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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «CRM-система для оптимізації логістики з використанням TypeScript»

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

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

_____ Богдан ЧОХЛЕНКО
(підпис)

Виконав: здобувач(ка) вищої освіти групи ТЦР-43

Богдан ЧОХЛЕНКО

Керівник: Вікторія ЖЕБКА
д-р техн. наук, проф.

Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Технологій цифрового розвитку

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Технології цифрового розвитку»

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

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Чохленку Богдану Андрійовичу

1. Тема кваліфікаційної роботи: «CRM-система для оптимізації логістики з використанням TypeScript»
керівник кваліфікаційної роботи д.т.н., професор, завідувач кафедри Технологій цифрового розвитку Вікторія ЖЕБКА затверджені наказом Державного університету інформаційно-комунікаційних технологій від — «20» лютого 2026 року № 51.

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

3. Вхідні дані до роботи:
1. Середовище розробки VisualStudioCode.
2. React, TypeScript.
3. Node.js, Express, Prisma, PostgreSQL.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити).
1. Аналіз предметної області логістики.
2. Аналіз та порівняння існуючих аналогів.
3. Дослідження програмних засобів для розробки додатку.
4. Розробити функціонал створеного додатку.

5. Перелік графічного матеріалу
1. Мета, об’єкт, Предмет.
2. Актуальність.
3. Програми аналоги.
4. Вимоги до додатку.
5. Діаграми UML.
6. Блок схеми.
7. Засоби реалізації.
8. Інтерфейс додатку.
9. Висновки.

6. Дата видачі завдання «_1_» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	20.02.26 – 22.03.26	
2	Аналіз існуючих аналогів	22.03.26 – 23.03.26	
3	Дослідження програмних засобів	23.03.26 – 01.04.26	
4	Моделювання об’єкту проектування	01.04.26 – 15.04.26	
5	Розробка функціоналу за стосунку	15.04.26 – 07.05.26	
6	Вступ, висновки, реферат	07.05.26 – 08.05.26	
7	Розробка презентації за стосунку	08.05.26 – 14.05.26	
8	Попередній захист роботи	18.06.26	

Здобувач(ка) вищої освіти _____ Богдан ЧОХЛЕНКО
(підпис)

Керівник
кваліфікаційної роботи _____ Вікторія ЖЕБКА
(підпис)

РЕФЕРАТ

Текстова частина бакалаврської роботи 62 с., 17 рис., 18 джерело.

Мета роботи – оптимізація логістики за допомогою розробленої веб-орієнтованої CRM-системи, що забезпечує централізоване ведення клієнтської бази, реєстрацію та обробку логістичних замовлень, відстеження статусів доставки і зменшення кількості ручних операцій у щоденній роботі підприємства.

Об'єкт дослідження - процеси управління логістичними операціями підприємства, що включають роботу з клієнтами, замовленнями, маршрутами та контролем доставки.

Предмет дослідження - методи та програмні засоби розробки CRM-системи для оптимізації логістики з використанням TypeScript.

Короткий зміст роботи: В роботі проаналізовано предметну область логістики та розглянуто існуючі CRM-рішення – HubSpot CRM, Pipedrive та Zoho CRM – визначено їх переваги і недоліки стосовно логістичних задач. Сформовано функціональні та нефункціональні вимоги до системи, розроблено технічне завдання та UML-діаграми. Обґрунтовано вибір технологічного стеку: TypeScript, React, Node.js, Express, Prisma ORM, PostgreSQL, Redis, Socket.io. Спроєктовано мультитенантну архітектуру бази даних з ізоляцією даних компаній та розмежуванням доступу за шістьма ролями: ADMIN, MANAGER, DISPATCHER, LOGIST, ACCOUNTANT, DRIVER. Реалізовано модуль авторизації на базі JWT, автоматичний розрахунок вартості замовлення з урахуванням типу виконання та оплати водія, інтеграцію з API Національного банку України для отримання актуальних курсів валют, а також real-time оновлення через Socket.io. Проведено автоматизоване тестування за допомогою Vitest та Supertest – 41 тест, усі проходять успішно.

Сферою використання системи є логістичні підприємства, що займаються організацією та контролем доставок і потребують централізованого інструменту для оперативного управління замовленнями та автопарком.

КЛЮЧОВІ СЛОВА: CRM-СИСТЕМА, ЛОГІСТИКА, TYPESCRIPT, REACT, NODE.JS, EXPRESS, PRISMA, POSTGRESQL, АВТОМАТИЗАЦІЯ, МАРШРУТ, ЗАМОВЛЕННЯ, ДОСТАВКА, ВЕБ-ЗАСТОСУНОК.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ - програмне забезпечення

БД - база даних

API - прикладний програмний інтерфейс

ORM - object-relational mapping (об'єктно-реляційне відображення)

IDE - інтегроване середовище розробки

UML - уніфікована мова моделювання

UI - інтерфейс користувача

UX - досвід користувача

CRM - система управління відносинами з клієнтами

JWT - JSON Web Token (токен авторизації)

REST - representational state transfer (архітектурний стиль API)

HTTP - hypertext transfer protocol

SQL - structured query language (мова структурованих запитів)

ТЗ - технічне завдання

СКВ - система контролю версій

RBAC - role-based access control (керування доступом на основі ролей)

CI/CD - continuous integration / continuous deployment

TTL - time to live (час життя кешу)

ERD - entity-relationship diagram (діаграма сутність-зв'язок)

НБУ - Національний банк України

ВСТУП

У сучасних умовах цифровізації бізнесу ефективне управління логістичними процесами є одним із ключових чинників стабільної роботи підприємства. Логістика охоплює велику кількість взаємопов’язаних операцій, серед яких обробка замовлень, ведення клієнтської бази, координація доставки, контроль статусів виконання заявок, облік маршрутів та взаємодія між менеджерами, клієнтами і виконавцями. Використання розрізнених таблиць, паперових записів або непов’язаних між собою програмних рішень ускладнює контроль над цими процесами, збільшує ймовірність помилок та знижує загальну продуктивність роботи.

Одним із дієвих шляхів вирішення зазначеної проблеми є впровадження CRM-системи, яка забезпечує централізоване зберігання даних, автоматизацію бізнес-процесів та підвищення прозорості логістичних операцій. Така система дає можливість зберігати інформацію про клієнтів, замовлення, маршрути, статуси доставки та історію взаємодії в єдиному інформаційному середовищі. Це дозволяє скоротити час на обробку даних, підвищити якість обслуговування та покращити керованість логістичних процесів.

Актуальність теми полягає в необхідності створення сучасного програмного рішення, яке дозволить оптимізувати логістичну діяльність підприємства за рахунок автоматизації основних операцій та підвищення надійності обробки інформації. Особливу роль у цьому відіграє використання мови програмування TypeScript, яка забезпечує статичну типізацію, покращує контроль структури даних, зменшує кількість помилок під час розробки та спрощує подальшу підтримку програмного забезпечення. Завдяки цьому TypeScript є доцільним вибором для реалізації масштабованих та надійних веб-систем.

Об'єкт дослідження - процеси управління логістичними операціями підприємства, що включають роботу з клієнтами, замовленнями, маршрутами та контролем доставки.

Предмет дослідження - методи та програмні засоби розробки CRM-системи для оптимізації логістики з використанням TypeScript.

Мета роботи – оптимізація логістики за допомогою розробленої веб-орієнтованої CRM-системи, що забезпечує централізоване ведення клієнтської бази, реєстрацію та обробку логістичних замовлень, відстеження статусів доставки і зменшення кількості ручних операцій у щоденній роботі підприємства.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Проаналізувати предметну область логістики та особливості використання CRM-систем у цій сфері.
2. Дослідити існуючі програмні рішення та визначити їх переваги і недоліки.
3. Сформувати функціональні та нефункціональні вимоги до системи.
4. Спроєктувати структуру системи, архітектуру програмного забезпечення та базу даних.
5. Обґрунтувати вибір мови програмування TypeScript та інших технологічних засобів реалізації.
6. Реалізувати основний функціонал CRM-системи.
7. Провести тестування розробленого програмного забезпечення та оцінити результати його роботи.

Наукова новизна роботи полягає у розробці структури CRM-системи, орієнтованої на оптимізацію логістичних процесів підприємства, а також у застосуванні TypeScript як основного засобу забезпечення надійності, типобезпеки та підтримуваності програмного рішення.

Практичне значення одержаних результатів полягає у можливості використання розробленої CRM-системи для автоматизації обробки замовлень, централізованого ведення клієнтської бази, контролю доставки та підвищення ефективності роботи персоналу. Упровадження такої системи сприятиме зменшенню кількості помилок, скороченню часу на виконання типових операцій та покращенню якості управління логістичною діяльністю підприємства.

1. Чохленко Б. А., Жебка В. В. Crm-система для оптимізації логістики з використанням TypeScript // Матеріали Всеукраїнської науково-технічної конференції «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. К. : ДУІКТ, 2026.

2. Чохленко Б. А., Жебка В. В. Crm-система для оптимізації логістики з використанням TypeScript // Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. К. : ДУІКТ, 2026.

1. АНАЛІЗ ТА ХАРАКТЕРИСТИКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Актуальність теми

Логістичний процес охоплює не одну дію, а цілий ланцюг взаємопов’язаних операцій: приймання замовлення, уточнення параметрів доставки, резервування ресурсів, планування маршруту, призначення відповідального працівника, контроль виконання і фіксацію результату. На практиці ці етапи часто розподілені між різними працівниками, тому відсутність єдиної системи призводить до втрати даних або появи суперечностей між записами.

Типовими проблемами логістичних підприємств є дублювання клієнтської інформації, ручне перенесення даних між різними сервісами, складність відстеження поточного статусу заявки та залежність якості обслуговування від окремих співробітників. Якщо менеджер звільняється або тимчасово відсутній, частина інформації про замовлення може залишитися тільки в його особистих нотатках або листуванні. Це створює ризики для стабільної роботи компанії.

CRM-система для логістики повинна вирішувати відразу кілька задач. По-перше, вона має централізувати інформацію про клієнтів та історію роботи з ними. По-друге, система має забезпечувати керування логістичними заявками від моменту створення до завершення. По-третє, важливо надати керівнику можливість контролювати навантаження співробітників, строки виконання і загальну динаміку замовлень.

Доцільність використання TypeScript у межах такої системи пов’язана з необхідністю чітко описувати бізнес-сутності: клієнта, замовлення, маршрут, транспортний засіб, статус, коментар, звіт. Статична типізація дозволяє зафіксувати структуру цих сутностей на рівні коду, що особливо важливо під час розробки масштабованих веб-додатків із клієнтською та серверною частиною.

Отже, тема розробки CRM-системи для оптимізації логістики є актуальною через практичну потребу бізнесу в автоматизації, контролі та прозорості щоденних операцій. Створення такого програмного продукту має не лише теоретичну, а й безпосередню прикладну цінність.

1.2 Огляд аналогів

На ринку існує значна кількість CRM-систем, однак більшість із них створювалась для автоматизації продажів, маркетингу або підтримки клієнтів. Логістика як предметна область має свою специфіку: тут важливі не лише контакти та угоди, а й статуси замовлень, маршрути, терміни виконання та розподіл задач між виконавцями. Саме тому перед тим як розробляти власне рішення, було проаналізовано три поширені CRM-системи - HubSpot CRM, Pipedrive та Zoho CRM - з метою оцінити, наскільки вони підходять для логістичного підприємства.

1. HubSpot CRM – одна з найвідоміших безкоштовних CRM-систем на ринку. Вона пропонує зручний інтерфейс, інструменти для роботи з контактами, автоматизацію email-комунікацій та базову аналітику. Система добре підходить для відділів продажів і маркетингу, де головне завдання - відстежувати взаємодію з потенційними клієнтами та вести угоди по воронці. Проте якщо говорити про логістику, HubSpot суттєво програє: у ній немає вбудованих інструментів для керування замовленнями, відстеження доставок чи роботи з маршрутами. Будь-яка спроба адаптувати систему під логістичний процес потребує додаткових інтеграцій або розробки власних розширень, що фактично нівелює перевагу готового рішення. (рис 1.1)

Рис. 1.1 Інтерфейс HubSpot CRM

Pipedrive – CRM-система, орієнтована на управління угодами через візуальну воронку продажів. Головна ідея продукту полягає в тому, щоб менеджер завжди бачив, на якому етапі знаходиться кожна угода і що потрібно зробити далі. Інтерфейс простий і зрозумілий, налаштування займає небагато часу. Для невеликого бізнесу, де логістика зводиться до кількох замовлень на день, Pipedrive можна частково пристосувати - наприклад, використовувати картки угод як замінник карток замовлень. Але такий підхід має очевидні обмеження: система не підтримує ні статусів доставки, ні маршрутів, ні ролей логіста чи диспетчера. Тобто з ростом обсягів роботи підприємству доведеться або шукати інше рішення, або витрачати ресурси на суттєве доопрацювання. (рис 1.2).

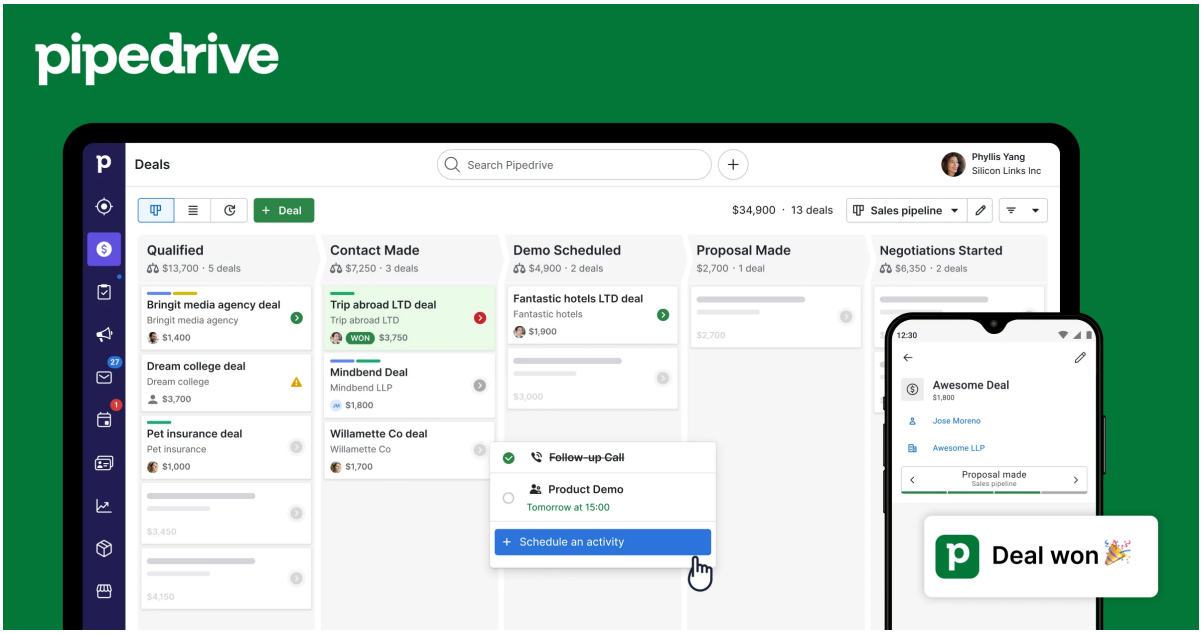


Рис. 1.2 Інтерфейс Pipedrive

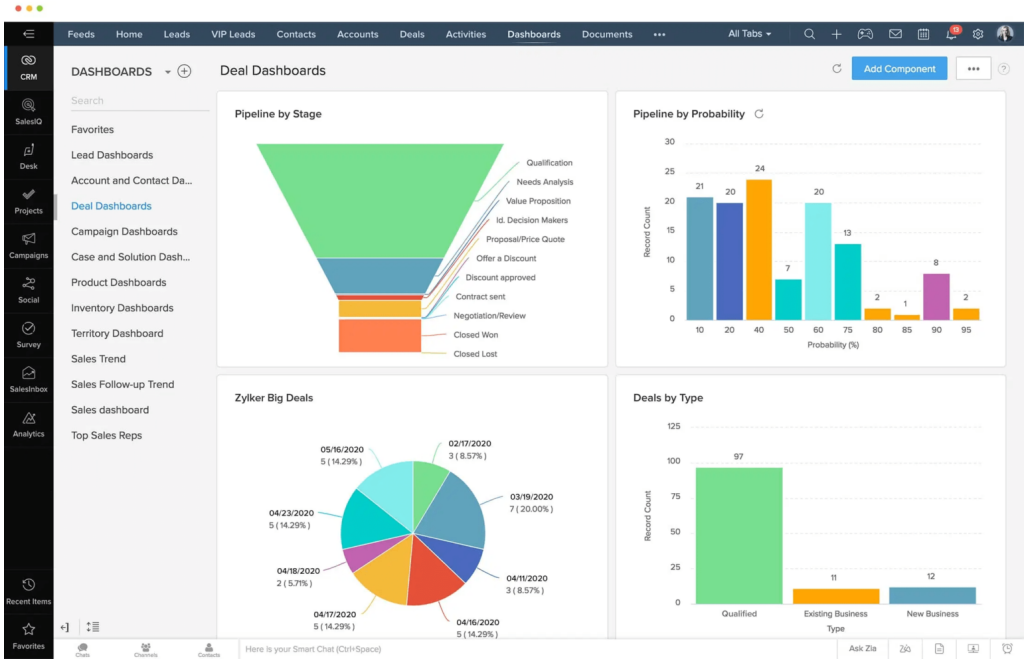


Рис. 1.3 Інтерфейс Zoho CRM

Zoho CRM – найбільш гнучка система серед розглянутих. Має модульну архітектуру, конструктор бізнес-процесів, підтримку автоматизації та широкі можливості кастомізації. Завдяки цьому Zoho CRM теоретично можна налаштувати під логістичні потреби: створити власні поля, сценарії зміни статусів, форми для замовлень. Проте на практиці це потребує значного часу та

технічної експертизи. Частина корисного функціоналу доступна лише в платних тарифах, а складність налаштування може бути невиправданою для невеликої команди. Zoho CRM - найближчий до задач логістики серед аналогів, але все одно не є готовим рішенням без суттєвих доопрацювань. Інтерфейс Zoho CRM зображено на рисунку 1.3.

Для зручності порівняння основні характеристики розглянутих систем зведено до таблиці 1.1.

Таблиця 1.1

Порівняння CRM-систем

Система	Основні переваги	Основні недоліки	Придатність для логістики
HubSpot CRM	Безкоштовний план, простий інтерфейс, зручна робота з контактами та комунікаціями.	Відсутній логістичний функціонал, система орієнтована на продажі та маркетинг.	Підходить лише для ведення клієнтської бази, логістичні задачі не покриває.
Pipedrive	Інтуїтивна воронка угод, легке налаштування, зрозумілий інтерфейс.	Немає модулів для маршрутів і доставок, не підтримує складні операційні процеси.	Можна частково використати для супроводу замовлень, але з суттєвими обмеженнями.
Zoho CRM	Гнучка модульна система, конструктор процесів, підтримка автоматизації.	Складне налаштування, частина функцій лише в платних тарифах.	Найближча до логістичного застосування, але потребує значної кастомізації.

Проведений огляд показує, що жодна з розглянутих систем не закриває потреби логістичного підприємства в готовому вигляді. Усі три були створені з іншою метою і лише частково перетинаються із задачами логістики. Використання будь-якої з них без доопрацювання означатиме або надлишковий

функціонал, яким ніхто не користується, або відсутність ключових інструментів - таких як статуси доставки, маршрути і контроль виконання.

Саме тому в рамках цієї роботи прийнято рішення розробити власну CRM-систему, яка орієнтована конкретно на логістичний процес. Такий підхід дозволяє отримати зрозумілий інтерфейс під конкретні задачі, уникнути залежності від платних тарифів і обмежень сторонніх платформ, а також реалізувати саме ту логіку роботи з замовленнями, яка потрібна підприємству.

1.3 Постановка задач дипломного проектування

Проведений аналіз предметної області та огляд існуючих аналогів показали, що готові CRM-системи не повністю відповідають потребам логістичного підприємства. Більшість із них орієнтована на продажі або загальну автоматизацію бізнесу, а логістичні сценарії - керування маршрутами, контроль статусів доставки, розподіл задач між менеджером і логістом - або відсутні, або потребують значного доопрацювання. Це і стало підставою для розробки власного програмного рішення.

Метою дипломної роботи є розробка веб-орієнтованої CRM-системи, яка забезпечує централізовану роботу з клієнтами та замовленнями, відстеження статусів виконання і зменшення кількості ручних операцій у щоденній роботі логістичного підприємства.

Об'єктом дослідження є процеси управління логістичними операціями підприємства, що включають роботу з клієнтами, замовленнями, маршрутами та контролем доставки.

Предметом дослідження є методи та програмні засоби розробки CRM-системи для оптимізації логістики з використанням TypeScript.

Для досягнення поставленої мети в рамках роботи було вирішено такі завдання:

1. Проаналізовано предметну область логістики та визначено основні проблеми, що виникають при ручному веденні операцій.

2. Розглянуто існуючі CRM-рішення та оцінено їх придатність для задач логістичного підрозділу.

3. Сформовано функціональні та нефункціональні вимоги до системи, визначено ролі користувачів та основні сценарії роботи.

4. Обґрунтовано вибір технологічного стеку: TypeScript, React, Node.js, Express, Prisma, PostgreSQL.

5. Спроєктовано архітектуру системи, модель бази даних та макети інтерфейсу.

6. Реалізовано основні модулі CRM-системи та проведено тестування програмного забезпечення.

Наукова новизна роботи полягає в розробці структури CRM-системи, адаптованої саме до потреб логістичного процесу, де ключовим є не лише облік клієнтів, а й повний цикл виконання замовлення - від прийняття заявки до фіксації результату доставки.

Практична значущість отриманих результатів полягає в тому, що розроблена система може бути використана як готова основа для цифровізації логістичного відділу підприємства. Її впровадження дозволить скоротити час обробки замовлень, зменшити кількість помилок через ручне введення даних і підвищити прозорість роботи персоналу.

2. ВИМОГИ ДО РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Роль технічного завдання у житті проекту

Технічне завдання – це перший і один із найважливіших документів у процесі розробки будь-якого програмного продукту. Його головна мета - зафіксувати, що саме має робити система, для кого вона призначена і який результат очікується на виході. Без чіткого технічного завдання розробка перетворюється на постійні перемовини між замовником і виконавцем, де кожна сторона уявляє кінцевий продукт по-своєму.

За загальними оцінками, підготовка передпроектної документації займає близько 15–20% від загального часу розробки. Приблизно 60% часу витрачається безпосередньо на реалізацію, а решта 20% - на тестування, виправлення помилок і введення продукту в експлуатацію. Якщо на першому етапі заощадити час і не опрацювати вимоги належним чином, ці відсотки перерозподіляться - і більша частина часу піде на переробки вже готових рішень, що значно дорожче.

Для CRM-системи логістичного спрямування технічне завдання має особливе значення, оскільки система охоплює кілька ролей користувачів і різні бізнес-сценарії. Якщо заздалегідь не визначити, які дії доступні менеджеру, а які - логісту чи адміністратору, у ході розробки неминуче виникнуть суперечності в логіці роботи системи. Те саме стосується статусів замовлень, правил зміни етапів і вимог до звітності - усе це краще обговорити до початку написання коду.

Щодо відповідальної особи за формування технічного завдання, то найкраще, якщо це буде людина, яка безпосередньо працюватиме з системою після її запуску – наприклад, керівник логістичного відділу або старший менеджер. Така людина краще за інших розуміє реальні потреби підрозділу і може пояснити, які задачі є пріоритетними, а які - другорядними.

Технічне завдання може оформлюватись як за стандартним шаблоном, так і у довільній формі. Проте уніфікований підхід має переваги: він дозволяє команді розробників швидше зорієнтуватись у задачах, заздалегідь виявити технічні складнощі та спланувати подальшу підтримку продукту. Для навчального проєкту це також важливо, оскільки якість технічного завдання безпосередньо відображає розуміння автором майбутньої системи.

2.2 Моделювання системи та UML-діаграми

До початку реалізації доцільно описати поведінку системи у вигляді моделей. Це дозволяє побачити не окремі екрани чи таблиці, а логіку роботи цілком. Для цієї мети зручно використовувати UML-діаграми, оскільки вони дають змогу формалізувати сценарії використання, ролі користувачів, структуру класів і переходи між станами.

Для CRM-системи логістичного спрямування ключовими є діаграма варіантів використання, діаграма активності та логічна модель сутностей. На діаграмі варіантів використання відображаються ролі адміністратора, менеджера та логіста. Менеджер створює клієнтів і замовлення, логіст працює з маршрутами і статусами доставки, адміністратор керує користувачами та довідниками системи.

Діаграма активності дозволяє описати типовий шлях заявки: створення замовлення, перевірка даних, призначення виконавця, планування доставки, виконання, завершення і фіксація результату. Така модель корисна тим, що допомагає виявити, на яких етапах потрібна валідація або зміна прав доступу

2.2.1 Діаграми UML

Unified Modeling Language (UML) – уніфікована мова моделювання, яка описує об’єкт в стандартизованому синтаксисі, що забезпечує абсолютну

зрозумілість між усіма членами команди розробників та іншими учасниками процесу розробки.

Варіанти використання діаграм можна поділити на три основні категорії:

1. Проектування. Моделювання архітектури проектів – основний шлях використання UML-діаграм. Розробка додатку чи веб-сайту значно прискорюється, якщо описати деталі та шляхи за допомогою діаграм.

2. Реверс-інжиніринг. Зворотня побудова UML-діаграм з існуючого коду програм дозволяє покращити якість його підтримки та уточення нюансів в документації проекту.

3. Документація. UML-діаграми дозволяють пришвидшити написання документації та покращити її читаємість та зрозумілість.

2.3 Формування технічного завдання до CRM-системи

1. Ціль створення додатку:

1.1. Централізація даних про клієнтів, замовлення та маршрути в єдиному середовищі.

1.2. Автоматизація обробки логістичних заявок від моменту створення до завершення доставки.

1.3. Забезпечення контролю за статусами виконання замовлень та навантаженням персоналу.

1.4. Зменшення кількості ручних операцій і ризику втрати або дублювання даних.

2. Основні функціональні вимоги:

2.1. Реєструвати нових клієнтів та редагувати їх контактні дані.

2.2. Створювати логістичні замовлення із зазначенням параметрів доставки, дати та типу послуги.

2.3. Призначати відповідального менеджера або логіста для кожного замовлення.

2.4. Змінювати статус замовлення та зберігати історію всіх змін.

2.5. Планувати маршрути доставки та прив'язувати їх до замовлень.

2.6. Переглядати список активних, завершених і проблемних замовлень із фільтрацією.

2.7. Формувати короткі аналітичні зведення щодо роботи відділу.

2.8. Видаляти або архівувати записи клієнтів і замовлень.

3. Діаграми та блок-схеми:

3.1 Діаграма варіантів використання

Варіант використання (Use Case) - це опис послідовності дій, які виконуються системою з метою отримання конкретного результату для користувача. На діаграмі варіантів використання актори позначаються символом людини, а самі варіанти використання - еліпсами. Між ними встановлюються зв'язки трьох типів: узагальнення, включення та розширення.

Узагальнення позначає спільну поведінку між акторами або варіантами використання. Включення застосовується, коли один варіант використання обов'язково містить інший як частину своєї логіки. Розширення описує необов'язкову поведінку, яка додається до базового варіанту використання за певних умов.

У CRM-системі визначено шість ролей користувачів відповідно до схеми бази даних: адміністратор (ADMIN), менеджер (MANAGER), диспетчер (DISPATCHER), бухгалтер (ACCOUNTANT), логіст (LOGIST) та водій (DRIVER). Кожна роль має свій набір доступних функцій. На основі аналізу предметної області та вимог до системи побудовано діаграму варіантів використання, зображену на рисунку 2.1.

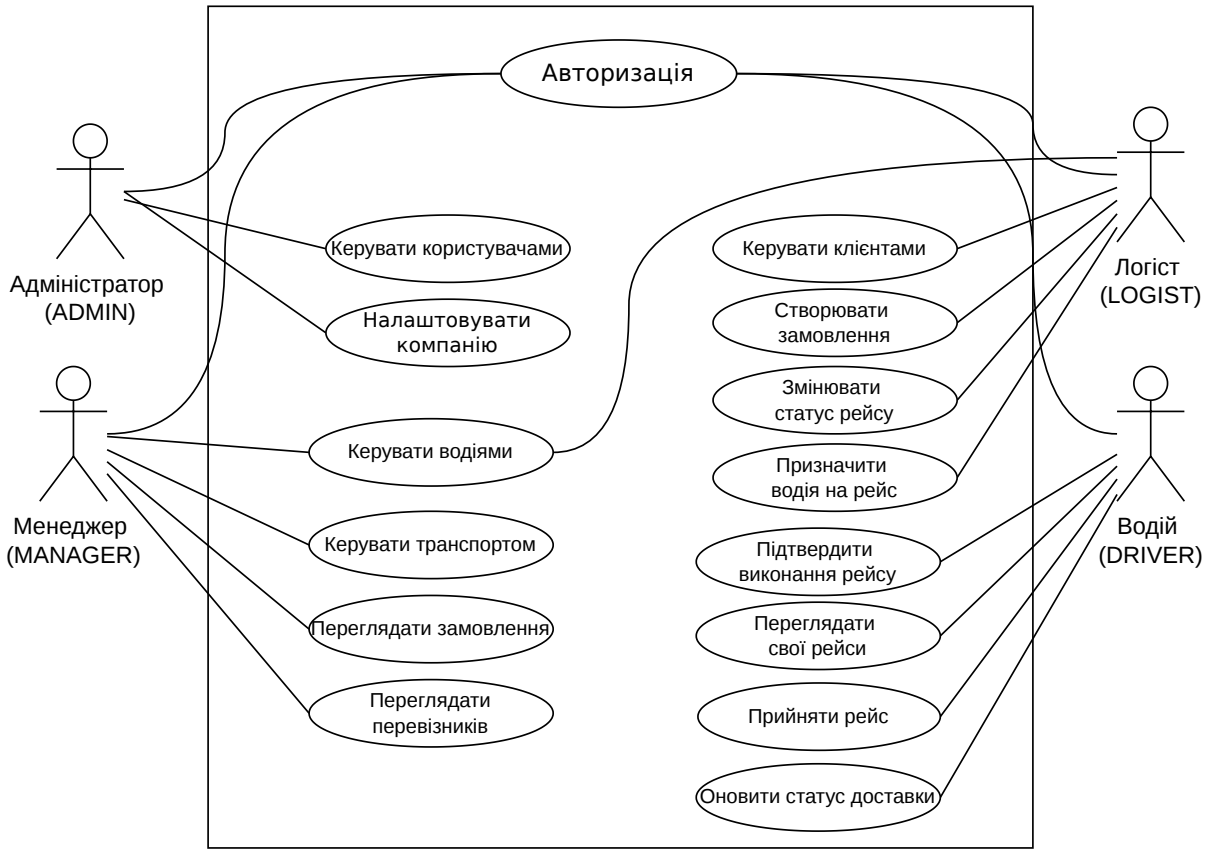


Рис. 2.1 UML діаграма варіантів використання CRM-системи

3.2 Діаграма станів замовлення

Діаграма станів відображає життєвий цикл окремого об'єкта - у даному випадку логістичного замовлення. На відміну від діаграми активності, яка показує загальний потік дій між різними об'єктами, діаграма станів фокусується на переходах між станами одного об'єкта залежно від подій, що відбуваються в системі.

Замовлення в CRM-системі проходить через п'ять станів відповідно до enum OrderStatus у схемі бази даних: NEW (нове), CONFIRMED (підтверджене), IN_TRANSIT (у дорозі), DELIVERED (доставлене), CANCELLED (скасоване). Кожен перехід між станами ініціюється конкретною дією користувача з відповідними правами доступу. Діаграму станів замовлення зображено на рисунку 2.2.

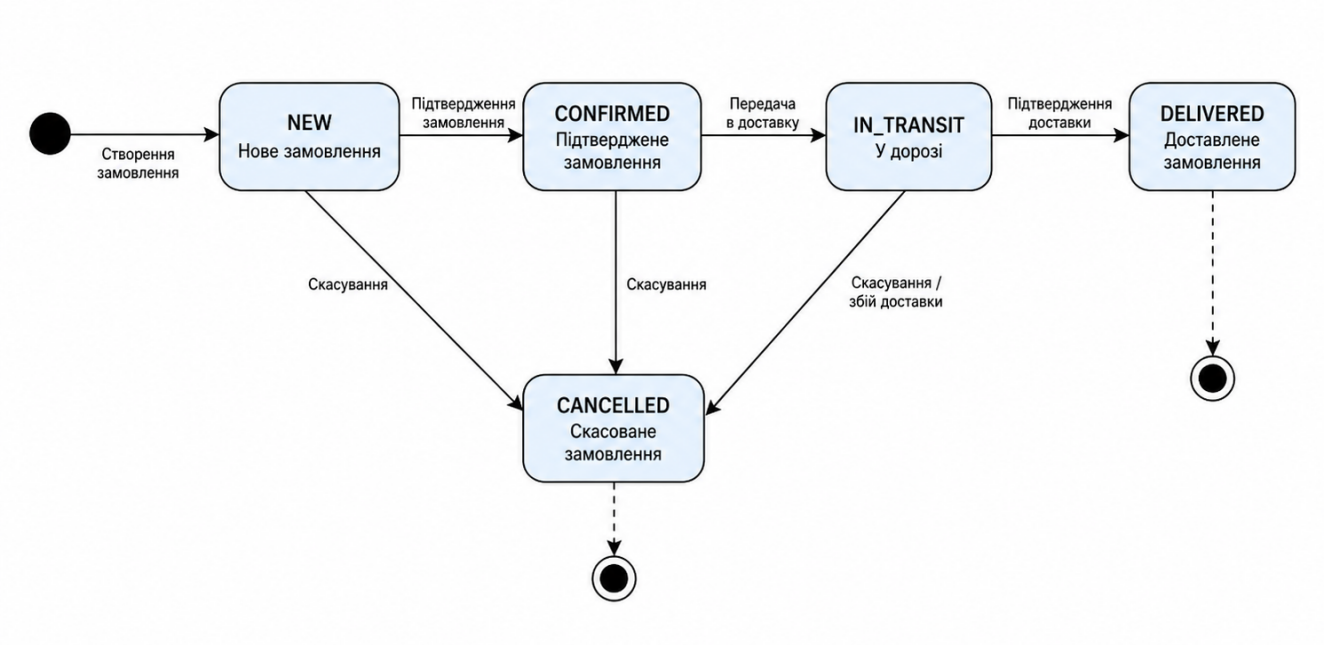


Рис. 2.2 UML діаграма станів замовлення в CRM-системі

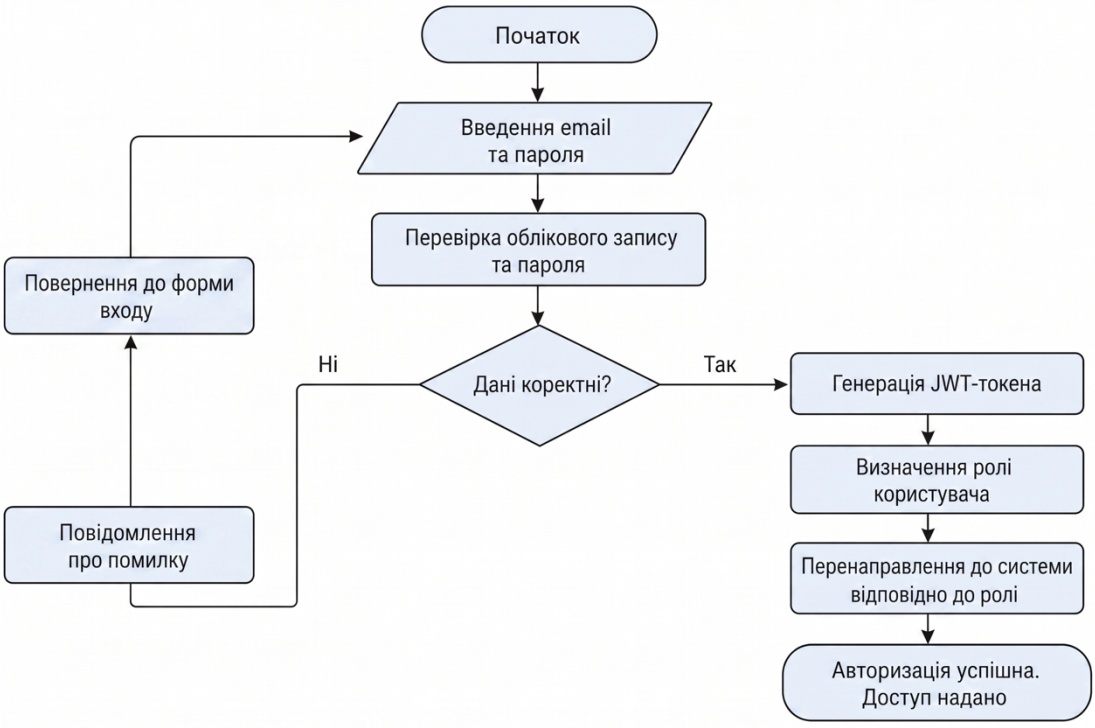


Рис. 2.3 Блок-схема алгоритму авторизації в CRM-системі

3.3 Блок-схема алгоритму авторизації

Для розуміння процесу входу до системи побудовано блок-схему алгоритму авторизації (рис. 2.3):

1. Користувач вводить email та пароль на сторінці входу.
2. Система перевіряє наявність облікового запису з вказаним email та коректність пароля.

2.1. Якщо дані введено правильно - генерується JWT-токен і користувач перенаправляється до системи відповідно до своєї ролі.

2.2. Якщо дані неправильні - виводиться повідомлення про помилку, користувач повертається до форми входу.

3. Користувача авторизовано, доступ надається відповідно до призначеної ролі.

3.4 Блок-схема алгоритму створення замовлення

Для відображення алгоритму реєстрації нового логістичного замовлення побудовано блок-схему (рис. 2.4):

1. Менеджер або диспетчер переходить до форми створення замовлення.
2. Вибирає клієнта зі списку або створює нового.
3. Заповнює дані вантажу, маршруту, дат та типу виконання (INTERNAL або EXTERNAL).
4. Система перевіряє коректність обов'язкових полів.

4.1. Якщо є помилки – відображається повідомлення про необхідність виправлення.

4.2. Якщо дані коректні – переходимо до наступного кроку.

5. Залежно від типу виконання призначається водій і транспорт або сторонній перевізник.
6. Система автоматично розраховує витрати, ціну та маржу.
7. Замовлення зберігається в базі даних зі статусом NEW.

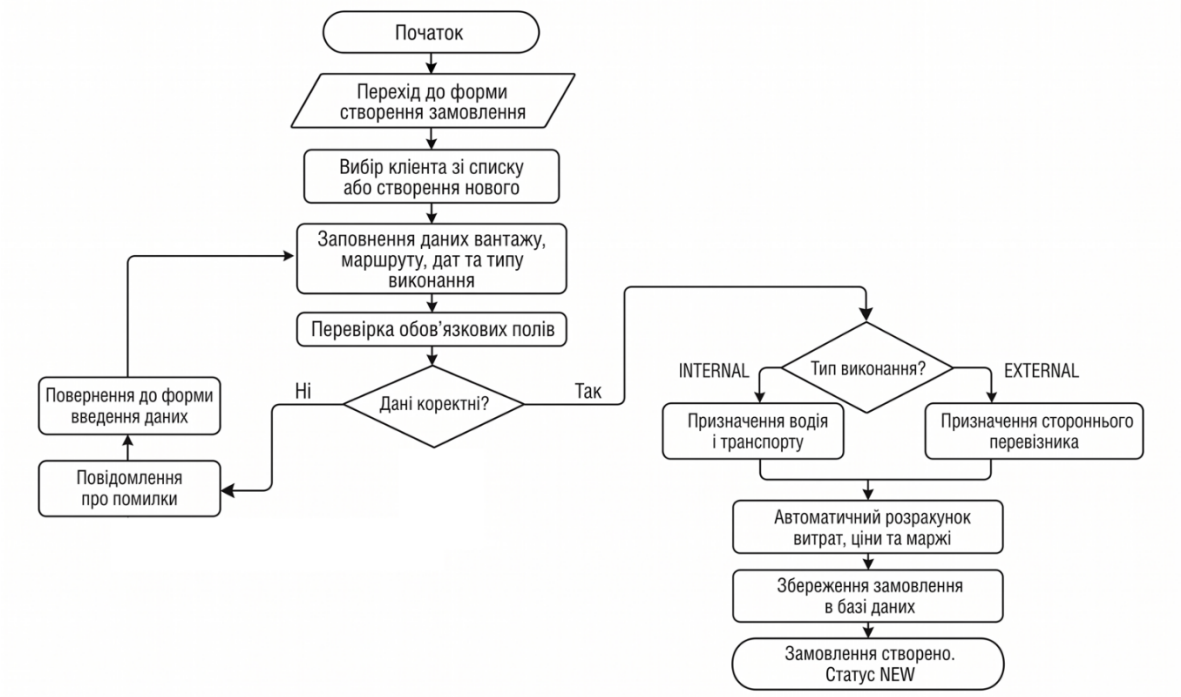


Рис. 2.4 Блок-схема алгоритму створення замовлення

3.5 Блок-схема алгоритму обробки замовлення

Для відображення алгоритму роботи з існуючим замовленням побудовано блок-схему (рис. 2.5):

1. Логіст або диспетчер відкриває список активних замовлень.

2. Чи існує замовлення з потрібним статусом?

2.1. Якщо ні – пошук або фільтрація за параметрами.

2.2. Якщо так – відкривається детальна картка замовлення.

3. Користувач змінює статус замовлення або редагує дані.

4. Система зберігає зміни в базі даних та оновлює фінансові показники.

5. За потреби – замовлення скасовується зі збереженням причини.

4. Опис варіантів використання:

4.1. Неавторизований користувач може лише відкрити сторінку входу - будь-які дані системи недоступні.

4.2. Авторизований користувач входить до системи за email та паролем і отримує доступ відповідно до своєї ролі.

4.3. Менеджер та диспетчер можуть створювати замовлення, обирати тип виконання та призначати ресурси.

4.4. Логіст може переглядати активні замовлення, змінювати їх статуси та додавати примітки.

4.5. Бухгалтер має доступ до фінансових показників замовлень: витрати, маржа, стан оплати.

4.6. Адміністратор керує обліковими записами, ролями користувачів та довідниками системи.

4.7. Водій може переглядати призначені йому замовлення та маршрути.

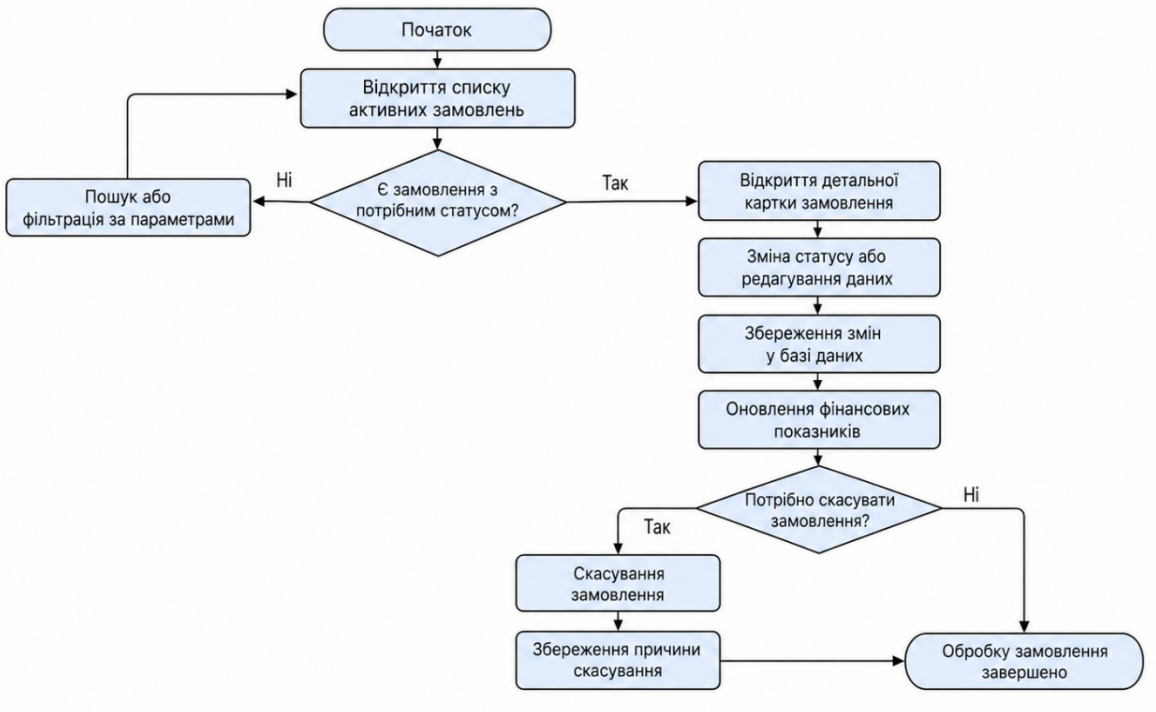


Рис. 2.5. Блок-схема алгоритму обробки замовлення

5. Вимоги до структури даних:

Таблиця 2.1

Сутність «Company»

№	Назва	Тип	Опис
1	name	Рядок	Назва компанії (обов’язкове)
2	email	Рядок	Електронна адреса (необов’язкове)
3	phone	Рядок	Телефон (необов’язкове)
4	address	Рядок	Адреса компанії (необов’язкове)
5	operation_mode	Enum	Режим роботи: OWN_FLEET / BROKER / HYBRID (обов’язкове)

Таблиця 2.2

Сутність «User»

№	Назва	Тип	Опис
1	first_name	Рядок	Ім’я користувача (обов’язкове)
2	last_name	Рядок	Прізвище користувача (обов’язкове)
3	email	Рядок	Email для входу, унікальний (обов’язкове)
4	password	Рядок	Хеш паролю (обов’язкове)
5	ui_language	Enum	Мова інтерфейсу: UA / ENG (обов’язкове)
6	is_active	Boolean	Активний обліковий запис (обов’язкове)

Таблиця 2.3

Сутність «Order»

№	Назва	Тип	Опис
1	order_number	Рядок	Унікальний номер замовлення (обов’язкове)
2	client_id	UUID	Посилання на клієнта (обов’язкове)
3	pickup_address	Рядок	Адреса завантаження (обов’язкове)
4	delivery_address	Рядок	Адреса доставки (обов’язкове)
5	pickup_date	Дата і час	Дата завантаження (обов’язкове)
6	execution_type	Enum	Тип виконання: INTERNAL / EXTERNAL (обов’язкове)
7	status	Enum	Статус замовлення (обов’язкове)
8	client_price	Число	Ціна для клієнта (обов’язкове)
9	total_cost	Число	Загальні витрати (обов’язкове)
10	margin	Число	Прибуток (обов’язкове)
11	margin_percent	Число	Відсоток маржі (обов’язкове)

Таблиця 2.4

Сутність «Driver»

№	Назва	Тип	Опис
1	first_name	Рядок	Ім'я водія (обов'язкове)
2	last_name	Рядок	Прізвище водія (обов'язкове)
3	phone	Рядок	Телефон (обов'язкове)
4	license_number	Рядок	Номер водійського посвідчення (обов'язкове)
5	pay_type	Enum	Тип оплати: PER_KM / PER_HOUR / PER_DAY / FIXED (обов'язкове)
6	pay_rate	Число	Ставка оплати (необов'язкове)
7	is_available	Boolean	Доступність водія (обов'язкове)

Таблиця 2.5

Сутність «Vehicle»

№	Назва	Тип	Опис
1	plate_number	Рядок	Номерний знак (обов'язкове)
2	type	Enum	Тип: TRUCK / VAN / CAR / REFRIGERATOR (обов'язкове)
3	capacity	Число	Вантажопідйомність у тоннах (обов'язкове)
4	fuel_type	Enum	Тип палива (необов'язкове)
5	fuel_consumption	Число	Витрата палива л/100км (необов'язкове)
6	cost_per_km	Число	Вартість км у гривнях (необов'язкове)
7	is_available	Boolean	Доступність транспорту (обов'язкове)

Таблиця 2.6

Сутність «Carrier»

№	Назва	Тип	Опис
1	company_name	Рядок	Назва компанії-перевізника (обов'язкове)
2	contact_person	Рядок	Контактна особа (обов'язкове)
3	phone	Рядок	Телефон (обов'язкове)
4	vehicle_types	Enum[]	Типи транспорту перевізника (необов'язкове)
5	max_capacity	Число	Максимальна вантажопідйомність (обов'язкове)
6	coverage_areas	Рядок[]	Зони покриття маршрутів (необов'язкове)
7	rating	Число	Рейтинг перевізника (обов'язкове)

6. Нефункціональні вимоги:

6.1. Інтерфейс користувача

6.1.1. Система підтримує два варіанти мови інтерфейсу: українська та англійська - обираються в налаштуваннях облікового запису.

6.1.2. Система коректно відображається у браузерах Chrome, Firefox та Edge актуальних версій.

6.1.3. Мінімальна підтримувана роздільна здатність екрана - 1280×720 пікселів.

6.2. Розгортання та середовище

6.2.1. Система розгортається як веб-застосунок і доступна через браузер без встановлення додаткового програмного забезпечення.

6.2.2. Серверна частина працює на Node.js з фреймворком Express, база даних - PostgreSQL, ORM - Prisma.

6.3. Вимоги до безпеки

6.3.1. Система не надає доступ до даних неавторизованим користувачам.

6.3.2. Автентифікація реалізована через JWT-токен, який передається в заголовку кожного запиту до захищених маршрутів API.

6.3.3. Права доступу розмежовуються відповідно до ролі користувача: ADMIN, MANAGER, DISPATCHER, ACCOUNTANT, LOGIST, DRIVER.

6.3.4. Дані різних компаній в системі повністю ізольовані одна від одної через прив'язку всіх записів до company_id.

2.4 Вибір мови програмування TypeScript

Перед початком розробки було розглянуто кілька мов програмування, які активно використовуються у сучасній веб-розробці. Основними кандидатами стали JavaScript, TypeScript та Python, оскільки саме вони найчастіше застосовуються для побудови повностекових веб-застосунків.

JavaScript – базова мова веб-розробки, яка підтримується всіма браузерами і має величезну екосистему. Її головний недолік - динамічна типізація. У невеликих проєктах це не критично, але коли система містить десятки взаємопов'язаних сутностей, відсутність явних типів призводить до помилок, які важко відловити до запуску програми.

Python широко використовується у backend-розробці разом із фреймворками Django або FastAPI. Мова має зрозумілий синтаксис і низький поріг входу. Проте використання Python на сервері при React на клієнті означає роботу з двома різними мовами в одному проєкті - це ускладнює обмін типами між частинами системи та збільшує кількість коду, який потрібно підтримувати.

TypeScript – це надбудова над JavaScript, яка додає статичну типізацію. Код TypeScript компілюється у звичайний JavaScript і виконується як у браузері, так і в

Node.js на сервері. Завдяки цьому одна мова використовується на всіх рівнях застосунку - від клієнтського інтерфейсу до серверної логіки і схеми бази даних.

Для наочного порівняння основні характеристики розглянутих мов зведено до таблиці 2.7.

Таблиця 2.7

Порівняння мов програмування

Критерій	JavaScript	Python	TypeScript
Типізація	Динамічна	Динамічна	Статична
Використання на клієнті	Так	Ні	Так
Використання на сервері	Так (Node.js)	Так	Так (Node.js)
Підтримка в VS Code	Висока	Висока	Відмінна
Виявлення помилок	Під час виконання	Під час виконання	На етапі написання коду
Підходить для великих проєктів	Частково	Частково	Так

Порівняння показує, що TypeScript є найбільш доцільним вибором для цього проєкту. Конкретні переваги, які вплинули на рішення:

1. Статична типізація – структура кожної сутності (Order, Client, Driver, Vehicle, Carrier) описується явно. Якщо в замовленні поле execution_type приймає лише значення INTERNAL або EXTERNAL, компілятор не дозволить передати туди щось інше - помилка виявляється ще до запуску.

2. Єдина мова на клієнті і сервері – React на фронтенді і Node.js + Express на бекенді написані однією мовою. Типи моделей даних описуються один раз і використовуються скрізь.

3. Інтеграція з Prisma – ORM генерує типізований клієнт на основі схеми бази даних. Будь-яка зміна в схемі одразу відображається в TypeScript-типах, що унеможливлює розбіжність між моделлю даних і кодом.

4. Підтримка у Visual Studio Code – автодоповнення, перевірка типів у реальному часі та навігація по коду значно прискорюють розробку.

Таким чином, TypeScript обрано як основну мову програмування для реалізації CRM-системи, оскільки він поєднує зручність JavaScript-екосистеми з надійністю статичної типізації, що є критично важливим для системи з великою кількістю взаємопов'язаних бізнес-сутностей

2.5 Вибір середовища розробки та серверного стеку

Середовищем розробки для цього проєкту обрано Visual Studio Code - безкоштовний редактор коду від Microsoft з вбудованою підтримкою TypeScript, інтегрованим терміналом, Git-клієнтом та великим вибором розширень. Для TypeScript-проєктів VS Code є фактичним стандартом: автодоповнення, перевірка типів у реальному часі та навігація по коду дозволяють працювати продуктивно без додаткових налаштувань. Інтерфейс Visual Studio Code зображено на рисунку 2.6.

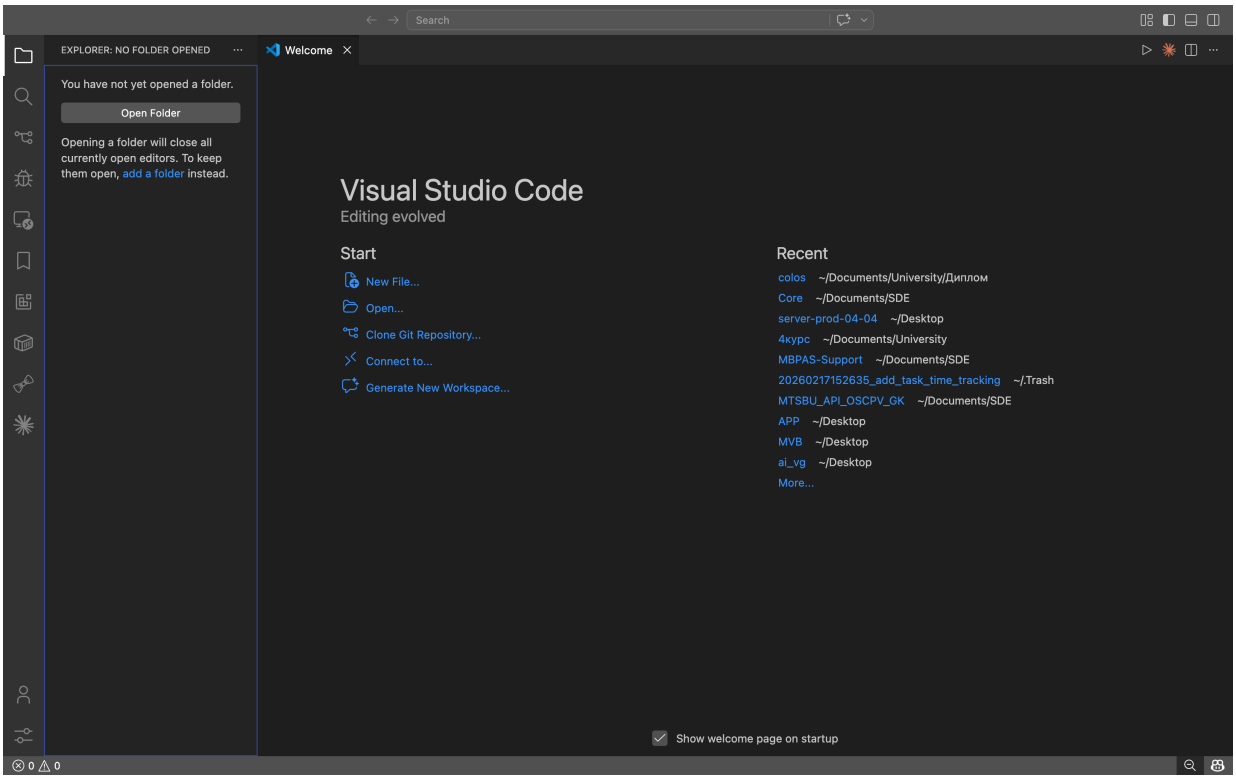


Рис. 2.6 Інтерфейс середовища розробки Visual Studio Code

Проєкт організовано як монорепозиторій colos/ з двома окремими модулями: server/ - серверна частина та client/ - клієнтська частина. Такий підхід дозволяє зберігати спільні типи та конфігурації в одному місці і спрощує розгортання через Docker Compose та CI/CD-пайплайн на базі GitHub Actions.

Серверна частина реалізована на Node.js з TypeScript. Node.js використовує асинхронну, подійно-орієнтовану модель обробки запитів на рушії V8, що дозволяє ефективно обслуговувати багато одночасних підключень без блокування потоку виконання. Це важливо для системи, де кілька користувачів з різними ролями працюють з даними паралельно.

Для побудови REST API використовується фреймворк Express 4.19. Він забезпечує систему роутерів та middleware для обробки запитів, авторизації та обробки помилок. Валідація вхідних даних виконується за допомогою бібліотеки Zod 4.3 - схеми валідації описуються окремо для кожного маршруту і перевіряють коректність даних не лише на рівні TypeScript-компілятора, а й безпосередньо під час виконання запиту.

Для роботи з базою даних використовується Prisma ORM 5.22, який генерує типізований клієнт на основі схеми моделі даних. Будь-яка зміна в структурі бази через міграції одразу відображається в TypeScript-типах, що унеможливлює розбіжність між моделлю і кодом. Як СУБД обрано PostgreSQL 16 - вона підтримує складні запити, транзакції, масиви як тип поля та індексацію, що активно використовується в схемі проєкту.

Для підвищення продуктивності в систему інтегровано Redis як кеш-сховище. Статистика дашборду та геолокація водіїв зберігаються в кеші з TTL 5 хвилин, що суттєво зменшує навантаження на базу даних при активній роботі кількох користувачів одночасно.

Авторизація реалізована через JWT-токени у поєднанні з bcryptjs для хешування паролів. Доступ до маршрутів API розмежовується за системою ролей RBAC: ADMIN, MANAGER, LOGIST, DRIVER, DISPATCHER, ACCOUNTANT.

Додаткову безпеку забезпечують Helmet та express-rate-limit, які захищають API від типових веб-атак та обмежують кількість запитів з однієї IP-адреси.

Для підтримки режиму реального часу використовується Socket.io 4.8 - він забезпечує live-оновлення статусів замовлень та передачу геолокації водіїв без постійного опитування сервера з боку клієнта.

Зовнішні інтеграції включають API Національного банку України для отримання актуальних курсів валют та API цін на паливо, що використовуються для автоматичного розрахунку витрат на замовлення.

Повний технологічний стек серверної частини наведено у таблиці 2.8.

Таблиця 2.8

Технологічний стек серверної частини

Категорія	Технологія	Версія
Runtime	Node.js + TypeScript	TS 5.9
Framework	Express	4.19
База даних	PostgreSQL + Prisma ORM	Prisma 5.22
Кеш	Redis (ioredis)	5.10
Авторизація	JWT + bcryptjs	-
Валідація	Zod	4.3
Real-time	Socket.io	4.8
PDF	pdf-lib	1.17
Email	Nodemailer	8.0
Безпека	Helmet + express-rate-limit	-
Тести	Vitest + Supertest	-

2.6 Засоби створення користувацького інтерфейсу

Після вибору мови програмування та серверного стеку необхідно визначити інструменти для побудови клієнтської частини системи. Оскільки CRM-система реалізується як веб-застосунок, основним питанням є вибір UI-бібліотеки та підходу до організації інтерфейсу.

На сьогоднішній день для побудови інтерфейсів веб-застосунків найбільш поширеними є три рішення: React, Vue.js та Angular. Кожне з них має свою філософію та підходить для різних типів проєктів.

Vue.js – прогресивний фреймворк із простим синтаксисом та низьким порогом входу. Добре підходить для невеликих застосунків, але при масштабуванні потребує додаткових інструментів для керування станом та типізацією.

Angular – повноцінний фреймворк від Google із вбудованою підтримкою TypeScript, dependency injection та модульною архітектурою. Є потужним інструментом для великих корпоративних застосунків, але має значний поріг входу та велику кількість абстракцій, які є надлишковими для проєктів середнього масштабу.

React – бібліотека від Meta для побудови інтерфейсів на основі компонентів. На відміну від Angular, React не нав'язує жорсткої структури і дає розробнику свободу у виборі додаткових інструментів. Компонентний підхід добре підходить для CRM-системи, де одні й ті самі елементи - таблиці, форми, модальні вікна, фільтри, статусні мітки - використовуються в різних модулях. Написавши компонент один раз, його можна перевикористовувати скрізь без дублювання коду.

Таблиця 2.9

Порівняння інструментів для побудови UI

Критерій	React	Vue.js	Angular
Поріг входу	Середній	Низький	Високий
Підтримка TypeScript	Відмінна	Хороша	Вбудована
Масштабованість	Висока	Середня	Висока
Розмір екосистеми	Дуже велика	Середня	Велика
Гнучкість архітектури	Висока	Середня	Низька
Підходить для CRM	Так	Частково	Так

Для збірки клієнтської частини використовується Vite 7 – сучасний інструмент, який забезпечує майже миттєвий старт у режимі розробки завдяки нативній підтримці ES-модулів. Порівняно зі старішим Webpack, Vite суттєво швидший як при старті, так і при оновленні сторінки після змін у коді.

Порівняння розглянутих підходів наведено у таблиці 2.9.

Окрім вибору основної бібліотеки, для повноцінного інтерфейсу CRM-системи використовується ряд додаткових інструментів:

Mapbox GL JS 3.18 – відображення карт та маршрутів доставки. Дозволяє в реальному часі показувати геолокацію водіїв та візуалізувати маршрути замовлень прямо в інтерфейсі.

Recharts 3.8 – побудова графіків та аналітичних діаграм. Використовується для відображення статистики замовлень, фінансових показників та навантаження персоналу на дашборді.

Socket.io-client 4.8 – забезпечує підключення до сервера в режимі реального часу. Завдяки цьому менеджер бачить оновлення статусів замовлень одразу після їх зміни, без необхідності оновлювати сторінку.

HugeIcons React – бібліотека іконок для оформлення інтерфейсу.

Таким чином, React 19 у поєднанні з Vite та набором спеціалізованих бібліотек є обґрунтованим вибором для реалізації клієнтської частини системи, оскільки забезпечує необхідну гнучкість, продуктивність і можливість швидкого розширення функціоналу.

2.7 Система керування базою даних

Як систему керування базами даних обрано PostgreSQL 16. Вона підтримує реляційну модель даних, цілісність зв’язків через зовнішні ключі, індексацію, транзакції та масиви як тип поля – що активно використовується в схемі проєкту, наприклад для зберігання зон покриття перевізника (coverage_areas) та підтримуваних типів транспорту (vehicle_types). Для роботи з PostgreSQL

використовується Prisma ORM, який описує всю модель даних у вигляді єдиної схеми та генерує типізований клієнт для TypeScript.

Модель даних проєкту побудована навколо концепції мультитенантності – кожна компанія працює у власному ізольованому просторі. Всі основні сутності містять поле company_id, яке прив'язує запис до конкретної компанії та унеможливлює доступ до чужих даних. Основні сутності та їх ключові поля зведено до таблиці 2.10.

Таблиця 2.10

Основні сутності бази даних

Сутність	Призначення	Ключові поля
Company	Компанія – власник даних	id, name, operation_mode
User	Обліковий запис працівника	id, email, password, company_id
UserRole	Роль користувача в системі	id, user_id, role
Client	Картка клієнта-замовника	id, company_name, contact_person, phone
Order	Логістичне замовлення	id, order_number, client_id, execution_type, status
Driver	Картка водія	id, first_name, last_name, license_number, pay_type
Vehicle	Транспортний засіб	id, plate_number, type, capacity, fuel_consumption
Carrier	Сторонній перевізник	id, company_name, rating, coverage_areas
ExchangeRate	Курси валют	id, currency, rate, fetched_at
FuelPrice	Ціни на паливо	id, fuel_type, price, fetched_at

Ключовою особливістю моделі є поле execution_type в сутності Order. Воно визначає, яким чином виконується замовлення - власними силами або через стороннього перевізника. Залежно від цього значення система використовує різні

набори полів: для INTERNAL - водій, транспортний засіб та розрахункові витрати на паливо і зарплату; для EXTERNAL - перевізник, узгоджена ціна та статус оплати. Така архітектура дозволяє в одній таблиці підтримувати обидві бізнес-моделі без дублювання структури.

Зв'язки між сутностями забезпечуються через зовнішні ключі з каскадним видаленням (onDelete: Cascade) для записів, які належать компанії. Це гарантує, що при видаленні компанії всі пов'язані дані також будуть видалені, без залишення «висячих» записів у базі.

2.8 Система контролю версій Git та сервіс GitHub

Система контролю версій – це інструмент для відстеження змін у коді. Без неї командна робота над одним проєктом перетворюється на постійний хаос: хтось переписав файл, хтось забув надіслати оновлення, і ніхто не знає, яка версія актуальна.

СКВ бувають трьох типів. Локальні - зберігаються тільки на одному комп'ютері, жодної спільної роботи. З централізованим сховищем - є один сервер, всі підключаються до нього; зручно, поки є інтернет, але якщо сервер ліг - нікому нічого не доступно. З розподіленим сховищем - кожен учасник має повну копію репозиторію у себе. Git працює саме так.

Git розробив Лінус Торвальдс у 2005 році для роботи над ядром Linux - завдання, де сотні людей одночасно правлять один і той самий код. Звідси і його головні переваги: він швидкий, безкоштовний, добре тримає гілки і рідко губить дані. В цьому проєкті Git використовується для ведення монорепозиторію colos/ - весь код серверної та клієнтської частини в одному місці, з окремою гілкою на кожну нову функцію.

GitHub - це хостинг для Git-репозиторіїв. Він додає те, чого в самому Git немає: pull request-и, code review, налаштування прав доступу.

3. РОЗРОБКА ТА ТЕСТУВАННЯ CPM-СИСТЕМИ

3.1 Реалізація інтерфейсу системи

Інтерфейс CRM-системи Colos реалізований як веб-застосунок на базі React 19 з TypeScript. Основним пріоритетом при розробці була зручність щоденної роботи - менеджер, логіст або диспетчер мають швидко знаходити потрібну інформацію і виконувати операції без зайвих кроків. Тому інтерфейс побудований за принципом мінімалізму: темна бічна панель навігації, світлий робочий простір і чіткі акцентні кольори для статусів та дій.

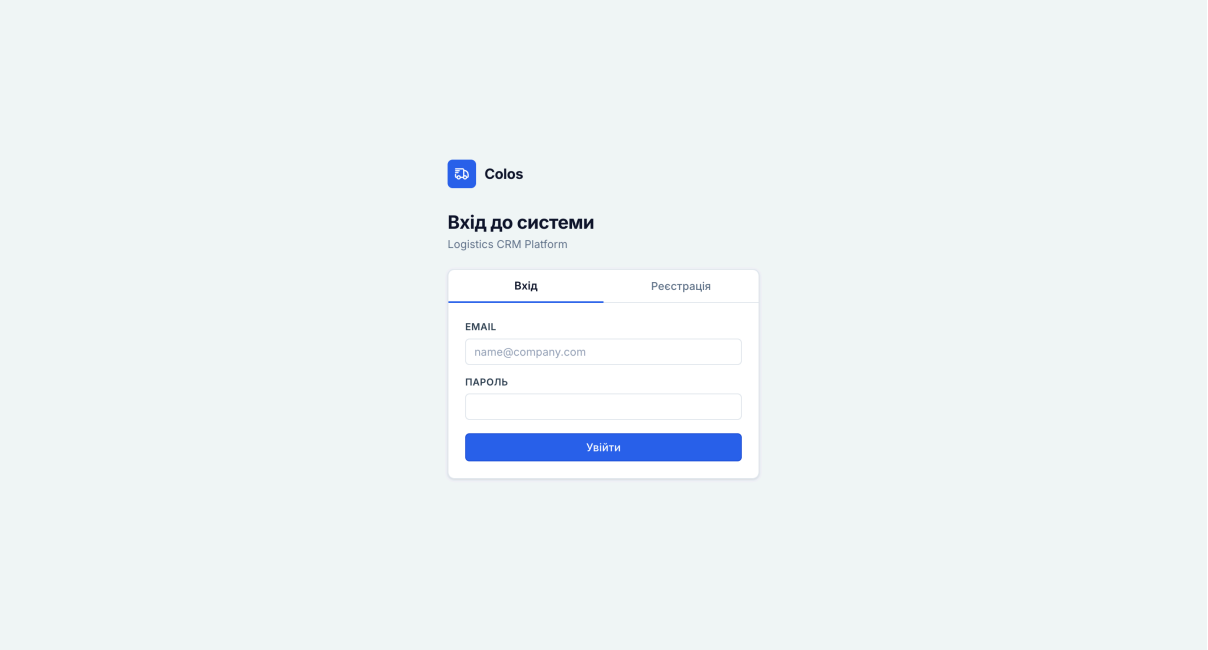


Рис. 3.1 Сторінка авторизації CRM-системи Colos

Навігаційна панель розділена на дві секції. Перша - MAIN - містить основні робочі модулі: Дашборд, Замовлення, Клієнти, Перевізники, Співробітники. Друга - FLEET MANAGEMENT - об'єднує інструменти керування автопарком: Водії та Транспорт. У верхній частині кожної сторінки відображається актуальний

курс USD та EUR, отриманий з API НБУ, роль поточного користувача та кнопка повідомлень.

Сторінка авторизації є першою точкою входу до системи. Вона містить форму з полями email та пароль, кнопку входу і вкладку реєстрації. Дизайн навмисно мінімалістичний - жодних зайвих елементів, лише логотип системи Colos та форма. Сторінку авторизації зображено на рисунку 3.1.

Дашборд є головним екраном системи після входу. Він відображає ключові операційні показники у вигляді карток: активні перевезення зі статусом SLA, кількість прострочених рейсів, показник OTD (On-Time Delivery), середній час доставки, завантаженість автопарку, загальна дебіторська заборгованість та маржинальність за обраний період. Період можна змінювати між Today, Week та Custom. Нижня частина дашборду містить операційну панель у форматі канбан із розподілом замовлень за статусами: NEW, CONFIRMED, IN_TRANSIT, DELIVERED, CANCELLED. Окремий блок відображає ризики та інциденти - SLA-порушення, затримки понад 4 години та замовлення з негативною маржею. Дашборд системи зображено на рисунку 3.2.

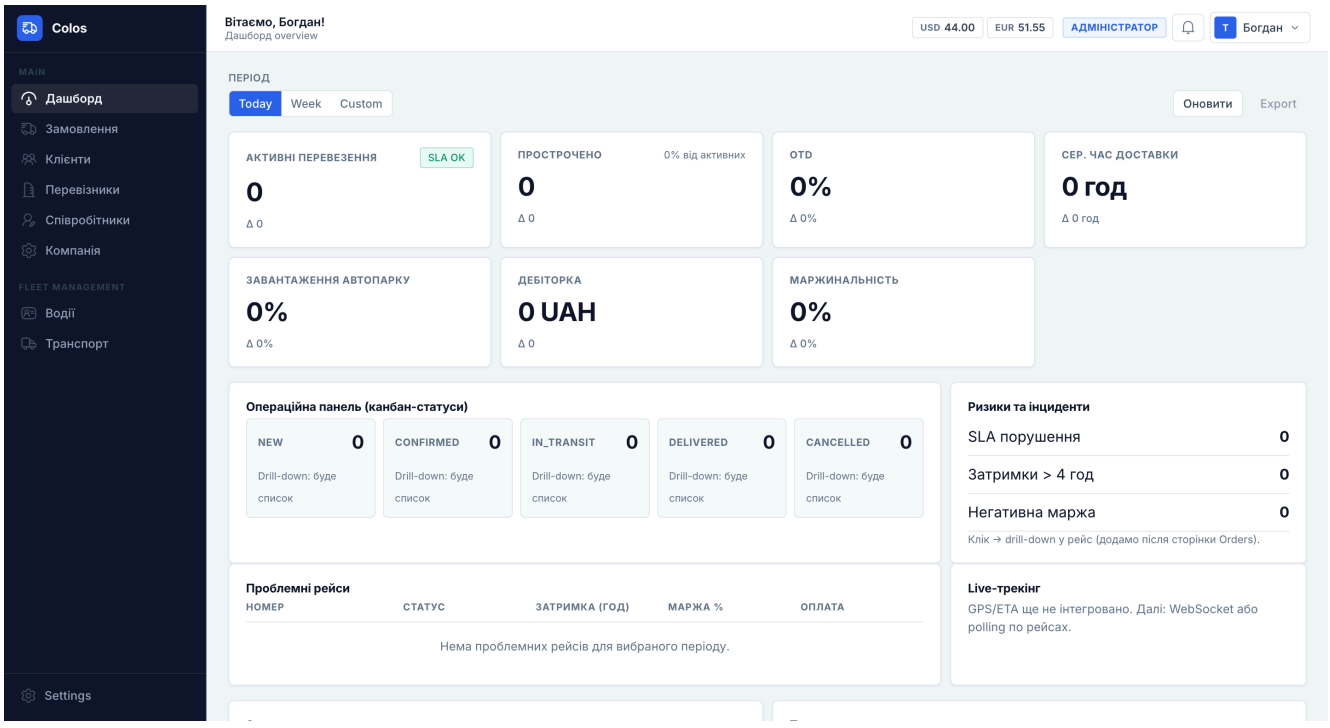


Рис. 3.2 Дашборд CRM-системи Colos

Сторінка замовлень відображає повний список логістичних замовлень у табличному форматі. Для кожного рядка видно номер замовлення, клієнта, статус, тип виконання, дату забору, ціну для клієнта, маржу у гривнях та відсотках, а також статус оплати. Статуси відображаються кольоровими мітками - зелений для завершених, синій для доставлених. Над таблицею розташовані поле пошуку та два фільтри: за статусом і за типом виконання. Кнопка «+ Нове» відкриває форму створення нового замовлення. Сторінку списку замовлень зображено на рисунку 3.3.

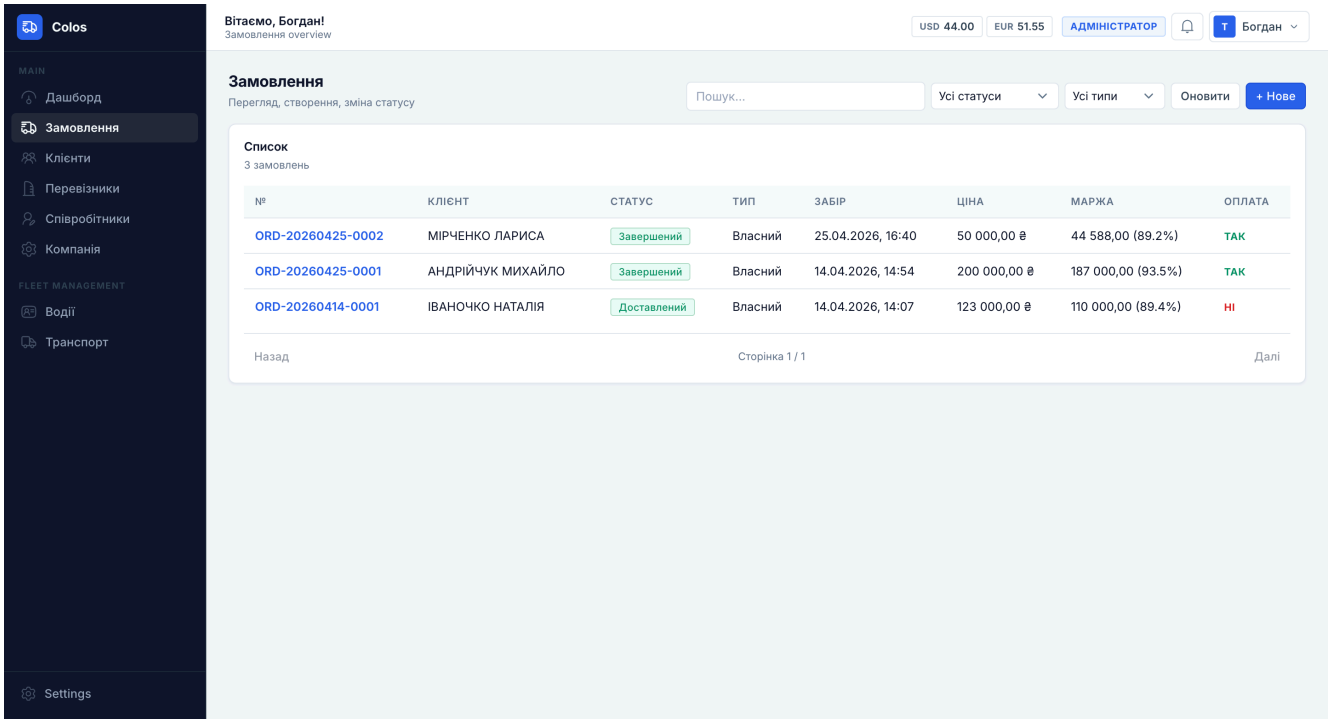


Рис. 3.3 Сторінка списку замовлень

Форма створення замовлення побудована як покроковий wizard із трьома логічними блоками. Перший блок - «Клієнт та маршрут» - дозволяє обрати існуючого клієнта або створити нового, вказати адресу забору та доставки, дату забору та планову дату доставки. Праворуч від форми розташована карта Mapbox, яка автоматично оновлюється після введення адрес і відображає маршрут між точками. Другий блок - «Вантаж» - містить поля для типу вантажу, кількості, одиниці виміру та маси. Третій блок - «Тип виконання» - дозволяє обрати між власним автопарком та стороннім перевізником, після чого відображаються

відповідні поля для призначення водія і транспорту або перевізника. Форму створення замовлення зображено на рисунку 3.4.

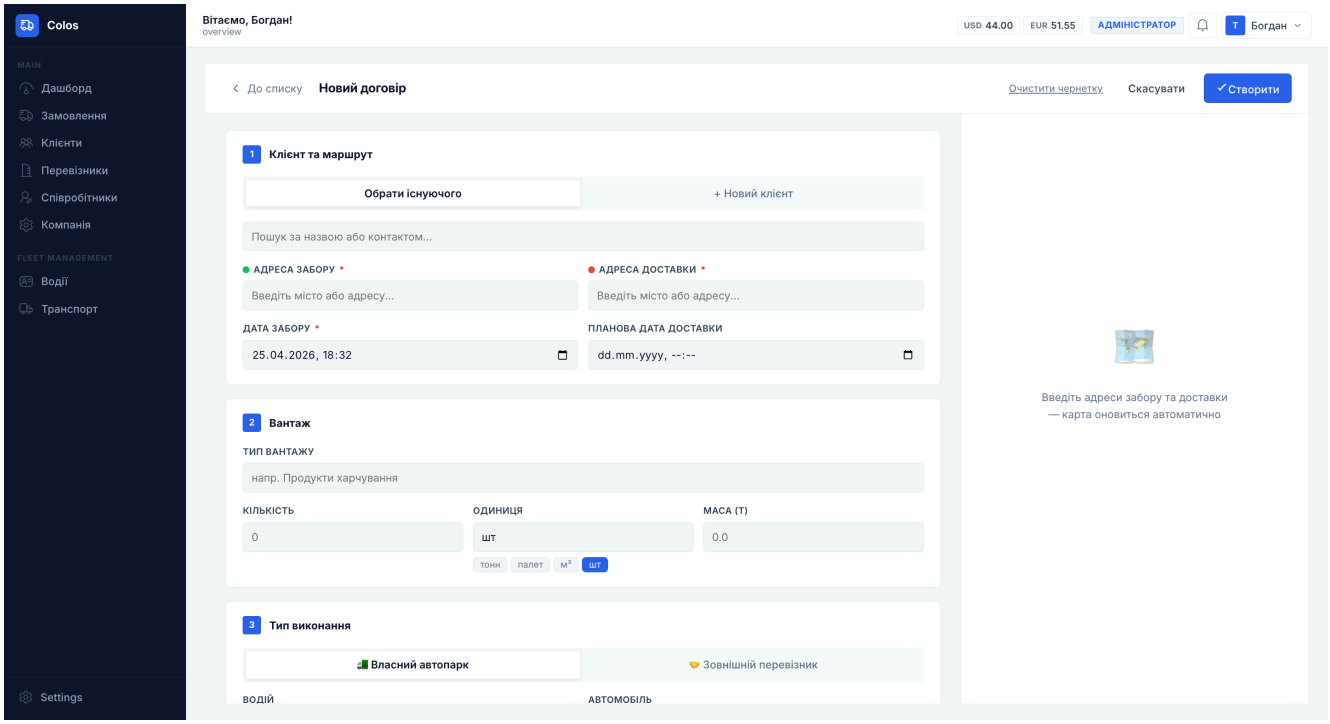


Рис 3.4 Форма створення нового замовлення

Сторінка водіїв відображає список усіх водіїв компанії з прізвищем, ім'ям, номером телефону, номером ліцензії та поточною доступністю. Доступність відображається кольоровою міткою: зелена - «Доступний», червона - «Зайнятий». Праворуч від таблиці розташована панель деталей, яка заповнюється при кліку на рядок. Над списком є поле пошуку, фільтр за доступністю та кнопка додавання нового водія. Сторінку водіїв зображено на рисунку 3.5.

Сторінка транспорту організована аналогічно до сторінки водіїв - таблиця зліва та панель деталей справа. У таблиці відображаються номерний знак, тип транспортного засобу (Вантажівка, Фургон, Легковий, Рефрижератор), вантажопідйомність у тоннах та статус доступності. Фільтрація доступна за типом транспорту та доступністю. Реальні дані показують автопарк із 20 одиниць техніки різних типів. Сторінку транспорту зображено на рисунку 3.6.

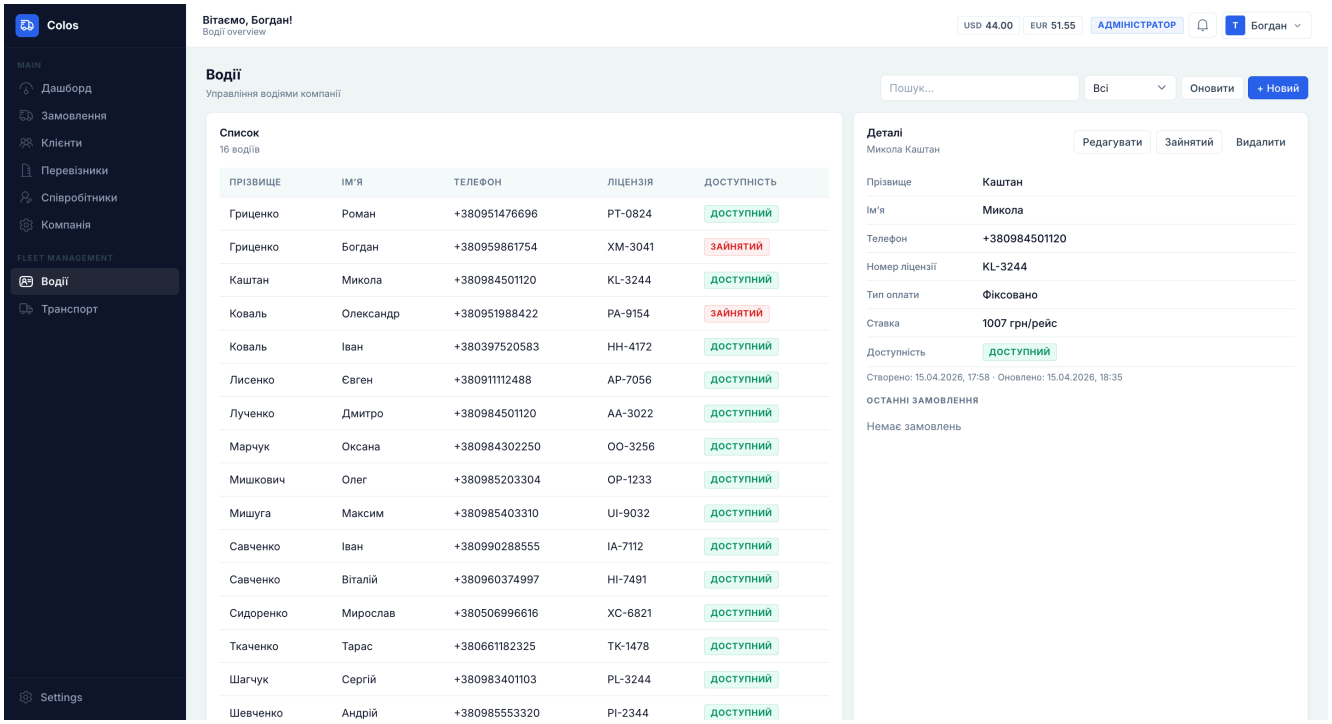


Рис. 3.5 Сторінка управління водіями

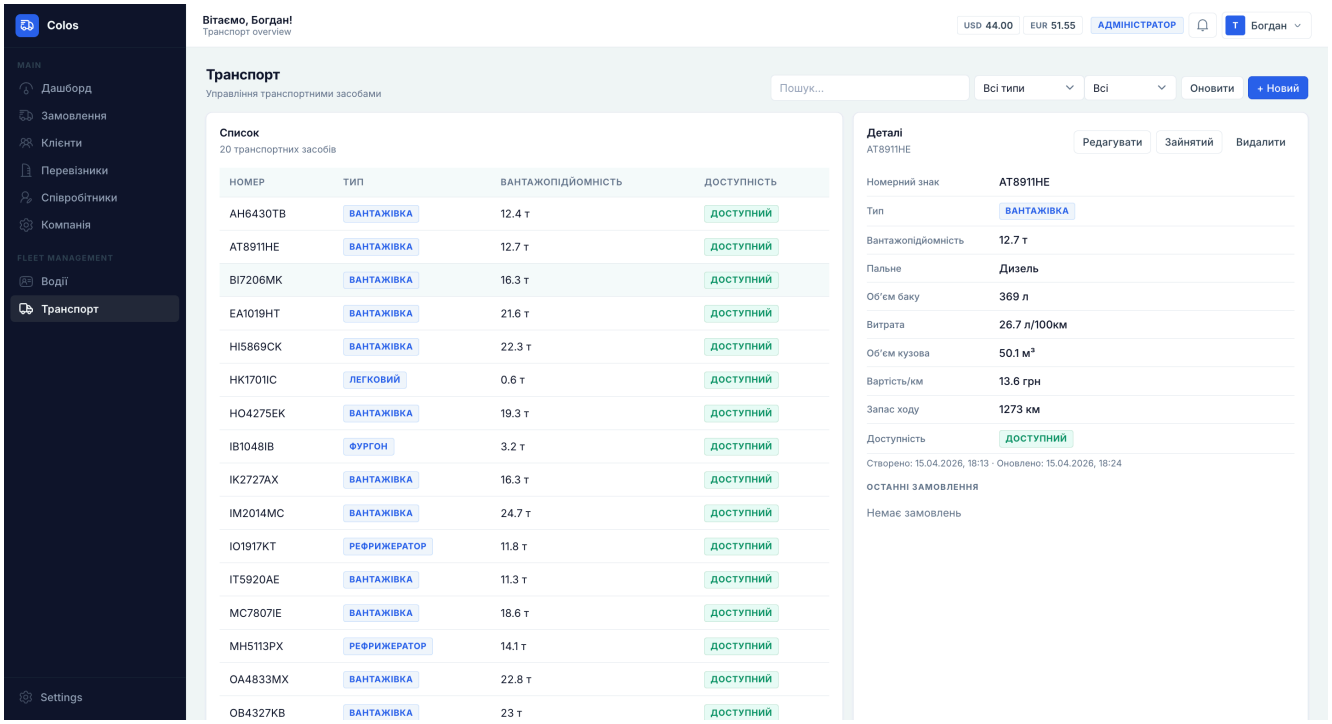


Рис.3.6 Сторінка управління транспортом

Реалізований інтерфейс забезпечує однорідність досвіду користувача в усіх модулях: однакова структура сторінок, уніфіковані кольори статусів,

консистентне розташування кнопок дій та фільтрів. Це скорочує час адаптації нових співробітників і зменшує кількість помилок під час щоденної роботи з системою.

3.2. Реалізація серверної частини та бази даних

Серверна частина системи Colos реалізована як REST API на базі Node.js та Express із підключенням до бази даних PostgreSQL через Prisma ORM. Код організовано за модульним принципом - кожна сутність має окремий роутер, контролер та схему валідації. Для підвищення продуктивності використовується Redis як кеш-сховище, авторизація побудована на JWT-токенах із розмежуванням прав за системою ролей RBAC, а real-time оновлення забезпечуються через Socket.io.

3.2.1 Проєктування та реалізація бази даних

База даних системи Colos побудована на PostgreSQL і описана через схему Prisma ORM. Вся модель даних побудована навколо мультитенантної архітектури - сутність Company є центральною точкою, до якої прив'язані абсолютно всі записи через поле company_id. Це забезпечує повну ізоляцію даних між різними компаніями, які можуть використовувати систему одночасно.

Схему бази даних системи Colos зображено на рисунку Б.1 (Додатку Б)

Розглянемо основні сутності схеми детальніше.

Company – центральна сутність системи. Містить назву, контактні дані та режим роботи компанії (operation_mode): OWN_FLEET - лише власний автопарк, BROKER - лише сторонні перевізники, HYBRID - обидва варіанти одночасно. Від цього поля залежить, які функції будуть доступні в інтерфейсі.

User та UserRole – обліковий запис працівника та його ролі. Ролі винесені в окрему таблицю UserRole, що дозволяє одному користувачу мати кілька ролей одночасно. Підтримувані ролі: ADMIN, MANAGER, DISPATCHER,

ACCOUNTANT, LOGIST, DRIVER. Пароль зберігається у вигляді хешу через bcryptjs - відкритий текст не потрапляє до бази навіть у процесі розробки.

Client – картка клієнта-замовника. Містить назву компанії або ім'я фізичної особи, контактну особу, телефон, email та адресу. Один клієнт може мати необмежену кількість замовлень.

Order – ключова сутність усієї системи. Замовлення містить адреси забору та доставки, дати, опис вантажу, а також поле execution_type зі значеннями INTERNAL або EXTERNAL. Саме це поле визначає бізнес-логіку: для внутрішнього виконання замовлення прив'язується до водія (driver_id) і транспортного засобу (vehicle_id), для зовнішнього - до стороннього перевізника (carrier_id). Фінансовий блок включає ціну для клієнта (client_price), загальні витрати (total_cost), маржу (margin) та відсоток прибутку (margin_percent). Статус замовлення змінюється через enum OrderStatus: NEW → CONFIRMED → IN_TRANSIT → DELIVERED / CANCELLED.

Driver – картка водія з особистими даними, номером ліцензії та параметрами оплати. Тип оплати (pay_type) може бути PER_KM, PER_HOUR, PER_DAY або FIXED - це використовується при автоматичному розрахунку витрат на замовлення. Поле is_available відображає поточну зайнятість водія.

Vehicle – транспортний засіб автопарку. Містить номерний знак, тип (TRUCK, VAN, CAR, REFRIGERATOR), вантажопідйомність, тип палива, витрату пального та вартість кілометру пробігу. Останні два поля використовуються разом із актуальними цінами на паливо для розрахунку собівартості рейсу.

Carrier – сторонній перевізник. Зберігає контактну особу, телефон, максимальну вантажопідйомність, рейтинг та зони покриття у вигляді масиву рядків. Використовується для замовлень із типом виконання EXTERNAL.

ExchangeRate та FuelPrice – таблиці для зберігання ринкових даних, які автоматично оновлюються з зовнішніх API. ExchangeRate містить актуальні курси

валют від НБУ, FuelPrice - ціни на паливо. Обидві таблиці мають поле fetched_at для відстеження актуальності даних.

3.2.2 Реалізація серверної частини

Серверна частина системи Colos побудована на Node.js з TypeScript і організована за модульним принципом. Кожна сутність системи має окремий файл роутера, що містить маршрути, middleware-стек та схему валідації. Такий підхід дозволяє розширювати систему без втручання в існуючий код - новий модуль підключається до головного файлу як незалежна одиниця. Структуру серверної частини зображено на рисунку 3.8.

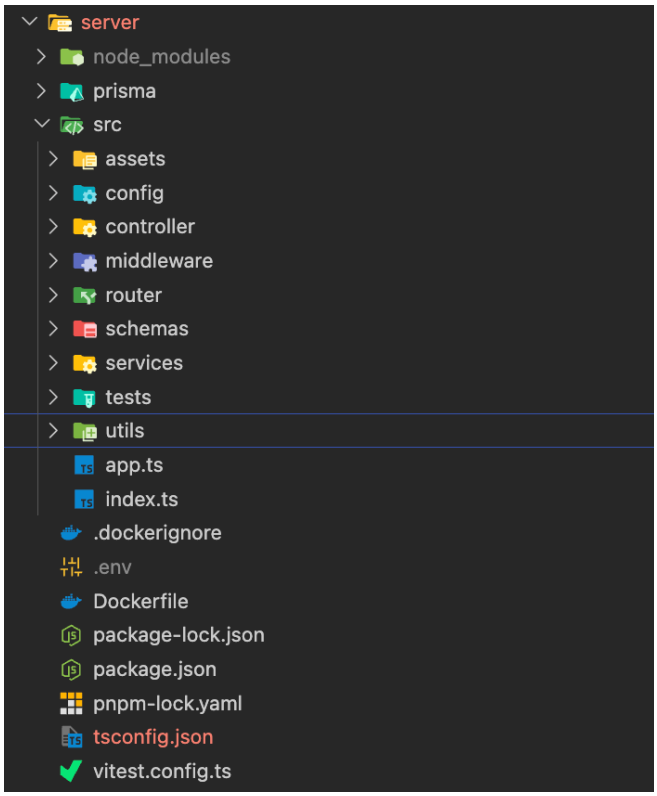


Рис. 3.8 Структура серверної частини проєкту у VS Code

Кожен вхідний запит проходить фіксований ланцюг middleware перед тим як потрапити до обробника бізнес-логіки. Порядок обробки є принциповим: глобальні middleware (Helmet, rate-limit) → перевірка токена (requireAuth) → перевірка ролі (authorize) → валідація тіла запиту (validate) → обробник. Якщо

будь-який крок не пройдено - запит зупиняється з відповідним HTTP-кодом, не досягаючи бази даних.

Авторизація використовує JWT-токенах. У токен вкладаються user_id, company_id та масив ролей. Middleware requireAuth верифікує підпис токена при кожному запиті, а authorize(roles[]) звіряє ролі користувача з дозволеним списком конкретного маршруту. company_id береться виключно з токена - це архітектурне рішення унеможливлює доступ до даних іншої компанії навіть при навмисній підміні параметрів запиту.

Вхідні дані перевіряються через Zod-схеми до потрапляння в обробник. Схема описує типи, обов'язковість та формат кожного поля. При невідповідності повертається 400 Bad Request з переліком конкретних помилок. Детальні приклади схем розглянуто в розділі 4.3.

Для зменшення навантаження на базу даних часто запитувані дані кешуються в Redis з TTL 5 хвилин. Двосторонній зв'язок між сервером і клієнтом у реальному часі забезпечується через Socket.io - оновлення статусів замовлень і геолокація водіїв передаються без перезавантаження сторінки.

3.3 Реалізація основного функціоналу

У цьому пункті розглядається реалізація ключових функціональних модулів CRM-системи Colos: авторизація та система middleware, розрахунок вартості замовлення, інтеграція із зовнішніми API та real-time оновлення.

3.3.1 Реалізація авторизації та middleware

Реалізація авторизація побудована на JWT-токенах. При успішному вході сервер генерує токен, до якого вкладаються user_id, company_id та масив ролей користувача. Клієнт зберігає токен і передає його з кожним запитом у заголовку Authorization: Bearer <token>.

Middleware requireAuth перевіряє наявність і коректність токена при кожному захищеному запиті:

typescript

```
export function requireAuth(req: Request, res: Response, next: NextFunction) {
  const token = getBearerToken(req);
  if (!token) return fail(res, 401, 'Missing access token');
  try {
    const payload = verifyAccessToken(token);
    (req as AuthenticatedRequest).auth = payload;
    return next();
  } catch {
    return fail(res, 401, 'Invalid or expired access token');
  }
}
```

Розмежування прав реалізовано через middleware authorize, який перевіряє ролі користувача:

typescript

```
export function authorize(allowed: Role[]) {
  return (req: Request, res: Response, next: NextFunction) => {
    const auth = (req as AuthenticatedRequest).auth;
    const hasAccess = auth.roles.some((r) => allowed.includes(r));
    if (!hasAccess) return fail(res, 403, 'Forbidden');
    return next();
  };
}
```

Валідація вхідних даних виконується через Zod. Middleware validate перевіряє тіло запиту до потрапляння в обробник:

typescript

```
export function validate(schema: ZodSchema) {
```

```
    return (req: Request, res: Response, next: NextFunction) => {
      const parsed = schema.safeParse(req.body);
      if (!parsed.success) {
        return fail(res, 400, 'Validation error', parsed.error.errors);
      }
      req.body = parsed.data;
      next();
    };
  }
```

Порядок підключення middleware є принциповим - база даних не отримує жодного запиту від неавторизованого або некоректного джерела.

3.3.2 Розрахунок вартості замовлення

Система підтримує два типи виконання замовлення, які визначають склад витрат. Для INTERNAL – власний автопарк – витрати складаються з вартості пального та зарплати водія. Для EXTERNAL – аутсорс – витрати дорівнюють узгодженій ціні перевізника.

Центральна функція computeFinancials виконує розрахунок фінансових показників:

typescript

```
// server/src/router/orders.router.ts
  function computeFinancials(data: {
    executionType: ExecutionType;
    estimatedFuelCost?: number | null;
    estimatedSalaryCost?: number | null;
    carrierAgreedPrice?: number | null;
    clientPrice: number;
  }) {
    const internalCost =
```

```
    (data.estimatedFuelCost ?? 0) + (data.estimatedSalaryCost ?? 0);
  const externalCost = data.carrierAgreedPrice ?? 0;
  const totalCost =
    data.executionType === 'INTERNAL' ? internalCost : externalCost;
  const margin = data.clientPrice - totalCost;
  const marginPercent =
    data.clientPrice > 0 ? (margin / data.clientPrice) * 100 : 0;
  return {
    total_cost: totalCost,
    margin: Math.round(margin * 100) / 100,
    margin_percent: Math.round(marginPercent * 100) / 100,
  };
}
```

Формули розрахунку:

- total_cost = estimatedFuelCost + estimatedSalaryCost (для INTERNAL)
- total_cost = carrierAgreedPrice (для EXTERNAL)
- margin = clientPrice − total_cost
- margin_percent = (margin / clientPrice) × 100

Функція викликається як при створенні, так і при редагуванні замовлення. При редагуванні поточні значення використовуються як fallback, якщо нові не передані:

typescript

```
// PUT /:id - редагування
  const financials = computeFinancials({
    executionType,
    estimatedFuelCost:
      toFloat(body.estimatedFuelCost) ?? existing.estimated_fuel_cost,
    estimatedSalaryCost:
```

```
    toFloat(body.estimatedSalaryCost) ?? existing.estimated_salary_cost,
   carrierAgreedPrice:
    toFloat(body.carrierAgreedPrice) ?? existing.carrier_agreed_price,
   clientPrice,
  });
```

Для автоматичної нумерації замовлень реалізовано функцію generateOrderNumber, яка формує унікальний номер у форматі ORD-YYYYMMDD-XXXX:

typescript

```
async function generateOrderNumber(companyId: string): Promise<string> {
  const ds = new Date().toISOString().slice(0, 10).replace(/-/g, '');
  const prefix = `ORD-${ds}-`;
  // ... пошук останнього номера і інкремент
  return `${prefix}${String(seq).padStart(4, '0')}`; // ORD-20260429-0001
}
```

3.3.3 Інтеграція з API НБУ та цінами на паливо

Зовнішні ринкові дані отримуються через сервіс market-data.service.ts. Планувальник запускає автоматичне оновлення двічі на день - о 08:00 та 18:00 за київським часом:

typescript

```
// server/src/services/market-data.service.ts
export function startMarketDataScheduler(): void {
  cron.schedule('0 8,18 * * *', async () => {
    await fetchAllMarketData();
  });
}
```

Курси валют отримуються з офіційного API Національного банку України. Система відстежує 5 валют: USD, EUR, PLN, GBP, CHF:

typescript

```
const TRACKED_CURRENCIES = ['USD', 'EUR', 'PLN', 'GBP', 'CHF'];
async function fetchExchangeRates(): Promise<void> {
  const response = await axios.get<NbuRate[]>(
    'https://bank.gov.ua/NBUStatService/v1/statdirectory/exchange?json',
    { timeout: 10_000 },
  );
  const filtered = response.data.filter((r) =>
    TRACKED_CURRENCIES.includes(r.cc)
  );
  await prisma.exchange_rates.createMany({
    data: filtered.map((r) => ({ currency: r.cc, rate: r.rate })),
  });
}
```

Оскільки публічного API для цін на паливо не існує, система отримує дані через парсинг HTML-сторінки auto.ria.com для обраного регіону. Якщо автоматичний скрапінг недоступний, адміністратор може ввести ціни вручну через маршрут POST /api/market-data/fuel-prices.

3.3.4 Real-time оновлення через Socket.io

Socket.io-сервер ініціалізується поверх HTTP-сервера Express і зберігається як синглтон:

typescript

```
// server/src/services/socket.ts
export function initSocketServer(httpServer: HttpServer): SocketServer {
  io = new SocketServer(httpServer, {
    cors: { origin: env.CLIENT_ORIGIN, credentials: true },
  });
```

```
  return io;
}
```

Кожне WebSocket-з'єднання проходить ту саму JWT-перевірку що й HTTP-запити. Токен передається через socket.handshake.auth.token:

typescript

```
io.use((socket, next) => {
  const token = socket.handshake.auth.token as string | undefined;
  if (!token) return next(new Error('unauthorized'));
  try {
    const payload = verifyAccessToken(token);
    socket.data.companyId = payload.company_id;
    socket.data.userId = payload.sub;
    next();
  } catch {
    next(new Error('unauthorized'));
  }
});
```

Після підключення сокет автоматично вступає у дві кімнати - компанії та користувача. Це забезпечує мультитенантну ізоляцію:

typescript

```
io.on('connection', (socket) => {
  socket.join(`company:${socket.data.companyId}`);
  socket.join(`user:${socket.data.userId}`);
});
```

Для надсилання подій реалізовано окремі функції. При зміні статусу замовлення подія надсилається всім підключеним клієнтам компанії, персональні сповіщення - конкретному користувачу:

typescript

```
// Зміна статусу → всі сокети компанії
```

```
export function emitOrderStatusChanged(companyId: string, data: {
orderId: string; orderNumber: string;
newStatus: string; previousStatus: string;
}): void {
io?.to(`company:${companyId}`).emit('order:statusChanged', data);
}

// Персональне сповіщення → конкретний користувач
export function emitNotification(userId: string, payload: NotificationPayload): void
{
io?.to(`user:${userId}`).emit('notification', payload);
}
```

Ці функції викликаються безпосередньо в обробниках маршрутів після збереження змін до бази даних, що гарантує узгодженість між станом БД та реальним часом на клієнті.

3.4 Тестування програмного забезпечення

Тестування є невід'ємним етапом розробки будь-якого програмного продукту - саме цей процес контролює якість та надійність кінцевого результату. У проєкті Colos тестування серверної частини реалізовано за допомогою фреймворку Vitest у поєднанні з бібліотекою Supertest, яка дозволяє надсилати HTTP-запити до Express-застосунку без запуску реального сервера.

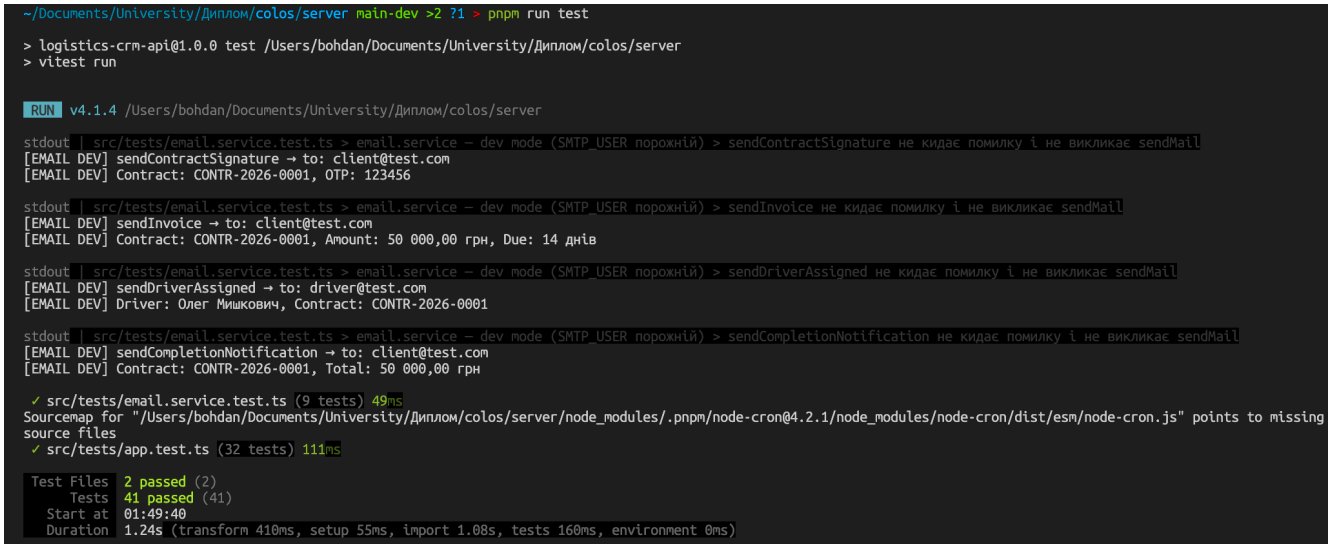


Рис. 3.9 Результати виконання автоматизованих тестів

Тест-кейс – це артефакт, що описує сукупність кроків, умов та параметрів, необхідних для перевірки реалізації певної функції або її частини. Тести охоплюють як позитивні сценарії - коректна поведінка при правильних вхідних даних, так і негативні – коректна обробка помилок при неправильних або відсутніх даних.

Таблиця 3.1

Тест-кейси для тестування функціоналу авторизації

Крок тесту	Вхідні дані	Очікуваний результат
Вхід з коректними даними	Валідний email та пароль	HTTP 200, повертається access token та дані користувача
Вхід з некоректним паролем	Валідний email, невірний пароль	HTTP 401, повідомлення «Invalid email or password»
Вхід з неіснуючим email	Неіснуючий email	HTTP 401, повідомлення «Invalid email or password»
Вхід з неактивним обліковим записом	Email деактивованого користувача	HTTP 403, повідомлення «User is inactive»
Запит до захищеного маршруту без токена	Відсутній заголовок Authorization	HTTP 401, повідомлення «Missing access token»
Запит до захищеного маршруту з простроченим токеном	Прострочений JWT	HTTP 401, повідомлення «Invalid or expired access token»

Таблиця 3.2

Тест-кейси для тестування функціоналу замовлень

Крок тесту	Вхідні дані	Очікуваний результат
Створення замовлення з коректними даними	Всі обов'язкові поля заповнені, execution_type: INTERNAL	HTTP 201, замовлення збережено, номер у форматі ORD-YYYYMMDD-XXXX
Створення замовлення без обов'язкових полів	Відсутній client_id або pickup_address	HTTP 400, повідомлення з описом відсутніх полів
Створення замовлення користувачем без прав	Роль DRIVER, коректні дані	HTTP 403, повідомлення «Forbidden»
Зміна статусу замовлення	Коректний orderId, новий статус CONFIRMED	HTTP 200, статус оновлено, Socket.io подія order:statusChanged надіслана
Зміна статусу на недопустиме значення	Статус INVALID_STATUS	HTTP 400, повідомлення про помилку валідації
Перегляд списку замовлень	Авторизований менеджер	HTTP 200, список замовлень компанії з пагінацією

За результатами тестування всі 41 тест проходять успішно. Результати виконання тестів зображено на рисунку 3.9.

Таблиця 3.3

Тест-кейси для тестування функціоналу налаштувань компанії

Крок тесту	Вхідні дані	Очікуваний результат
Отримання даних компанії	Авторизований користувач будь-якої ролі	HTTP 200, дані компанії у форматі camelCase
Отримання даних без токена	Відсутній заголовок Authorization	HTTP 401, повідомлення «Missing access token»
Оновлення даних компанії адміністратором	Роль ADMIN, поле name: 'Нова назва'	HTTP 200, дані оновлено, updated_at проставлено
Оновлення даних не-адміністратором	Роль MANAGER, коректні дані	HTTP 403, повідомлення «Forbidden», база даних не викликається
Оновлення з некоректними даними	name: 'X' (менше 2 символів)	HTTP 400, повідомлення про помилку валідації
Очищення nullable-поля	email: null	HTTP 200, поле email відсутнє у відповіді

Таблиця 3.4

Тест-кейси для тестування інтеграції із зовнішніми API

Крок тесту	Вхідні дані	Очікуваний результат
Отримання курсів валют	Запит до GET /api/market-data/exchange-rates	HTTP 200, актуальні курси USD, EUR, PLN, GBP, CHF
Примусове оновлення ринкових даних	POST /api/market-data/fetch, роль ADMIN	HTTP 200, дані оновлено з API НБУ та auto.ria.com
Ручне введення цін на паливо	POST /api/market-data/fuel-prices, коректні ціни	HTTP 200, ціни збережено з source: 'manual'
Введення некоректних цін	Від'ємне значення ціни	HTTP 400, повідомлення про помилку валідації

Проведене тестування підтвердило коректну роботу всіх реалізованих модулів системи. Автоматизовані тести покривають як стандартні сценарії використання, так і граничні випадки - некоректні дані, відсутні токени, недостатні права доступу. Використання Vitest та Supertest дозволило виявити та усунути потенційні помилки на етапі розробки, до розгортання системи.

ВИСНОВКИ

Робота присвячена розробці CRM-системи для логістичного підприємства, що забезпечує централізоване керування клієнтами, замовленнями, автопарком та сторонніми перевізниками в єдиному веб-застосунку.

1. Проведено аналіз предметної області логістики та обґрунтовано актуальність розробки власної CRM-системи. Встановлено, що більшість існуючих рішень орієнтована на продажі та маркетинг і не покриває специфічних потреб логістичного підрозділу - керування маршрутами, контроль статусів доставки, розрахунок маржинальності та розподіл задач між різними ролями персоналу.

2. Проаналізовано три популярні CRM-системи - HubSpot CRM, Pipedrive та Zoho CRM. Визначено їх переваги та недоліки з точки зору логістичного застосування. На основі порівняння обґрунтовано доцільність розробки власного рішення, адаптованого під конкретний бізнес-процес без надлишкового функціоналу та залежності від сторонніх платформ.

3. Описано програмні засоби, які застосовувалися для розробки системи. Після аналізу вимог до програмного забезпечення визначено технологічний стек: TypeScript як основна мова програмування, React 19 та Vite на клієнтській частині, Node.js та Express на серверній, Prisma ORM та PostgreSQL для роботи з базою даних, Redis для кешування, Socket.io для real-time оновлень. Обґрунтовано, що використання TypeScript на всіх рівнях застосунку забезпечує типову безпеку та спрощує підтримку системи.

4. Спроєктовано та реалізовано архітектуру системи з підтримкою мультитенантності - кілька компаній можуть використовувати систему одночасно з повною ізоляцією даних через company_id. Реалізовано розмежування доступу за шістьма ролями: ADMIN, MANAGER, DISPATCHER, LOGIST, ACCOUNTANT, DRIVER. Розроблено модуль автоматичного розрахунку вартості замовлення з урахуванням типу виконання, витрат на паливо та оплати водія.

Реалізовано інтеграцію з API Національного банку України для отримання актуальних курсів валют та парсинг цін на паливо з auto.ria.com.

5. Для забезпечення якості програмного забезпечення проведено тестування з використанням Vitest та Supertest. Розроблено 41 автоматизований тест, що охоплює авторизацію, керування замовленнями, налаштування компанії та інтеграцію із зовнішніми API. Всі тести проходять успішно, що підтверджує коректну роботу реалізованих модулів.

6. Розроблена система буде корисною для менеджерів, логістів та диспетчерів підприємств, що займаються організацією та контролем доставок. Рекомендовано застосовувати систему як основний інструмент оперативного керування логістичними процесами з можливістю подальшого розширення функціоналу - додавання модуля GPS-трекінгу водіїв, мобільного застосунку для водіїв та розширеної аналітики по маршрутах.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. HubSpot CRM - офіційна документація [Електронний ресурс] – Режим доступу: https://www.hubspot.com/products/crm
2. Pipedrive - офіційна документація [Електронний ресурс] – Режим доступу: https://www.pipedrive.com
3. Zoho CRM - офіційна документація [Електронний ресурс] – Режим доступу: https://www.zoho.com/crm
4. TypeScript - офіційна документація [Електронний ресурс] – Режим доступу: https://www.typescriptlang.org/docs
5. React - офіційна документація [Електронний ресурс] – Режим доступу: https://react.dev
6. Node.js - офіційна документація [Електронний ресурс] – Режим доступу: https://nodejs.org/en/docs
7. Express.js - офіційна документація [Електронний ресурс] – Режим доступу: https://expressjs.com
8. Prisma ORM - офіційна документація [Електронний ресурс] – Режим доступу: https://www.prisma.io/docs
9. PostgreSQL - офіційна документація [Електронний ресурс] – Режим доступу: https://www.postgresql.org/docs
10. Redis - офіційна документація [Електронний ресурс] – Режим доступу: https://redis.io/docs
11. Socket.io - офіційна документація [Електронний ресурс] – Режим доступу: https://socket.io/docs/v4
12. Zod - офіційна документація [Електронний ресурс] – Режим доступу: https://zod.dev
13. Vite - офіційна документація [Електронний ресурс] – Режим доступу: https://vitejs.dev/guide

14. Mapbox GL JS - офіційна документація [Електронний ресурс] – Режим доступу: https://docs.mapbox.com/mapbox-gl-js/api
15. Національний банк України - API курсів валют [Електронний ресурс] – Режим доступу: https://bank.gov.ua/ua/open-data/api-dev
16. Docker - офіційна документація [Електронний ресурс] – Режим доступу: https://docs.docker.com
17. Чохленко Б. А., Жебка В. В. Crm-система для оптимізації логістики з використанням TypeScript // Матеріали Всеукраїнської науково-технічної конференції «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. К. : ДУІКТ, 2026.
18. Чохленко Б. А., Жебка В. В. Crm-система для оптимізації логістики з використанням TypeScript // Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. К. : ДУІКТ, 2026.

ДОДАТКИ А

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Технологій цифрового розвитку

CRM-система для оптимізації логістики з використанням TypeScript

Виконав студент 4 курсу
Групи ТЦР-43
Чохленко Богдан Андрійович
Керівник роботи
Д-р техн. наук, професор, зав. кафедри ТЦР Жебка Вікторія Вікторівна

Київ-2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Об'єкт дослідження** - процеси управління логістичними операціями підприємства, що включають роботу з клієнтами, замовленнями, маршрутами та контролем доставки.
- **Предмет дослідження** - методи та програмні засоби розробки CRM-системи для оптимізації логістики з використанням TypeScript.
- **Мета роботи** – оптимізація логістики за допомогою розробленої веб-орієнтованої CRM-системи, що забезпечує централізоване ведення клієнтської бази, реєстрацію та обробку логістичних замовлень, відстеження статусів доставки і зменшення кількості ручних операцій у щоденній роботі підприємства.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Проаналізувати предметну область логістики та дослідити існуючі CRM-рішення.
2. Сформувати вимоги до системи та визначити ролі користувачів (6 ролей: ADMIN, MANAGER, DISPATCHER, LOGIST, ACCOUNTANT, DRIVER).
3. Спроектувати архітектуру системи та розробити схему бази даних.
4. Реалізувати REST API на Node.js + Express з JWT-авторизацією та RBAC.
5. Розробити клієнтський інтерфейс на React з інтеграцією Mapbox та Socket.io.
6. Провести автоматизоване тестування за допомогою Vitest та Supertest.

АНАЛІЗ АНАЛОГІВ

Функціональність/Показник	**HubSpot CRM**	**Pipedrive**	**Zoho CRM**	**Colos CRM**
Логістичний функціонал	-	-	Частково	+
Керування маршрутами	-	-	-	+
Статуси доставки	-	-	-	+
Фінансовий блок / маржа	-	-	-	+
Real-time оновлення	+	-	+	+

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Веб	Сервер	База даних
React JS	Node.js	PostgreSQL
TypeScript	TypeScript	Redis
Mapbox	Express.js	Prisma
Socket.io	Vitest	
Vite		

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ

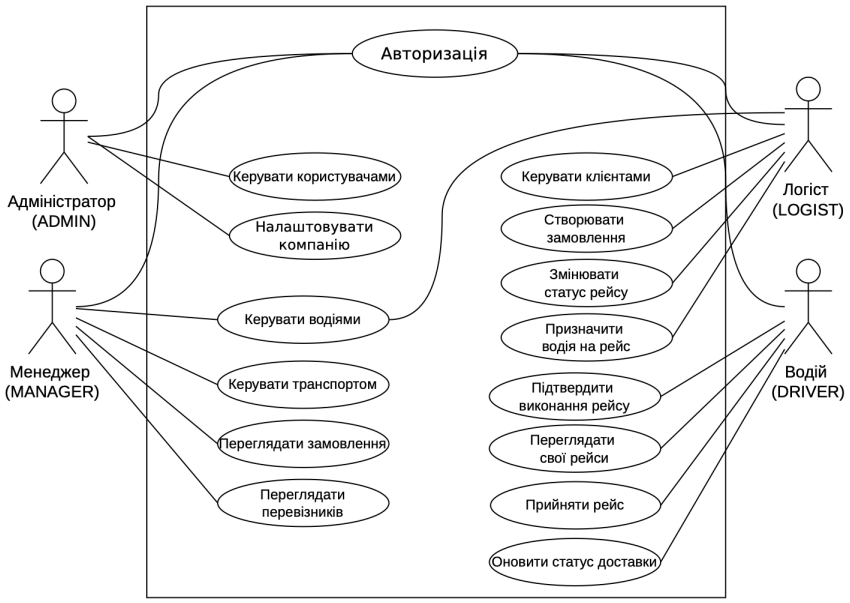
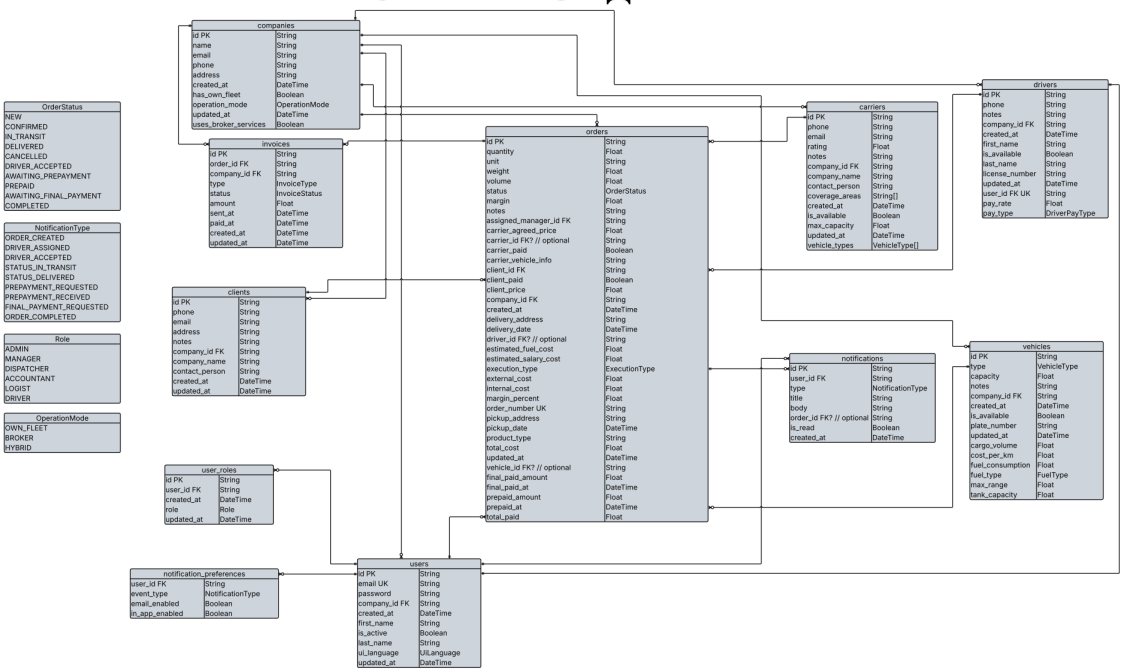
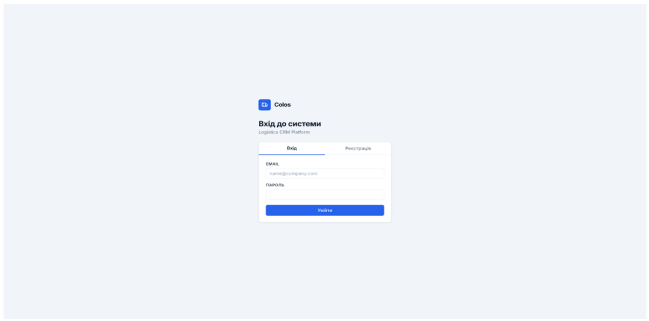


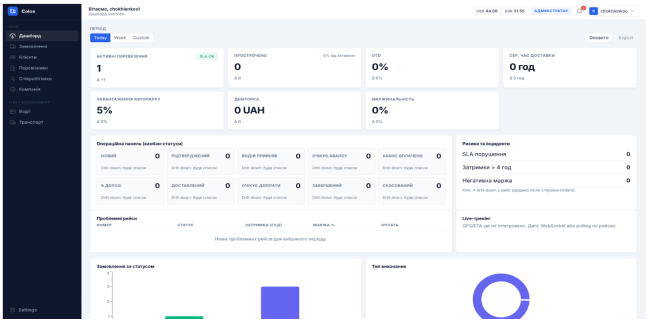
СХЕМА БАЗИ ДАНИХ



ЕКРАННІ ФОРМИ

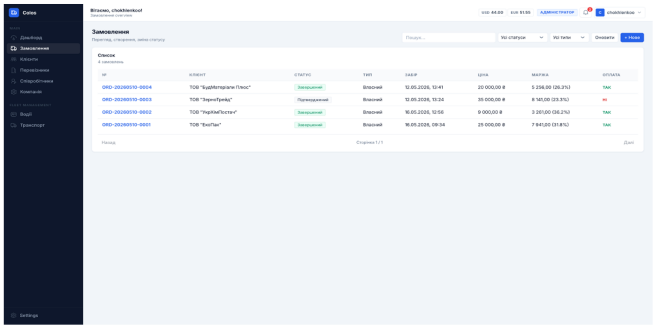


Сторінка авторизації

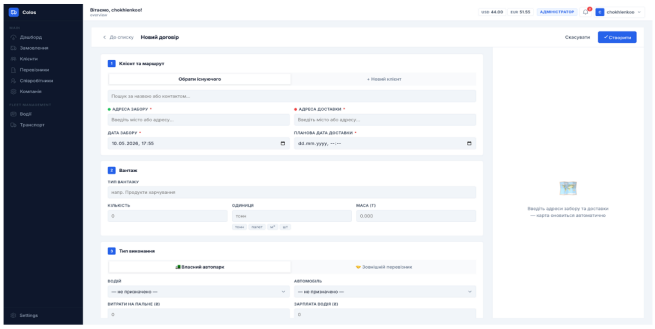


Дашборд

ЕКРАННІ ФОРМИ

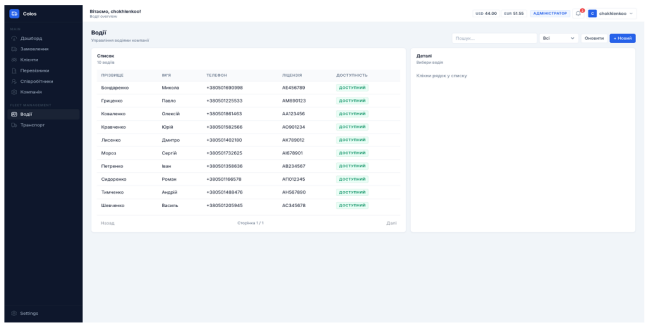


Сторінка списку замовлень

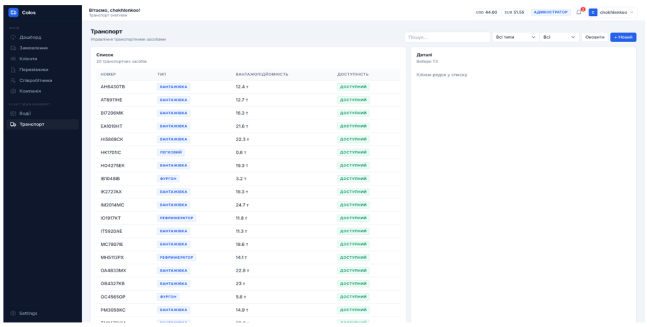


Форма створення нового замовлення

ЕКРАННІ ФОРМИ



Сторінка управління водіями



Сторінка управління транспортом

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Чохленко Б.А., Жебка В.В. Децентралізація даних у сучасних інформаційних системах: необхідність та переваги // Матеріали Всеукраїнської науково-технічної конференції «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. К.: ДУІКТ, 2026.

2. Чохленко Б.А., Жебка В.В. Автоматизація логістичних процесів засобами CRM-системи // Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. К.: ДУІКТ, 2026.

ВИСНОВОК

1. Проаналізовано предметну область логістики та CRM-аналоги. Обґрунтовано необхідність власного рішення.
2. Спроектовано мультитенантну архітектуру з ізоляцією даних компаній через company_id та RBAC з 6 ролями.
3. Розроблено модуль автоматичного розрахунку вартості замовлення з урахуванням типу виконання, палива та оплати водія.
4. Реалізовано інтеграцію з API НБУ (5 валют) та цінами на паливо для актуального розрахунку собівартості.
5. Проведено 41 автоматизований тест (Vitest + Supertest) - всі проходять успішно.
6. Система готова до впровадження і може бути розширена модулем GPS-трекінгу та мобільним застосунком для водіїв.

ДЯКУЮ ЗА УВАГУ!

ДОДАТКИ Б

Рисунок Б.1 - ERD-схема бази даних CRM-системи Colos