення refresh-токену (POST /auth/refresh)	HTTP 200, новий accessToken, оновлена cookie
Повторне використання анульованого refresh-токену	HTTP 401 Unauthorized

Позитивні сценарії перевіряють коректність переходів станів замовлення (CREATED → ASSIGNED → IN_PROGRESS → COMPLETED) та пов'язану зміну статусу водія (RESERVED під час призначення, AVAILABLE після завершення). Негативні сценарії підтверджують роботу централізованої обробки винятків: класи GlobalExceptionHandler і SecurityExceptionHandler перетворюють доменні та безпекові винятки на відповідні HTTP-коди (400, 401, 403, 409). Зокрема, перевірено розмежування доступу за ролями (403 для недозволеної ролі, 401 за відсутності токена) та механізм ротації refresh-токенів, за якого повторне використання анульованого токена відхиляється з кодом 401 (RefreshTokenRevokedException).

Приклади виконання тестових запитів через Swagger UI наведено на рис. 3.11 та рис. 3.12.

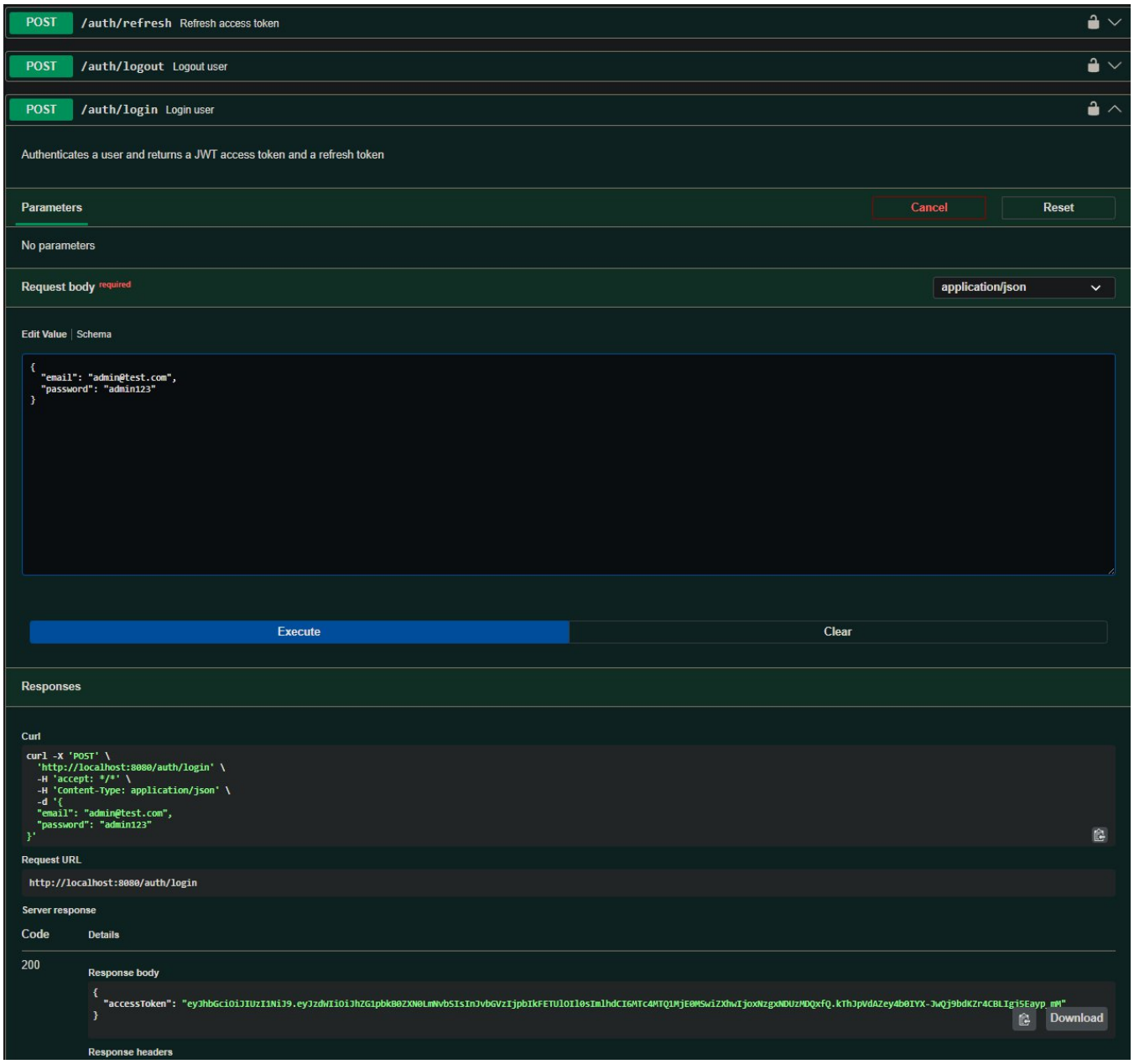

Рисунок 3.11 — Тестування POST /auth/login

На рис. 3.11 показано виконання запиту автентифікації POST /auth/login через інтерфейс Swagger UI. У тілі запиту у форматі application/json передано облікові дані користувача — email та пароль. Після натискання кнопки Execute сервер повертає код відповіді 200, а в тілі відповіді міститься JWT access-токен (поле accessToken). Це підтверджує коректну роботу механізму автентифікації та видачу токена доступу для подальших захищених запитів.

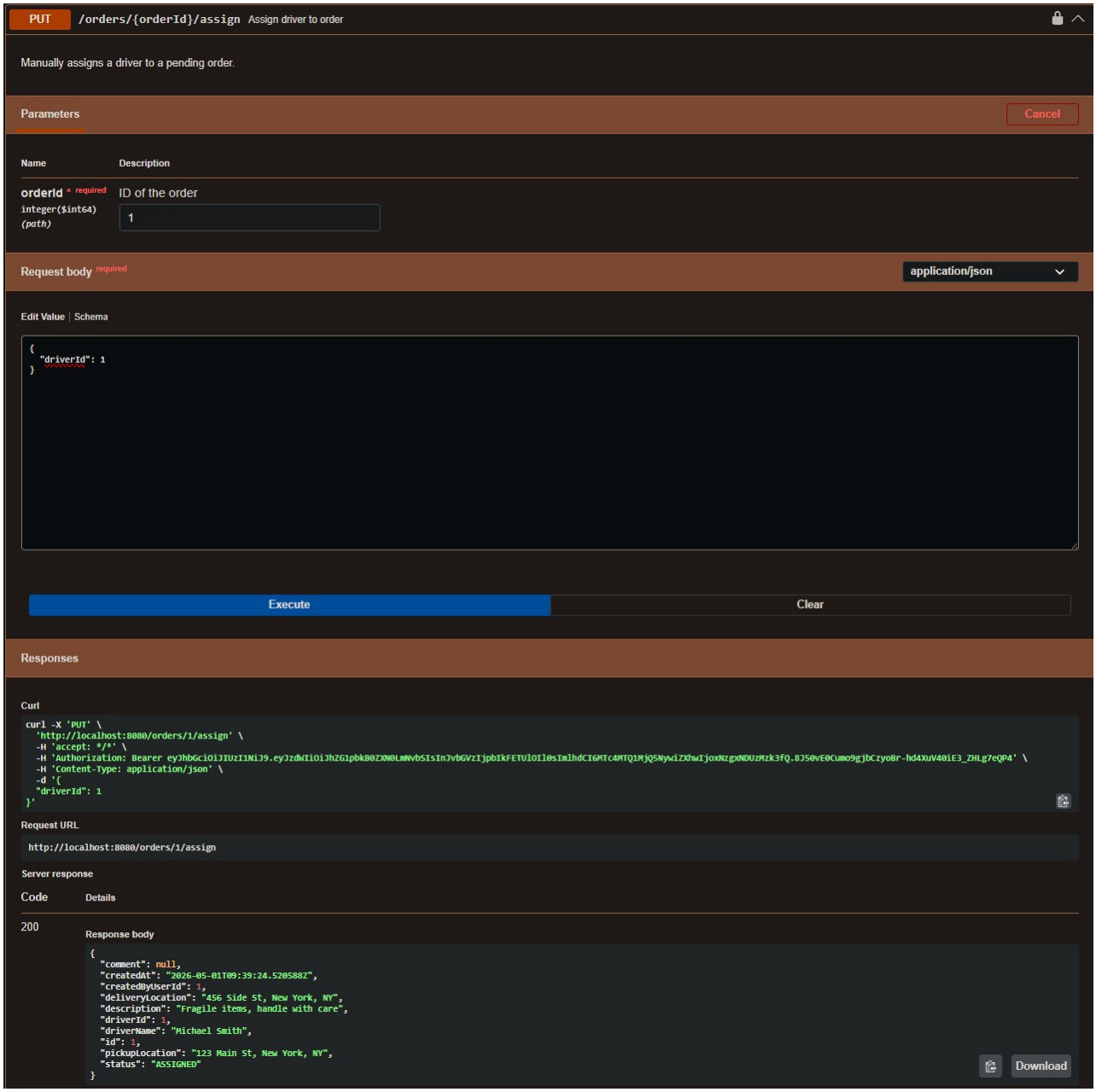

Рисунок 3.12 — Тестування PUT /orders/{id}/assign

На рис. 3.12 наведено виконання запиту PUT /orders/{orderId}/assign на призначення водія до замовлення. У параметрі шляху orderId вказано ідентифікатор замовлення, у тілі запиту — driverId, а сам запит супроводжується заголовком Authorization з Bearer-токеном, отриманим на етапі автентифікації. У відповідь сервер повертає код 200 та оновлений об'єкт замовлення, у якому поле status набуло значення ASSIGNED, а до замовлення прив'язано водія (driverId та driverName). Це демонструє коректне виконання захищеної бізнес-операції з контролем доступу за токеном.

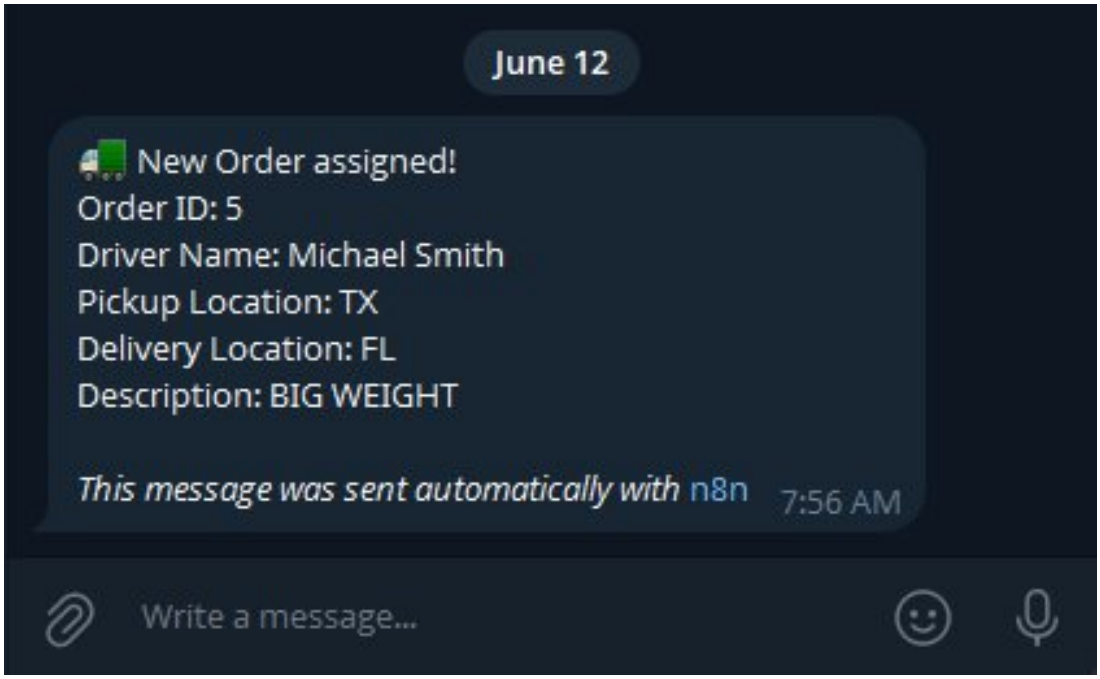

Рисунок 3.13 — Приклад Telegram сповіщення

Перевірка функціонування webhook-інтеграції здійснюється шляхом спостереження за отриманням Telegram-сповіщень та листів електронної пошти після виконання відповідних операцій. Коректність маршрутизації подій верифікується за відповідністю типу сповіщення типу операції. Приклад сповіщення наведено на рис. 3.13.

3.6 Реалізація апаратного модуля

1) Прошивка мікроконтролера ESP32. Прошивку розроблено у середовищі Arduino IDE. У процедурі ініціалізації setup() виконується підключення до мережі WiFi, ініціалізація OLED-дисплея та апаратного UART для GPS-приймача, після чого виконується автентифікація на сервері (POST /auth/login) і отримання JWT-токена, який зберігається у NVS. Облікові дані задаються на етапі прошивки. Скорочений фрагмент наведено на рис. 3.14.

```
const char* WIFI_SSID    = "...";
const char* WIFI_PASS    = "...";
const char* API_URL      = "http://192.168.0.10:8080";
const char* DRIVER_EMAIL = "driver@example.com";
const char* DRIVER_PASS  = "...";
const int   DRIVER_ID    = 5;
String token;
void setup() {
  pinMode(LED_PIN, OUTPUT);
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  while (WiFi.status() != WL_CONNECTED) delay(500);
// NEO-6M (UART2)
  Serial2.begin(9600, SERIAL_8N1, GPS_RX, GPS_TX);
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C); // OLED SSD1306
    (I2C)
// POST /auth/login -> JWT
  token = login(API_URL, DRIVER_EMAIL, DRIVER_PASS);
  saveToken(token);
}
```

Рисунок 3.14 — Лістинг фрагмента процедури ініціалізації прошивки

У головному циклі loop() відбувається безперервне зчитування NMEA-повідомлень із GPS-приймача та, раз на хвилину, надсилання поточних координат до сервера методом PUT /drivers/{id}/location із заголовком Authorization: Bearer. Результат запиту відображається світлодіодною індикацією. Фрагмент циклу наведено на рис. 3.15.

```
void loop() {
  while (Serial2.available()) gps.encode(Serial2.read());
  if (millis() - lastSend > 60000 && gps.location.isValid()) {
    String body = "{\"location\":\"" +
    String(gps.location.lat(), 6)
                + "," + String(gps.location.lng(), 6) + "\"}";
    HTTPClient http;
    http.begin(String(API_URL) + "/drivers/" + DRIVER_ID +
    "/location");
    http.addHeader("Content-Type", "application/json");
    http.addHeader("Authorization", "Bearer " + token);
    int code = http.PUT(body);
    digitalWrite(LED_PIN, code == 200 ? HIGH : LOW);
    http.end();
    lastSend = millis();
  } }
```

Рисунок 3.15 — Лістинг фрагмента головного циклу прошивки (полінг GPS та надсилання локації)

2) Реалізація обробки локації на серверній частині. Після збереження нових координат серверна частина шукає активне замовлення водія, обчислює відстань до пункту призначення і за умови наближення (менше порогу 2 км) генерує подію прямим викликом WebhookService.send(). Запобігання дублюванню реалізовано на стороні n8n (див. п. 3.6.3), тому додаткові поля у базі даних не потрібні. Ключові фрагменти наведено на рис. 3.16.

```
@Transactional
public DriverResponseDto updateDriverCurrentLocation(Long id,
    LocationDto dto) {
    Driver driver = driverRepository.findByIdForUpdate(id)
        .orElseThrow(() -> new DriverNotFoundException("Driver
    not found"));
    driver.setCurrentLocation(dto.getLocation());
    driverRepository.save(driver);
    orderRepository.findActiveByDriver(id,
    OrderStatus.IN_PROGRESS)
        .ifPresent(order -> {
         double dist = haversineMeters(dto.getLocation(),
    order.getDeliveryLocation());
            if (dist < 2000) {
                webhookService.send(
             WebhookEventType.DRIVER_LOCATION_NEAR_DESTINATION,
             NearDestinationDto.of(driver, order));
            }
        });
    return driverMapper.toDto(driver);
}
```

Рисунок 3.16 — Лістинг логіки умовної події

Координати у полях current_location та delivery_location зберігаються у форматі рядка «широта,довгота»; відстань обчислюється за формулою гаверсинусів допоміжним методом haversineMeters().

3) Інтеграція з n8n та тестування інтеграції. У гілці DRIVER_LOCATION_NEAR_DESTINATION перед вузлами сповіщення розміщено Code-вузол дедуплікації. Він використовує власне сховище стану workflow (getWorkflowStaticData), у якому фіксує ідентифікатори вже опрацьованих замовлень: перше повідомлення для кожного orderId

пропускається далі, наступні відхиляються. Реалізацію вузла наведено на рис. 3.17.

```
const store = $getWorkflowStaticData('global');
store.notified = store.notified || {};
const id = item?.json?.data?.orderId;
if (!id) {
    return [];
}
if (store.notified[id]) {
    return [];
}
store.notified[id] = true;
return $input.all();
```

Рисунок 3.17 — Лістинг вузла дедуплікації сповіщень у n8n

Після Code-вузла гілку, як і решту, паралельно під'єднано до вузлів Telegram та Email, які формують повідомлення виду «Водій {name} під'їжджає до пункту призначення замовлення #{orderId}».

Тестування інтеграції проводилося у два етапи. На першому етапі коректність серверної логіки перевірялася незалежно від трекера за допомогою утиліти curl (та Postman), що відтворює запит, ідентичний тому, який надсилає ESP32, наведено на рис. 3.18

```
curl -X PUT http://localhost:8080/drivers/5/location \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1NiJ9..." \
  -H "Content-Type: application/json" \
  -d '{"location":"49.5512,25.5905"}'
```

Рисунок 3.18 — Лістинг curl запиту

На другому етапі модуль випробовувався у складі системи: backend розгортався локально у Docker, ESP32 та сервер під'єднувалися до спільної мережі WiFi; за відсутності реального GPS-сигналу координати підмінювалися тестовими значеннями. Перевірявся повний ланцюг

проходження даних: трекер → REST-ендпоінт → перевірка відстані на backend → подія → маршрутизація у n8n → дедуплікація → сповіщення. Перелік сценаріїв наведено у таблиці 3.4.

Таблиця 3.4 — Тест-кейси інтеграції апаратного модуля

Сценарій	Очікуваний результат
Валідні координати, відстань більша за поріг	HTTP 200, локацію оновлено, подію не згенеровано
Координати в межах порогу, перше спрацювання	HTTP 200, згенеровано подію, n8n пропускає сповіщення (Telegram + Email)
Повторні координати в межах порогу по тому ж замовленню	HTTP 200, подію згенеровано, вузол дедуплікації n8n відхиляє повторне сповіщення
Запит без заголовка Authorization	HTTP 401 Unauthorized
Некоректний формат координат	HTTP 400 Bad Request

Коректність маршрутизації події верифікувалася за фактом надходження відповідного Telegram-повідомлення та листа електронної пошти після перетину порогової відстані.

РОЗДІЛ 4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Аналіз умов праці та шкідливих виробничих факторів при розробці програмного забезпечення

Розробка програмного забезпечення є різновидом інтелектуальної (розумової) праці, що виконується переважно в офісному або домашньому середовищі. Основним знаряддям праці розробника є персональний комп'ютер з відеодисплейним терміналом (ВДТ), а характер діяльності передбачає тривалу взаємодію людини з екраном, клавіатурою та маніпулятором у межах одного робочого місця. Відповідно до ДСанПіН 3.3.2.007-98 [28], робота з ВДТ належить до категорії робіт з підвищеним зоровим навантаженням та тривалим статичним навантаженням на м'язи шийно-плечового поясу й попереку. Хоча виробничий процес не пов'язаний із прямою фізичною небезпекою, постійна дія низки шкідливих факторів за відсутності належної організації робочого місця здатна спричинити стійке погіршення здоров'я працівника та зниження його працездатності.

Першим і найвагомішим фактором є зорове навантаження. Тривала фіксація погляду на екрані монітора, робота з дрібним шрифтом, висока яскравість і мерехтіння зображення зумовлюють розвиток астенопії — зорової втоми, що проявляється відчуттям різі, сухості та печіння в очах, розпливчастістю зображення й головним болем. Сукупність таких симптомів у міжнародній практиці позначають як комп'ютерний зоровий синдром. Причиною є також зменшення частоти кліпання під час зосередженої роботи, що призводить до пересихання слізної плівки рогівки. Для запобігання перевтомі органів зору нормується освітленість робочої поверхні в межах 300–500 лк відповідно до ДБН В.2.5-28:2018 [29], а також рекомендується

періодичне перенесення погляду на віддалені об'єкти за принципом регулярних коротких пауз.

Другим суттєвим фактором є статичне м'язове навантаження, спричинене вимушеною тривалою нерухомою позою. Багатогодинне сидіння з нахиленою головою та підведеними передпліччями перевантажує м'язи шиї, плечового поясу та поперекового відділу хребта, що з часом може призвести до формування стійких порушень постави, остеохондрозу та тунельних синдромів кистей рук. До факторів виробничого середовища належить також електромагнітне випромінювання низьких частот, джерелом якого є монітор та інше периферійне обладнання, проте сучасні рідкокристалічні дисплеї характеризуються істотно нижчим його рівнем порівняно з електронно-променевими.

Значний вплив на самопочуття та продуктивність працівника має мікроклімат приміщення. Згідно з ДСанПіН 3.3.2.007-98, для робочого місця оператора ВДТ оптимальною вважається температура повітря в межах 20–25 °С, відносна вологість 40–60 % та швидкість руху повітря не більше 0,1 м/с; рівень шуму не повинен перевищувати 50 дБА. Недотримання цих параметрів — підвищена температура, надмірна сухість повітря через роботу систем опалення, протяги чи сторонній шум — знижує концентрацію уваги, прискорює настання втоми та погіршує загальне самопочуття. Окремо слід виділити психоемоційне навантаження: робота розробника пов'язана з постійним вирішенням складних інтелектуальних задач, відповідальністю за результат, роботою в умовах стислих термінів (дедлайнів) і необхідністю безперервного засвоєння нових технологій. Тривала дія цих чинників є джерелом хронічного стресу та може призводити до професійного вигорання, тому під час нервових напружень доцільно своєчасно звертатися до відповідних фахівців та застосовувати методи психологічного розвантаження.

Таким чином, попри відсутність явних фізичних небезпек, праця розробника програмного забезпечення супроводжується комплексом шкідливих факторів зорового, фізіологічного, фізичного та психоемоційного

характеру. Це обумовлює необхідність дотримання нормативних вимог до організації робочого місця та раціонального режиму праці й відпочинку.

4.2 Вимоги до організації робочого місця та режиму праці розробника

Належна організація робочого місця є основним засобом зниження впливу шкідливих виробничих факторів, проаналізованих у попередньому підрозділі. Вимоги до приміщень для роботи з відеодисплейними терміналами встановлено НПАОП 0.00-1.28-10 [30]. Згідно з ними, площа на одне робоче місце оператора ВДТ має становити не менше 6,0 м², а об'єм приміщення — не менше 20,0 м³ на одного працівника. Такі норми забезпечують достатній повітрообмін, унеможливлюють скупченість обладнання та сприяють підтриманню нормованих параметрів мікроклімату. Приміщення повинне мати природне та штучне освітлення, при цьому монітор слід розташовувати так, щоб віконні прорізи знаходилися збоку (переважно ліворуч) від оператора — це запобігає прямому засвічуванню екрана та утворенню відблисків, що додатково навантажують зір. Нормована освітленість робочої поверхні від штучного освітлення становить 300–500 лк відповідно до ДБН В.2.5-28:2018.

Не менш важливим є дотримання ергономічних параметрів безпосередньо робочого місця. Відстань від очей працівника до екрана монітора має становити 600–700 мм, а верхній край екрана слід розміщувати на рівні очей або нижче на 10–15°, що забезпечує природне положення голови та шиї. Висота робочої поверхні стола має регулюватися або відповідати діапазону 680–800 мм, а кут між плечем і передпліччям під час роботи з клавіатурою повинен становити 90–110°. Робоче крісло має бути обладнане механізмом регулювання висоти сидіння та кута нахилу спинки з підтримкою поперекового відділу хребта, що дозволяє працівникові підтримувати фізіологічно правильну позу й рівномірно розподіляти статичне навантаження. Клавіатуру та маніпулятор слід розташовувати на одній

поверхні в зоні досяжності рук, уникаючи зайвого витягування та згинання кистей.

Поряд з організацією простору вирішальне значення має раціональний режим праці та відпочинку. Відповідно до ДСанПіН 3.3.2.007-98, тривалість безперервної роботи з монітором не повинна перевищувати 2 годин, після чого необхідно робити перерву тривалістю не менше 15 хвилин. Загальна тривалість роботи з ВДТ протягом робочого дня для операторів-програмістів не повинна перевищувати 6 годин. Під час регламентованих перерв рекомендується не залишатися за робочим столом, а виконувати комплекс вправ для зняття зорової втоми та розминки м'язів шиї, плечового поясу й кистей рук. Зокрема, для зниження навантаження на органи зору ефективним є періодичне перенесення погляду з екрана на віддалені об'єкти, а також короткочасне заплющування очей. Дотримання такого режиму дозволяє запобігти накопиченню втоми й підтримувати стабільну працездатність протягом усього робочого дня.

Окремим організаційним заходом охорони праці є проведення інструктажів. Інструктажі з питань охорони праці проводяться на всіх підприємствах, в установах і організаціях незалежно від характеру їх трудової діяльності, підлеглості та форми власності; їх мета — навчити працівника правильно й безпечно для себе та навколишнього середовища виконувати свої трудові обов'язки. Своєчасне ознайомлення розробника з вимогами безпеки, правилами організації робочого місця та режиму праці, а також контроль за їх дотриманням з боку роботодавця є запорукою збереження здоров'я працівника й тривалого підтримання високої продуктивності праці.

Отже, комплексне дотримання нормативних вимог до приміщення, ергономіки робочого місця та режиму праці й відпочинку, регламентованих ДСанПіН 3.3.2.007-98, НПАОП 0.00-1.28-10 та ДБН В.2.5-28:2018, дозволяє мінімізувати вплив шкідливих виробничих факторів на організм розробника програмного забезпечення та забезпечити безпечні й сприятливі умови його професійної діяльності.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи було виконано наступне:

– Проведено аналіз предметної галузі вантажних перевезень та визначено основні задачі диспетчерської служби. Здійснено огляд існуючих програмних рішень і встановлено, що розглянуті рішення не забезпечують одночасно необхідного рівня гнучкості налаштування, відкритості коду та інтеграції з платформою n8n, що стало підставою для розробки власної системи. Сформульовано функціональні та нефункціональні вимоги, визначено рольову модель RBAC. Особливу увагу приділено забезпеченню безпеки даних та контролю доступу на рівні бізнес-операцій. Обґрунтовано вибір технологічного стеку: Java / Spring Boot, PostgreSQL, n8n, Docker/Docker Compose, React / TypeScript.Отримана архітектура забезпечує масштабованість системи та можливість незалежного розвитку окремих модулів.

– Спроєктовано всі ключові компоненти системи. Розроблено узагальнену тришарову архітектуру з подвійним AOP-механізмом (аудит та webhook-автоматизація). Спроєктовано схему бази даних з ENUM-типами та Flyway-міграціями. Визначено специфікацію REST API з рольовим контролем доступу. Це дозволило уніфікувати взаємодію між клієнтською та серверною частинами системи. Розроблено модель безпеки JWT + Refresh Token Rotation з pessimistic locking. Спроєктовано уніфікований n8n-workflow з маршрутизацією бізнес-подій через Switch-вузол та паралельною відправкою сповіщень у Telegram і Email, що забезпечує гнучку оркестрацію подій та можливість швидкого додавання нових каналів сповіщень. Спроєктовано апаратний модуль моніторингу місцезнаходження водія на базі мікроконтролера ESP32 із приймачем NEO-6M, для якого розроблено комплект графічного матеріалу: структурну, функціональну, принципову та монтажну схеми. Це забезпечує повний цикл збору, передачі та обробки геоданих у режимі реального часу.

– Описано реалізацію всіх ключових модулів системи. Реалізовано модуль автентифікації JWT + Refresh Token Rotation з механізмом token reuse detection та pessimistic locking для конкурентних операцій. Реалізовано AOP-based webhook-інтеграцію через WebhookAspect та WebhookService: метод send(WebhookEventType, Object) надсилає WebhookEventDto на endpoint для типів подій з паралельними сповіщеннями у Telegram і Email. Реалізовано двошаровий AOP-аудит та React SPA з рольовою маршрутизацією. Реалізовано апаратний модуль (GPS-трекер на базі ESP32): прошивку з полінгом GPS-координат та їх надсиланням до серверної частини, умовну подію наближення до пункту призначення з перевіркою відстані за формулою гаверсинусів на backend і дедуплікацією сповіщень на рівні оркестратора n8n. Система розгортається у Docker Compose з контейнерами для серверної частини, клієнтського застосунку, бази даних та платформи автоматизації.

Усі ключові бізнес-процеси системи взаємодіють через подієву модель, що зменшує зв’язність між модулями та спрощує їх масштабування. Запроваджені механізми асинхронної обробки та чергування подій забезпечують стабільну роботу системи. Отримані результати підтверджують коректність обраної архітектурної моделі та її придатність до подальшого розширення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Жаровський Р.О., Луцик Н.С., Осухівська Г.М., Паламар А.М., Тиш Є.В. Методичні вказівки до виконання кваліфікаційної роботи бакалавра для здобувачів першого (бакалаврського) рівня вищої освіти за спеціальністю 123 «Комп'ютерна інженерія» усіх форм навчання. – Тернопіль: ТНТУ, 2024. – 39 с.

2. OWASP Top Ten 2021: веб-ресурс. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 08.05.2026).

3. Луцків А., Лупенко С., Пасічник В. Паралельні та розподільнені обчислення: навч. посібник. – Львів: Видавництво «Магнолія 2006», 2024. – 566 с.

4. OpenAPI Specification 3.0.3: специфікація. URL: https://spec.openapis.org/oas/v3.0.3 (дата звернення: 05.05.2026).

5. Yatsyshyn V., Pastukh O., Zharovskyi R., Shabliy N.Software tool for productivity metrics measure of relational database management system. Mathematical Modeling. No 1 (48). 2023. P. 7-17.

6. Walls C. Spring Boot in Action. – Shelter Island: Manning Publications, 2016. – 264 p.

7. Pillai A. Mastering Spring Boot 3.0. – Birmingham: Packt Publishing, 2023. – 454 p.

8. Шеремета В.З., Жаровський Р.О. Використання Spring Boot та інтеграція другорядних інструментів для створення сучасних веб-додатків // Матеріали XII міжн. наук.-техн. конф. молодих учених та студентів «Актуальні задачі сучасних технологій» (11–12 грудня 2023). – Тернопіль: ТНТУ, 2023. – С. 439.

9. Spring Framework Reference Documentation: офіційна документація. URL: https://docs.spring.io/spring-framework/docs/current/reference/html/ (дата звернення: 15.05.2026).

10. Spilca L. Spring Security in Action. – Shelter Island: Manning Publications, 2020. – 480 p.

11. Obe R., Hsu L. PostgreSQL: Up and Running. – Sebastopol: O'Reilly Media, 2017. – 356 p.

12. n8n Workflow Automation Documentation: офіційна документація. URL: https://docs.n8n.io/ (дата звернення: 18.05.2026).

13. Docker Documentation: офіційна документація. URL: https://docs.docker.com/ (дата звернення: 10.05.2026).

14. Banks A., Porcello E. Learning React. – Sebastopol: O'Reilly Media, 2020. – 310 p.

15. React Documentation: офіційна документація. URL: https://react.dev/ (дата звернення: 14.05.2026).

16. Newman S. Building Microservices. – Sebastopol: O'Reilly Media, 2021. – 616 p.

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

18. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. – Boston: Addison-Wesley, 1994. – 395 p.

19. Kiczales G., Lamping J., Mendhekar A. [et al.] Aspect-Oriented Programming Proceedings of ECOOP. – Jyväskylä, 1997. – P. 220–242.

20. PostgreSQL 16 Documentation: офіційна документація. URL: https://www.postgresql.org/docs/16/ (дата звернення: 12.05.2026).

21. Flyway Database Migrations Documentation: офіційна документація. URL: https://documentation.red-gate.com/fd (дата звернення: 11.05.2026).

22. Fielding R.T. Architectural Styles and the Design of Network-based Software Architectures: PhD dissertation. – Irvine: University of California, 2000. – 162 p.

23. Spring Security Reference Documentation: офіційна документація. URL: https://docs.spring.io/spring-security/reference/index.html (дата звернення: 15.05.2026).

24. Паламар М.І., Стрембіцький М.О., Паламар А.М. Проектування комп'ютеризованих вимірювальних систем і комплексів: навч. посібник. – Тернопіль: ТНТУ, 2019. – 150 с.

25. Шеремета В.З., Жаровський Р.О. Тестування веб-додатків, розроблених на основі Spring Boot за допомогою Testing. Матеріали XI наук.-техн. конф. ТНТУ «Інформаційні моделі, системи та технології» (18–19 грудня 2024). – Тернопіль: ТНТУ, 2024. – С. 162.

26. Свергун С., Жаровський Р. Тестування програмного забезпечення, побудованого на мікросервісній архітектурі. Матеріали X наук.-техн. конф. ТНТУ «Інформаційні моделі, системи та технології». – Тернопіль: ТНТУ, 2022. – С. 92.

27. Слюз І., Жаровський Р. Критерії ефективності тестування комп'ютерної інформаційної системи. Матеріали XI міжн. наук.-техн. конф. молодих учених та студентів «Актуальні задачі сучасних технологій». – Тернопіль: ТНТУ, 2022. – С. 174.

28. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин ДСанПіН 3.3.2.007-98. – Київ: МОЗ України, 1998.

29. ДБН В.2.5-28:2018. Природне і штучне освітлення. – Київ: Мінрегіон України, 2018.

30. НПАОП 0.00-1.28-10. Правила охорони праці під час експлуатації електронно-обчислювальних машин. – Київ: Держгірпромнагляд, 2010.

Додаток А

Технічне завдання

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп’ютерно-інформаційних систем і програмної інженерії

Кафедра комп’ютерних систем та мереж

“Затверджую”

Завідувач кафедри КС

_____ Осухівська Г.М.

“ 2 ” лютого 2026 р.

Комп’ютерна система керування процесами вантажних перевезень із використанням платформи n8n

ТЕХНІЧНЕ ЗАВДАННЯ

на _11_ листках

Вид робіт: Кваліфікаційна робота

На здобуття освітнього ступеня «Бакалавр»

Спеціальність 123 «Комп’ютерна інженерія»

«УЗГОДЖЕНО»

Керівник кваліфікаційної роботи

_____ к.т.н., ст. викл. Стадник Н.Б.

“ 2 ” лютого 2026 р.

«ВИКОНАВЕЦЬ»

Студент групи СІ-41

_____ Кармазин О.Б.

“ 2 ” лютого 2026 р.

Тернопіль 2026

1 Загальні відомості

1.1 Повна назва та її умовне позначення

Повна назва теми кваліфікаційної роботи: «Комп’ютерна система керування процесами вантажних перевезень із використанням платформи n8n».

Умовне позначення кваліфікаційної роботи: КС КРБ 123.СІ-22-167.00.00

1.2 Виконавець

Студент групи СІ-41, факультету комп’ютерно-інформаційних систем і програмної інженерії, кафедри комп’ютерних систем та мереж, Тернопільського національного технічного університету імені Івана Пулюя, Кармазин О.Б

1.3 Підстава для виконання роботи

Підставою для виконання кваліфікаційної роботи є наказ по університету №4/9-188 від 24.04.2026 р

1.4 Планові терміни початку та завершення роботи

Плановий термін початку виконання кваліфікаційної роботи – 26.01.2026 р.

Плановий термін завершення виконання кваліфікаційної роботи – 14.06.2026 р.

1.5 Порядок оформлення та пред’явлення результатів роботи

Порядок оформлення пояснювальної записки та графічного матеріалу здійснюється у відповідності до чинних норм та правил ISO, ЕСКД, ЕСПД та ДСТУ.

Пред’явлення проміжних результатів роботи з виконання кваліфікаційної роботи здійснюється у відповідності до графіку, затвердженого керівником роботи. Попередній захист кваліфікаційної роботи відбувається при готовності роботи – наявності пояснювальної записки та графічного матеріалу.

Пред’явлення результатів кваліфікаційної роботи відбувається шляхом захисту на відповідному засіданні ЕК, ілюстрацією основних досягнень за допомогою графічного матеріалу.

2 Призначення і цілі створення системи

2.1 Призначення системи

Комп'ютерна система керування процесами вантажних перевезень із використанням платформи n8n призначена для автоматизації диспетчерської діяльності транспортно-логістичного підприємства: розподілу замовлень між водіями, відстеження стану їх виконання та своєчасного інформування відповідальних осіб про ключові події перевізного процесу. Система враховує наявність різних категорій користувачів (диспетчер, адміністратор, водій), розмежування їх прав доступу та ведення журналу дій.

Система може бути реалізована у вигляді окремого програмного продукту або як складова підсистема інформаційної інфраструктури транспортного підприємства. До її складу входять модуль автоматизації бізнес-процесів на основі платформи n8n та апаратний модуль моніторингу місцезнаходження водія на базі мікроконтролера ESP32. Користувачами системи є малі та середні транспортно-логістичні підприємства, диспетчерські служби й окремі перевізники. Застосування low-code платформи автоматизації

забезпечує гнучкість і розширюваність рішення та спрощує підключення нових каналів сповіщення.

2.2 Мета створення системи

Метою створення системи є автоматизація диспетчерських процесів, скорочення часу призначення водіїв до замовлень, забезпечення прозорості операцій через журналювання та своєчасне інформування про ключові події перевізного процесу.

Досягнення поставленої мети можливе шляхом розв'язання наступних завдань:

– аналіз предметної галузі та існуючих програмних рішень для диспетчеризації;

– визначення вимог до системи та обґрунтування вибору технологічного стеку;

– проєктування архітектури серверної частини та схеми бази даних;

– реалізація REST API з рольовою моделлю доступу та JWT-автентифікацією;

– розробка модуля автоматизації бізнес-процесів на основі платформи n8n;

– реалізація системи аудиту та журналювання подій;

– розробка апаратного модуля моніторингу місцезнаходження водія та його інтеграція із серверною частиною;

– тестування ключових компонентів та розгортання системи у контейнерному середовищі.

2.3 Характеристика об'єкту

Основними функціями системи є створення та розподіл замовлень, відстеження стану їх виконання, керування статусами водіїв і автоматичне

сповіщення про ключові події, що дає змогу диспетчерській службі ефективно координувати перевізний процес.

Для автоматизації процесу диспетчеризації використовуються наступні структурні компоненти:

– серверна частина з реалізацією бізнес-логіки та REST API;

– реляційна база даних для зберігання інформації про замовлення, водіїв та користувачів;

– платформа автоматизації бізнес-процесів для маршрутизації подій і надсилання сповіщень;

– клієнтський веб-застосунок для диспетчерів і водіїв;

– апаратний модуль моніторингу місцезнаходження водія.

3 Вимоги до системи

3.1 Вимоги до системи в цілому

3.1.1 Вимоги до структури та функціонування системи

До структури системи висуваються такі основні вимоги:

– наявність серверної частини з реалізацією бізнес-логіки та REST API;

– наявність реляційної бази даних для зберігання інформації системи;

– наявність модуля автоматизації бізнес-процесів на основі платформи n8n;

– наявність клієнтського веб-інтерфейсу для взаємодії користувачів;

– наявність апаратного модуля моніторингу місцезнаходження водія;

– можливість інтеграції із зовнішніми каналами сповіщення.

До основних функціональних вимог належать:

– здатність створювати замовлення та призначати їх водіям;

– можливість змінювати статуси замовлень і водіїв із дотриманням визначених переходів;

– здатність автентифікувати користувачів та розмежовувати доступ за ролями;

– можливість інтеграції із зовнішніми каналами сповіщення.

– здатність генерувати бізнес-події та маршрутизувати їх до каналів сповіщення;

– можливість надсилання повідомлень у месенджер та електронною поштою;

– здатність фіксувати дії користувачів і події безпеки в журналі аудиту;

– можливість приймати та обробляти дані про координати місцезнаходження водіїв.

3.1.2 Вимоги до способів та засобів зв’язку між компонентами системи

Зв'язок між компонентами системи відповідає вимогам до взаємодії елементів комп'ютерних систем на основі технології клієнт-сервер та виконується за протоколами HTTP/HTTPS. Обмін даними здійснюється у форматі JSON через REST API, а передавання бізнес-подій до платформи автоматизації – за допомогою механізму webhook.

3.1.3 Вимоги по діагностуванню системи

Діагностування функціонування системи відбувається згідно з графіком профілактичних заходів або при виникненні нештатних ситуацій. Для діагностування використовуються вбудовані засоби журналювання серверної частини, інструменти моніторингу стану контейнерів та засоби перегляду історії виконання процесів у платформі автоматизації.

3.1.4 Перспективи розвитку, проєктування системи

Перспективами розвитку системи є підключення нових каналів сповіщення, розширення переліку оброблюваних бізнес-подій та інтеграція із суміжними системами підприємства. Модернізація можлива у випадку зміни бізнес-вимог диспетчерського процесу або міграції окремих компонентів на

інші платформи класу Low/No code. Утилізація системи виконується у випадку морального старіння використовуваних технологій.

3.1.5 Вимоги до апаратного забезпечення

Мінімальні вимоги до серверної станції:

– тактова частота процесора – 2,0 ГГц з 4-ма паралельними потоками;

– об'єм оперативної пам'яті – 8 ГБ;

– об'єм дискового простору – 64 ГБ.

Вимоги до клієнтських станцій:

– тактова частота процесора – 1,6 ГГц;

– об'єм оперативної пам'яті – 4 ГБ;

– наявність мережевого з'єднання.

Апаратний модуль моніторингу реалізується на базі мікроконтролера ESP32 із приймачем NEO-6M та модулем бездротового зв'язку WiFi.

3.1.6 Вимоги до програмного забезпечення

Вимоги до програмного забезпечення клієнтських станцій – операційна система будь-якого типу, наявність сучасного веб-браузера.

Вимоги до програмного забезпечення сервера – наявність середовища виконання серверної частини, системи керування реляційною базою даних, платформи автоматизації бізнес-процесів та засобів контейнеризації для розгортання компонентів системи.

3.2 Показники призначення

Система повинна передбачати можливість масштабування. Можливості масштабування забезпечуються засобами використовуваного базового програмного і технічного забезпечення, зокрема контейнеризації компонентів системи.

3.2.1 Вимоги до надійності

Система повинна забезпечувати працездатність та відновлення своїх функцій при виникненні наступних ситуацій:

– при збоях в системі електропостачання апаратної частини;

– при помилках в роботі апаратних засобів;

– при помилках, пов'язаних з програмним забезпеченням (ОС і драйвери пристроїв).

Для захисту апаратури від стрибків напруги і комутаційних завад повинні застосовуватися мережні фільтри.

3.3 Вимоги до безпеки

Зовнішні елементи технічних засобів системи, що перебувають під напругою, повинні мати захист від випадкового дотику, а самі технічні засоби – занулення або захисне заземлення.

Система електроживлення повинна забезпечувати захисне вимикання при перевантаженнях і коротких замиканнях у колах навантаження, а також аварійне ручне вимикання.

Загальні вимоги пожежної безпеки повинні відповідати нормам на побутове електрообладнання. У разі пожежі не мають виділятися отруйні гази і дим. Після зняття електроживлення має бути доступне застосування будь-яких засобів пожежогасіння.

3.3.1 Вимоги до експлуатації, технічного обслуговування, ремонту і зберігання компонентів системи

Мікроклімат у приміщеннях повинен відповідати нормам виробничого мікроклімату по ДСН 3.3.6.042-99:

– температура повітря в межах від +10 °С до +35 °С;

– відносна вологість повітря при 25 °С в межах від 30% до 80%;

– атмосферний тиск 760±25 мм рт. ст.

Періодичне технічне обслуговування використовуваних технічних засобів має проводитися відповідно до вимог технічної документації, але не рідше ніж один раз на рік. Обслуговування і тестування повинні включати перевірку всіх використовуваних засобів: серверного обладнання, апаратного модуля моніторингу, систем передачі даних та пристроїв безперебійного живлення. На підставі результатів тестування проводиться аналіз причин виявлених дефектів і вживаються заходи щодо їх ліквідації.

3.4 Вимоги до захисту інформації від несанкціонованого доступу

Система повинна забезпечувати захист від несанкціонованого доступу на рівні не нижче встановленого вимогами, що пред'являються до категорії 1Д за класифікацією чинного документа «Автоматизовані системи. Захист від несанкціонованого доступу до інформації. Класифікація автоматизованих систем».

Компоненти підсистеми захисту від НСД повинні забезпечувати:

– ідентифікацію та автентифікацію користувача на основі механізму JWT;

– перевірку повноважень користувача при роботі з системою;

– розмежування доступу користувачів відповідно до рольової моделі (RBAC).

Рівень захищеності засобів обчислювальної техніки, що здійснюють обробку конфіденційної інформації, повинен відповідати вимогам класу захищеності згідно з чинними нормативними документами щодо захисту від несанкціонованого доступу до інформації.

3.4.1 Вимоги по збереженню інформації при аваріях

Інформація при виникненні аварійних ситуацій повинна бути збережена на резервних носіях. Цілісність даних забезпечується транзакційністю операцій реляційної бази даних та механізмом резервного копіювання.

3.4.2 Вимоги по стандартизації і уніфікації

Система повинна відповідати вимогам ергономіки і зручності користування за умови комплектування високоякісним обладнанням (ЕОМ, монітор і інше обладнання), що має необхідні сертифікати відповідності і безпеки.

3.4.3 Вимоги до функцій (завдань), що виконуються системою:

– забезпечення зручного інтерфейсу для диспетчерів і водіїв.
– розподіл замовлень та керування статусами водіїв.
– автоматичне сповіщення про ключові події перевізного процесу.
– забезпечення високої швидкодії та надійності обробки запитів.
– журналювання системи.
– приймання або відхилення замовлення водієм.
– керування користувачами та їх статусами аміністратором.

4 Вимоги до документації

Документація повинна відповідати вимогам ЄСКД та ДСТУ

Комплект документації повинен складатись з:

– пояснювальної записки;
– графічного матеріалу:
  а) Структурна схема.
  б) Схема електрична принципова.
  в) Макет системи.
  г) Схема електрична функціональна.

5 Стадії та етапи проєктування

Таблиця 1 – Стадії та етапи виконання кваліфікаційної роботи бакалавра

№ етапу	Назва етапу виконання кваліфікаційної роботи бакалавра	Термін виконання
1	*Розробка технічного завдання*	*26.01 – 02.02*
2	*Робота над першим розділом «Аналіз технічного завдання»*	*03.02 – 15.02*
3	*Робота над другим розділом «Проєктна частина»*	*20.04 – 02.05*
4	*Робота над третім розділом «Практична частина»*	*02.05 – 20.05*
5	*Робота над четвертим розділом «Безпека життєдіяльності, основи охорони праці»*	*20.05 –* 25.05
6	*Оформлення пояснювальної записки і графічного матеріалу*	*26.05 – 7.06*
7	*Перевірка на академічний плагіат, перевірка керівником та консультантами*	*8.06 – 14.06*
8	*Попередній захист кваліфікаційної роботи бакалавра*	*15.06 – 21.06*
9	*Захист кваліфікаційної роботи бакалавра*	*26.06*

6 Додаткові умови виконання кваліфікаційної роботи

Під час виконання кваліфікаційної роботи у дане технічне завдання можуть вноситися зміни та доповнення.

Додаток Б

Перелік елементів

Позн.	Найменування	К-ть	Примітка
	Мікронтролер		
U1	ESP32-WROOM-32 (DevKit V1)	1	
	Модулі		
J1	GPS-приймач NEO-6M (GY-GPS6MV2)	1	UART, NMEA 0183
U2	OLED SSD1306, 0,91"	1	I²C
	Світлодіоди		
D1	LED 5 mm, Kingbright WP710A10ID	1	
	Кнопки		
SW1	Кнопка тактова (SW_Push)	1	
	Резистори		
R1	MFR-25FBF52-220R	1	220 Ом
R2	MFR-25FBF52-1K	1	1 кОм
R3	MFR-25FBF52-2K	1	2 кОм

Змн.	Арк.	№ докум.	Підпис	Дата	КС КРБ 123.167.00.00 ПЕ		
Розроб.		Кармазин О.Б.			Комп'ютерна система керування процесами вантажних перевезень із використанням платформи n8n Перелік елементів	Літ.	Арк. 1 / Аркушів 1
Перевір.		Стадник Н.Б.					
Реценз.		Деркач М.В				ТНТУ, каф. КС, гр. СІс-41	
Н. Контр.		Луцик Н.С.					
Затверд.		Осухівська Г.М.					

Додаток В

UML діаграма бази даних

Рисунок В.1 – UML діаграма бази даних