Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Веб-застосунок для керування логістикою вантажоперевезень

(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІ-21-2
Андрій ПОЛЕЖАЄВ
(власне ім’я, прізвище)

Спеціальність
123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп’ютерна інженерія
(повна назва освітньої програми)

Керівник: ас. Владислав ХОЛЄВ
(посада, власне ім’я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім’я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп’ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Полежаєву Андрію Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Веб-застосунок для керування логістикою вантажоперевезень

затверджена наказом по університету від “ 26 ” травня 2025 р. № 424 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 17 червня 2025 р.

3. Вхідні дані до роботи

1) документація мови програмування Java
2) документація технології створення настільних програм Swing UI
3) перелік використаних програмних засобів: ОС Windows 11
4) інтегроване середовище IntelliJ IDEA Community Edition 2025.1.1.1

4. Перелік питань, що потрібно опрацювати у роботі

1. Аналіз предметної області
2. Використовувані технології
3. Програмна реалізація
4. Особливості використання додатку
5. Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій 12 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Аналіз проблеми та огляд існуючих рішень	27.04 – 29.04	
2	Формування переліку вимог до програми	30.04 – 01.05	
3	Вибір технології розробки та	02.05	
	інструментальних засобів		
4	Розробка алгоритмічного забезпечення	03.05 – 05.05	
5	Розробка та тестування програми	06.05 – 30.05	
6	Відлагодження програмних модулів	31.05 – 02.06	
7	Оформлення матеріалів до атестаційної роботи	03.06 – 16.06	

Дата видачі завдання " 26 " травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ас. Владислав Холєв
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 70 с., 21 рис., 1 табл., 1 дод., 14 джерел.

ЛОГІСТИКА, ТРАНСПОРТНІ ПЕРЕВЕЗЕННЯ, ІНТЕРНЕТ, ОБРОБКА ДАНИХ, БАЗА ДАНИХ, POSTGRESQL, MVC, API, HTTP, КЛІЄНТ-СЕРВЕР.

Основною метою даної бакалаврської роботи є розробка якісної архітектури програмного забезпечення, яке може використовуватись початковими компаніями у сфері логістики для управління перевезеннями. Передбачено можливість зберігання інформації про вантажі та користувачів, а також спільне використання програми з інтуїтивно зрозумілим інтерфейсом і добре задокументованим серверним рівнем із HTTP-запитами.

Для виконання даної роботи було проведено аналіз переваг та недоліків існуючих рішень. Програмне забезпечення було розроблено після визначення всіх вимог. Для реалізації поставленого завдання використовувалися мова програмування Java, інтерфейс Swing UI, фреймворк Spring Boot, сервіс аутентифікації AWS Cognito, реляційна база даних PostgreSQL, а також інструмент Postman.

ABSTRACT

Bachelor’s thesis: 70 pages, 21 figures, 1 tables, 1 appendices, 14 sources.

LOGISTICS, TRANSPORTATION, INTERNET, DATA PROCESSING, DATABASE, POSTGRESQL, MVC, API, HTTP, CLIENT-SERVER

The major goal of this thesis is the development of high-quality architecture program that can be used by starting companies in the logistics sphere for transportation management. There is a possibility to store information about loads and users, and it is available the common usage of a program that provides with understandable interface and good commented with HTTP-requests on the server layer.

In order to do this work an analysis of advantages an disadvantages of existing solutions was done. The software was developed after all requirements had been drafted. The Java programming language, the Swing UI, the Spring Boot framework, the AWS Cognito, the PostgreSQL relational database, the Postman instrument were used to complete the task.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

API – Інтерфейс прикладного програмування (Application Programming Interface)

AWS – Хмарна платформа Amazon Web Services (Amazon Web Services)

CRM – Система управління взаємовідносинами з клієнтами (Customer Relationship Management)

CRUD – Операції створення, читання, оновлення та видалення даних (Create, Read, Update, Delete)

DHL – Міжнародна логістична компанія (Dalsey, Hillblom and Lynn)

DTO – Об'єкт передавання даних (Data Transfer Object)

ERP – Система планування ресурсів підприємства (Enterprise Resource Planning)

EWM – Система управління складом (Extended Warehouse Management)

FedEx – Міжнародна служба експрес-доставки (Federal Express)

GTS – Глобальна торгівельна система (Global Trade Services)

JPA – Інтерфейс доступу до даних у Java (Java Persistence API)

JVM – Віртуальна машина Java (Java Virtual Machine)

McKinsey – Глобальна консалтингова компанія (McKinsey & Company)

MVC – Архітектурний шаблон модель-подання-контролер (Model-View-Controller)

ORM – Об’єктно-реляційне відображення (Object-Relational Mapping)

REST – Стиль архітектури для побудови веб-сервісів (Representational State Transfer)

SAP – Програмне забезпечення для бізнес-процесів (Systems, Applications and Products in Data Processing)

SaaS – Програмне забезпечення як сервіс (Software as a Service)

SCM – Управління ланцюгом постачання (Supply Chain Management)

SQL – Мова структурованих запитів до бази даних (Structured Query Language)

S/4HANA – Платформа для зберігання та обробки даних (SAP Business Suite 4 for SAP HANA)

TM – Управління транспортом (Transportation Management)

TMS – Система управління транспортом (Transportation Management System)

UI – Інтерфейс користувача (User Interface)

UPS – Міжнародна кур'єрська служба (United Parcel Service)

СКБД – Система керування базами даних

ВСТУП

У наш час перевезення вантажів стало не просто важливою частиною логістики, а справжнім фундаментом для всієї економіки. Якщо товари не будуть вчасно доставлені – це призведе до збоїв у торгівлі, виробництві та навіть у повсякденному житті людей. Тому логістика займається тим, щоб організувати процеси переміщення продукції максимально ефективно та з мінімальними затратами.

Сама логістика вантажоперевезень охоплює різні напрями роботи: від того, як спланувати маршрут, до того, як передати водію документи й упевнитися, що товар дійшов до місця вчасно. Сюди входить:

- підготовка маршруту;
- вибір транспорту;
- контроль виконання;
- зберігання інформації про весь процес.

Існує багато способів доставити вантаж: машинами, потягами, літаками або навіть кораблями. Найчастіше в Україні, як і в багатьох інших країнах, використовуються саме автомобілі. Вони зручні, тому що можуть доїхати майже в будь-яке місце без пересадок чи перевантажень.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

У процесі логістики бере участь велика кількість сторін, кожна з яких виконує свою важливу роль. З одного боку, це замовники перевезень, наприклад торговельні мережі або склади, яким необхідно доставити товари з однієї точки в іншу. З іншого боку – це перевізники, які мають у своєму розпорядженні транспортні засоби для здійснення доставки. Крім того, важливу роль відіграють водії та диспетчери, що координують рух транспорту та відповідають за його оперативну роботу. Усе це об'єднується компаніями, які надають комплексні логістичні послуги та забезпечують злагоджену взаємодію між усіма учасниками процесу. Не менш важливою складовою логістики є документообіг: вантаж повинен супроводжуватись офіційними документами, зокрема товарно-транспортними накладними, актами приймання-передачі тощо. Відсутність таких документів унеможливлює законну та контрольовану доставку.

Типовий логістичний процес включає кілька основних етапів, які послідовно забезпечують переміщення вантажу: спочатку відбувається прийом замовлення від клієнта, далі система або логіст формує оптимальний маршрут, після чого підбирається водій і відповідний транспортний засіб. Наступним кроком є завантаження товару на склад або в пункті відправлення, далі здійснюється безпосередньо доставка, під час якої важливо контролювати переміщення вантажу в реальному часі. Після прибуття виконується розвантаження, а на завершення відбувається оформлення та архівація всіх супровідних документів.

У сучасних умовах практично на кожному з цих етапів використовуються цифрові технології. Наприклад, системи типу TMS (Transportation Management System) дозволяють автоматично прокладати маршрути, відслідковувати транспорт у режимі онлайн, керувати документацією та аналізувати ефективність перевезень. Такі рішення вже

стали стандартом у сучасній логістиці, підвищуючи точність, швидкість і прозорість усіх логістичних операцій (рисунок 1.1).

Рисунок 1.1 – Основні етапи логістичного циклу

Сучасна логістика неможлива без використання цифрових технологій, які суттєво змінюють підхід до управління процесами перевезень. Інформаційні системи відіграють ключову роль у автоматизації основних логістичних функцій, таких як прийом і обробка замовлень, управління складськими запасами, планування оптимальних маршрутів, контроль за виконанням завдань та постійна взаємодія з клієнтами і водіями. Вони також забезпечують створення звітності та проведення аналітики, що дозволяє швидше приймати управлінські рішення на основі реальних даних.

Запровадження подібних технологій не лише підвищує продуктивність роботи, а й робить логістичну систему більш прозорою, передбачуваною та

здатною до адаптації у змінних умовах. Одним із прикладів таких рішень є системи типу TMS (Transportation Management System), які дають змогу централізовано управляти перевезеннями, автоматизувати побудову маршрутів і координувати дії між усіма учасниками логістичного процесу. Завдяки цьому логістика стає більш гнучкою, технологічною та ефективною.

Попри значний прогрес у сфері цифровізації, багато логістичних компаній досі стикаються з низкою серйозних проблем, які заважають їм працювати ефективно. Найпоширенішими з них є низький рівень автоматизації основних процесів, неефективне використання персоналу, зокрема диспетчерів і водіїв, а також відсутність систем моніторингу у реальному часі. Часто інформація дублюється у різних джерелах, що ускладнює її обробку, а брак інтеграції з картографічними сервісами призводить до помилок у побудові маршрутів. Окрім цього, компанії мають труднощі з аналітикою та плануванням на основі фактичних даних.

Однією з найважливіших задач у логістиці залишається оптимізація маршрутів, оскільки помилки на цьому етапі можуть спричинити суттєві втрати. Неправильно спланований маршрут часто призводить до перевитрат пального, зростання витрат на технічне обслуговування транспорту, порушення термінів доставки, а також до зниження рівня довіри з боку клієнтів.

Сучасні алгоритми побудови маршрутів враховують не лише найкоротшу відстань між точками, а й такі фактори, як дорожня ситуація, затори, погодні умови, графіки роботи складів та фізичні характеристики вантажу. Завдяки впровадженню геоінформаційних сервісів, таких як Google Maps або OpenStreetMap, ці параметри можна враховувати динамічно, в режимі реального часу, що істотно підвищує якість логістичного планування.

Цифрова трансформація суттєво змінює логістичну галузь, забезпечуючи покращення якості обслуговування клієнтів, зниження операційних витрат, прозорість ланцюгів постачання, швидку обробку замовлень і точніше планування навантаження. До основних цифрових

інструментів належать мобільні застосунки для водіїв, системи автоматичних сповіщень, онлайн-трекінг, аналітичні модулі для оцінки ефективності доставки, а також інтегровані CRM і ERP-системи, що об'єднують логістичні процеси в єдину цифрову платформу.

1.1 Актуальність теми

Логістика вантажоперевезень стрімко трансформується під впливом цифрових технологій, глобалізації, змін у поведінці споживачів та динаміки ринку. Потреба в автоматизованих, адаптивних і ефективних рішеннях в логістичній сфері сьогодні є як ніколи актуальною. Особливо це актуально в умовах стрімкого розвитку цифрових технологій, які дедалі більше впроваджуються в усі сфери людської діяльності, включно з логістикою.

1.1.1 Глобальні виклики та ринок логістики

Сучасна логістика стикається з багатьма викликами, які стосуються не лише конкуренції між компаніями, а й необхідності відповідати високим стандартам обслуговування, законодавчим вимогам та розвитку цифрових технологій. Усе більше клієнтів очікують швидкого, зручного та прозорого сервісу, тому логістичні компанії змушені шукати нові способи покращення своєї роботи.

Згідно з дослідженнями відомих логістичних компаній (наприклад, DHL, McKinsey), приблизно 40% витрат підприємств у сфері логістики пов’язані з такими проблемами, як неправильне планування маршрутів, простої транспорту та зайві адміністративні процеси. Це призводить до втрати часу, грошей та зниження рівня задоволеності клієнтів (рисунок 1.2).

Ще одним важливим викликом є потреба швидко пристосовуватися до змін – як на ринку, так і в технологіях. Через це компанії активно впроваджують цифрові рішення**,** наприклад, веб-системи для управління

перевезеннями, які дозволяють відстежувати вантажі, вести облік, аналізувати дані та приймати обґрунтовані рішення.

Крім того, важливим стає екологічний підхід: оптимізація маршрутів для зменшення витрат пального, використання більш екологічного транспорту, зменшення паперового документообігу завдяки електронним платформам.

Рисунок 1.2 – Вплив цифровізації на ефективність доставки

1.1.2 Український контекст

Логістична система в Україні сьогодні перебуває в процесі активної трансформації. Зміни в економіці, перебудова ринків, зміна логістичних маршрутів, зростання попиту на швидкі доставки. Усе це створює нові виклики та вимагає сучасних підходів до управління перевезеннями. Логістика має бути не просто функцією доставки, а гнучким та адаптивним процесом, здатним швидко реагувати на зовнішні зміни.

У багатьох випадках малі та середні підприємства в Україні продовжують використовувати застарілі інструменти наприклад, таблиці в Excel, телефонні дзвінки або групові чати у Viber чи Telegram для

координації перевезень. Такий підхід може бути зручним на перших етапах, але з часом створює великі проблеми: помилки у плануванні, водії не отримують оновлену інформацію, диспетчери не мають повної картини ситуації. Усе це призводить до плутанини, затримок у доставці та зниження рівня обслуговування клієнтів.

Особливо актуальною стала потреба у цифровізації логістичних процесів. Підприємства починають розуміти, що без сучасного програмного забезпечення важко забезпечити точність, швидкість та ефективну взаємодію між усіма учасниками процесу такими як диспетчерами, водіями, менеджерами та клієнтами. Саме тому все більше компаній починають впроваджувати прості веб-системи, трекінг, облік замовлень та автоматизовані панелі управління.

1.1.3 Зміни у споживчій поведінці та запити клієнтів

Останні роки показали, що клієнти – як фізичні особи, так і бізнес – стали вимогливішими. Вони очікують:

- доставки «день у день» або хоча б чіткого дотримання термінів;
- точного прогнозування часу прибуття;
- можливості відстеження вантажу онлайн у реальному часі;
- гнучкого вибору водіїв або маршрутів;
- автоматичних сповіщень та інформування про статус замовлення;
- наявності електронних документів та підписів.

Усе це потребує централізованої, адаптивної та цифрової платформи управління логістикою. Веб-застосунки стають тим інструментом, що дозволяє об’єднати всі ці функції в одне рішення.

Автоматизація логістики вантажоперевезень є надзвичайно актуальною в умовах цифрової епохи, коли як держава, так і бізнес активно переходять на сучасні ІТ-рішення. Це дозволяє зменшити витрати, оптимізувати маршрути, знизити навантаження на персонал та скоротити кількість помилок. Завдяки

аналітиці компанії отримують змогу ухвалювати обґрунтовані рішення, а клієнти – якісніший сервіс. Системи легко адаптуються до різних масштабів бізнесу й дозволяють швидко масштабуватись при зростанні обсягів.

Також такі рішення мають соціальну цінність – сприяють створенню нових робочих місць, розвитку «розумної логістики», зменшенню шкідливих викидів та підвищенню прозорості процесів, що особливо важливо в умовах кризових ситуацій.

З технічного боку створення подібних веб-систем є цілком реальним: існує багато відкритих інструментів, API, хмарних сервісів і бібліотек, які дозволяють розробляти ефективні рішення навіть за обмежених ресурсів.

1.2 Загальний огляд існуючих рішень

У сучасному цифровому світі логістика вантажоперевезень є невід'ємною складовою економічної діяльності як великих корпорацій, так і середнього та малого бізнесу. Ефективне управління логістичними процесами дозволяє скорочувати витрати, покращувати обслуговування клієнтів та підвищувати загальну конкурентоспроможність. Саме тому розробка інноваційних рішень у цій галузі набуває особливого значення.

На ринку вже існує чимало рішень, які вирішують завдання логістичного планування, обліку та аналітики. Серед них – як потужні комерційні продукти зі складною архітектурою, так і відкриті платформи, які потребують доопрацювання. Водночас, більшість з них розроблені для використання в умовах великих міжнародних корпорацій та вимагають значних ресурсів як для впровадження, так і для повсякденного використання. Це обмежує їх доступність для середнього та малого бізнесу, який також має нагальну потребу в автоматизації логістичних процесів.

До найвідоміших і найбільш поширених систем логістичного управління відносяться:

- SAP Transportation Management (вважається флагманом у сфері

корпоративної логістики. Забезпечує повний цикл управління транспортом, однак його вартість та складність інтеграції роблять його малопридатним для малого бізнесу [1]);

- Oracle Logistics Cloud (ще одне потужне рішення, орієнтоване на глобальні компанії з розгалуженою структурою. Його функціонал вражає, однак простота використання залишається під питанням[2]);

- Descartes TMS (зручний для міжнародних перевезень, має надійну систему відстеження вантажів. Водночас, через вузьку спеціалізацію він не завжди підходить для внутрішніх перевезень[3]);

- Transporeon (гнучка платформа, популярна в Європі. Проте її ефективність напряму залежить від наявності широкої мережі партнерів і користувачів[4]);

- OpenTMS (приклад відкритої системи, яка приваблює можливістю безкоштовного використання, але вимагає технічної експертизи для налаштування[5]).

1.2.1 SAP Transportation Management

SAP Transportation Management (SAP TM) – це спеціалізований програмний продукт, створений компанією SAP для управління складними транспортними процесами у великих організаціях (рисунок 1.3). Система дозволяє координувати всі етапи логістичного циклу: від створення замовлення на перевезення до остаточного розрахунку вартості доставки та аналітичного супроводу.

Ключовою особливістю SAP TM є її глибока інтеграція з іншими модулями SAP ERP, такими як SAP S/4HANA, SAP EWM, SAP GTS тощо. Це забезпечує наскрізний контроль над усім логістичним ланцюгом, починаючи від зберігання вантажу на складі й завершуючи його доставкою клієнту.

У функціональному плані SAP TM надає широкі можливості:

- автоматизоване планування маршрутів із врахуванням обмежень

(вантажопідйомність, час доставки, особливості вантажу);

- розрахунок вартості транспортування на основі тарифних сіток;
- управління автопарком і сторонніми перевізниками;
- моніторинг у реальному часі та оповіщення при відхиленнях від маршруту;
- аналітичні звіти щодо ефективності перевезень.

Водночас, попри очевидні переваги, система має і ряд обмежень, особливо в контексті її впровадження в країнах із менш розвиненою логістичною інфраструктурою або в середовищі малого бізнесу. По-перше, SAP TM – це платформа з високою вартістю ліцензії. Її придбання та обслуговування є економічно доцільним переважно для великих корпорацій із глобальним масштабом діяльності. По-друге, налаштування SAP TM вимагає участі висококваліфікованих фахівців – SAP-консультантів, які мають сертифікацію й досвід у впровадженні подібних систем. Це значно збільшує загальні витрати на запуск. Крім того, адаптація системи до специфіки конкретного підприємства потребує часу і часто супроводжується необхідністю змін у внутрішніх бізнес-процесах.

Для локального ринку, зокрема в українських реаліях, SAP TM є занадто "важкою" платформою: її складна архітектура, англомовний інтерфейс та потреба в серйозних апаратних ресурсах роблять її малопридатною для підприємств малого та середнього бізнесу. У таких випадках доцільніше використовувати легші, більш адаптивні системи з відкритим кодом або локалізовані SaaS-рішення.

Рисунок 1.3 – Загальний конвейєр дій у типовій TMS

1.2.2 Oracle Logistics Cloud

Oracle Logistics Cloud (рисунок 1.4) – це програмний продукт корпоративного рівня, який входить до складу Oracle Cloud SCM і орієнтований на повну цифровізацію та автоматизацію логістичних процесів у великих організаціях. Його головна особливість – це реалізація у вигляді SaaS-платформи, що означає розгортання та використання без потреби у власному серверному обладнанні.

Цей сервіс надає компаніям змогу планувати, керувати та контролювати процеси перевезення вантажів на локальному та міжнародному рівнях. Він об'єднує дані з різних джерел (ERP, CRM, сторонні API), дозволяючи формувати єдину інформаційну картину для менеджерів і логістів.

Серед ключових можливостей Oracle Logistics Cloud варто відзначити:

- автоматизоване управління замовленнями та поставками з використанням адаптивних сценаріїв;

- інтеграцію з глобальними логістичними операторами (FedEx, DHL, UPS тощо);
- інтелектуальну маршрутизацію з урахуванням термінів, витрат і доступності транспорту;
- аналітичні панелі та прогнози, побудовані на базі Oracle Analytics;
- можливість контролю доставки в реальному часі;
- підтримку електронного документообігу й комплаєнсу з міжнародними стандартами.

Основна перевага Oracle Logistics Cloud полягає в тому, що вона надається як сервіс через інтернет, що значно спрощує доступ до функціональності для компаній з віддаленими офісами або підрозділами. Проте така зручність має і свою ціну. Платформа орієнтована насамперед на середній та великий бізнес, а вартість підписки, консультаційних послуг та інтеграції з іншими системами може стати серйозною фінансовою перешкодою для невеликих компаній.

Крім того, для повноцінного впровадження Oracle Logistics Cloud зазвичай потрібно адаптувати внутрішні бізнес-процеси компанії до шаблонів, що використовуються в системі. Це означає не лише витрати на технічну реалізацію, а й на навчання персоналу, оновлення корпоративної стратегії роботи з логістикою та постійне технічне супроводження.

Незважаючи на це, для великих організацій, які працюють у міжнародному середовищі та мають високі вимоги до прозорості логістики, Oracle Logistics Cloud є потужним інструментом, що дозволяє суттєво підвищити ефективність і конкурентоспроможність. Платформа також підтримує мобільний доступ, що дозволяє керівникам логістичних підрозділів контролювати процеси безпосередньо з планшетів або смартфонів. Це особливо актуально в умовах динамічного середовища та необхідності прийняття рішень у дорозі.

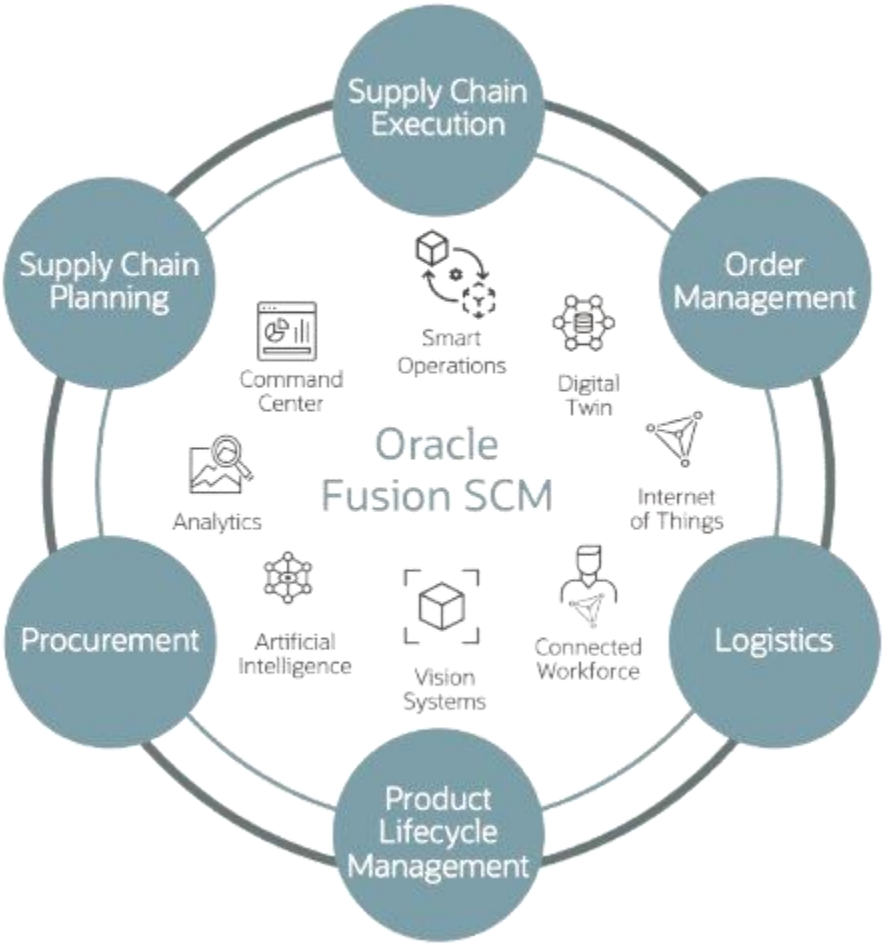


Рисунок 1.4 – Схема складових Oracle Logictics Cloud

1.2.3 Descartes TMS

Descartes Transportation Management System (TMS) – це одна з провідних логістичних платформ, орієнтованих на міжнародні перевезення та інтеграцію з митними службами. Продукт розроблено канадською компанією Descartes Systems Group, яка спеціалізується на побудові рішень для глобального ланцюга постачань (рисунок 1.5).

Головна відмінність цієї системи полягає в тісній взаємодії з митними органами різних країн, що дозволяє автоматизувати обмін даними, прискорити проходження процедур оформлення вантажів та зменшити ризики, пов'язані з бюрократичними бар'єрами. Завдяки цим можливостям Descartes TMS широко використовується логістичними операторами, які

працюють на трансконтинентальному рівні.

Серед основних можливостей платформи:

- підтримка електронного декларування товарів;
- інтеграція з міжнародними службами відстеження вантажів;
- можливість автоматичного розрахунку мит та податків;
- зберігання історії переміщень та повного аудиту документів;
- контроль відповідності митному та торговельному законодавству;
- інтерфейси для взаємодії з великими перевізниками та постачальниками.

Descartes TMS функціонує як модульна хмарна система, що дозволяє компаніям обирати тільки ті компоненти, які необхідні саме їм. Це забезпечує певну гнучкість у конфігурації та скорочує час на впровадження. Проте, попри всі ці переваги, система має і свої обмеження. Зокрема, через фокус на глобальних операціях, Descartes TMS часто виявляється надмірно складною та непрактичною для компаній, що працюють переважно в межах одного регіону або країни. Її функціонал може виявитися надлишковим для підприємств, які не займаються міжнародною логістикою, а інтеграція з локальними сервісами (наприклад, українськими картографічними або складськими рішеннями) може вимагати значних технічних доопрацювань. Також слід зазначити, що більшість документації та інтерфейсів Descartes TMS англомовні, що може ускладнити процес навчання персоналу в компаніях, де не всі працівники володіють іноземною мовою і деякі функції можуть бути недоступними без додаткової оплати або активації. Крім цього, для ефективної роботи системи необхідне стабільне інтернет-з’єднання та відповідне технічне забезпечення. Усе це варто брати до уваги ще на етапі планування впровадження.

Рисунок 1.5 – Descrates TMS

1.2.4 Transporeon

Transporeon – це сучасна хмарна платформа управління логістикою, яка реалізує концепцію екосистемної взаємодії між учасниками ланцюга постачання. Система надає цифровий простір, де вантажовідправники, перевізники, 3PL-оператори та отримувачі вантажу можуть спільно працювати над плануванням, виконанням і аналізом перевезень (рисунок 1.6).

Основна ідея Transporeon полягає у створенні відкритої логістичної мережі, де компанії мають змогу обирати партнерів, проводити тендери на транспортування, відстежувати статус доставки, вести документацію та аналізувати показники ефективності в єдиному цифровому середовищі.

Ключові функціональні можливості платформи включають:

- тендерний модуль, що дозволяє автоматизувати вибір перевізника;
- інструменти для відстеження вантажів у режимі реального часу;
- систему сповіщень і погоджень для зміни умов доставки;
- інтеграцію з ERP-системами замовників;

- аналітичні панелі для контролю витрат і ефективності логістики;
- електронний документообіг із підписами, відповідно до вимог ЄС.

Завдяки веб-орієнтованій архітектурі, Transporeon не потребує встановлення програмного забезпечення – доступ здійснюється через браузер, що є додатковою зручністю для мобільних користувачів і регіональних офісів.

Однак ефективність використання цієї платформи значною мірою залежить від наявності широкої мережі партнерів, які також є учасниками Transporeon-екосистеми. Якщо компанія має обмежену кількість перевірених перевізників або не взаємодіє з великим числом клієнтів, функціональність платформи може залишитися частково незадіяною.

Ще однією особливістю є залежність від зовнішньої цифрової інфраструктури: у разі перебоїв в інтернеті або недоступності сервісу користувачі можуть втратити доступ до критично важливої інформації.

Для великих європейських логістичних компаній, які мають стабільну партнерську базу, Transporeon є потужним інструментом для централізації процесів і підвищення прозорості співпраці. Проте для нових або невеликих бізнесів, які ще не мають широкої мережі контактів у сфері перевезень, платформа може виявитися надмірною або навіть обмежувальною в користуванні.

Крім зазначеного, варто звернути увагу на рівень технічної підтримки, який надається користувачам. Для новачків у сфері цифрової логістики може знадобитися додаткове навчання персоналу. Також слід враховувати, що частина функцій платформи може бути недоступною без підключення додаткових модулів. Це може вплинути на загальну вартість користування системою, тому бажано ретельно оцінити потреби компанії перед впровадженням.

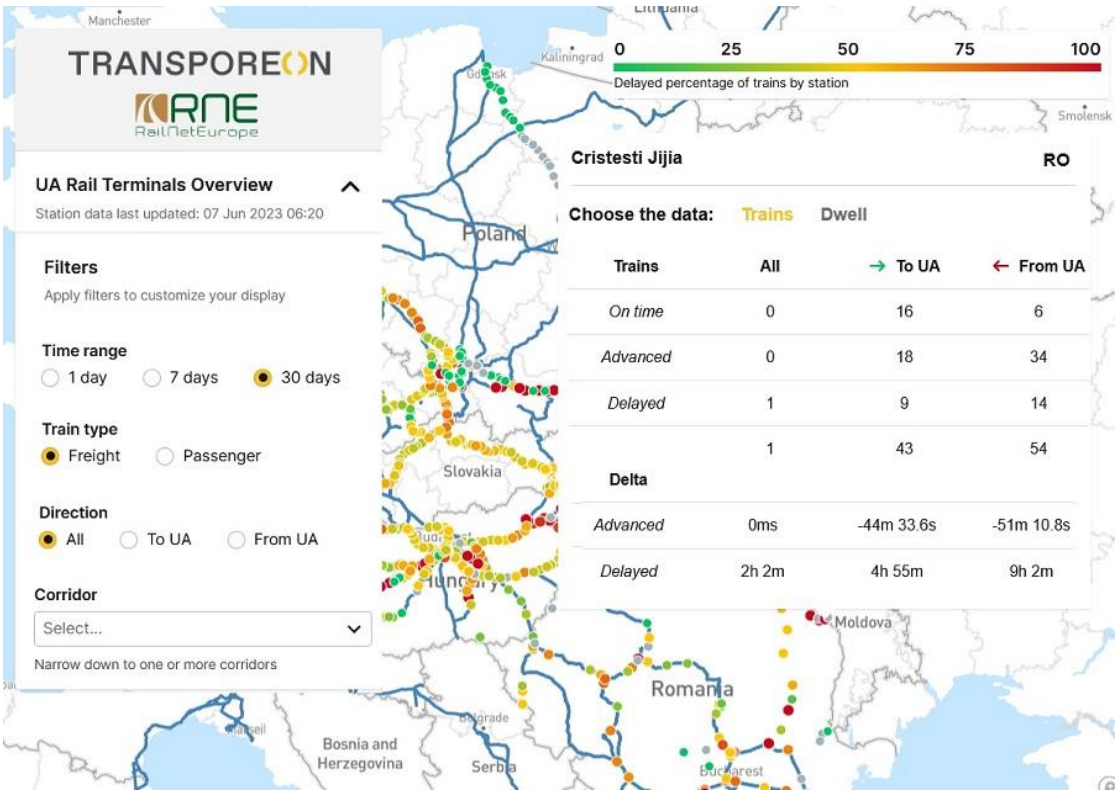


Рисунок 1.6 – Зовнішній вигляд додатку Transporeon

1.2.5 OpenTMS

OpenTMS – це відкрита система управління транспортною логістикою, яка розповсюджується під ліцензією open-source. Вона створена як альтернатива дорогим комерційним TMS-рішенням і орієнтована на команди розробників та підприємства, які мають власний IT-ресурс для адаптації та подальшого розвитку системи.

Основна перевага OpenTMS полягає в повній відкритості коду, що дозволяє модифікувати функціонал під конкретні бізнес-потреби. Завдяки цьому користувачі отримують змогу будувати індивідуальні модулі: від базового управління замовленнями до інтеграції з власними CRM або ERP-рішеннями. Серед доступного функціоналу:

- керування маршрутами, транспортними засобами та замовленнями;
- збереження історії перевезень і трекінг вантажів;
- підтримка багатокористувацького середовища з правами доступу;
- API для підключення зовнішніх сервісів і мобільних застосунків;

- можливість зберігання та експорту логістичних звітів.

OpenTMS використовує типові сучасні технології, такі як Java, MySQL/PostgreSQL, REST API, що робить її сумісною з більшістю серверних платформ. Це дозволяє легко розгортати систему як на локальних серверах, так і в хмарному середовищі.

Однак разом із перевагами, пов’язаними з гнучкістю та відсутністю витрат на ліцензію, OpenTMS має низку суттєвих обмежень:

- відсутність професійної технічної підтримки, що ускладнює вирішення критичних проблем у виробничому середовищі;
- високий поріг входу для користувачів без досвіду у програмуванні;
- складність налаштування інтерфейсу та бізнес-логіки без участі розробника;
- відсутність зручної документації українською мовою що може створити труднощі в розумінні архітектури рішення.

У зв’язку з цим OpenTMS є доцільним вибором переважно для невеликих, але технічно грамотних компаній або стартапів, які прагнуть зекономити на комерційних продуктах, водночас зберігаючи контроль над внутрішньою IT-інфраструктурою. Для великих підприємств або компаній, які не мають в штаті технічної команди, впровадження OpenTMS може стати викликом через відсутність готових модулів і складну криву навчання.

Також варто зазначити, що останнім часом зростає інтерес до OpenTMS з боку невеликих компаній та окремих розробників. У мережі з’являється дедалі більше прикладів налаштування, відкритих модулів та користувацьких доповнень, які можуть стати в пригоді новим користувачам. Доступ до коду відкриває можливості для створення власних рішень або оптимізації вже наявних функцій. Для багатьох це шанс самостійно вибудувати зручну систему під свої потреби без додаткових витрат. Це вимагає більше часу й технічних зусиль, але натомість забезпечує повну незалежність і контроль над логістичними процесами (табл. 1.1).

Таблиця 1.1 – Порівняльна характеристика аналогічних систем управління логістикою

Назва системи	Тип ліцензії	Цільова аудиторія	Ключові переваги	Основні недоліки
SAP Transportation Management	Комерційна (висока вартість)	Глобальні корпорації	Глибока інтеграція з ERP, повний контроль над логістичним циклом	Дороге впровадження, потреба у консультантах, складність для малого бізнесу
Oracle Logistics Cloud	Комерційна (SaaS)	Міжнародні компанії середнього та великого масштабу	Хмарна архітектура, розширена аналітика, висока масштабованість	Високі витрати на інтеграцію, потреба в адаптації процесів, навчання персоналу
Descartes TMS	Комерційна	Оператори міжнародної логістики	Митна інтеграція, підтримка глобальних перевезень, хмарна модульність	Слабка локалізація, англомовний інтерфейс, надмірність для локального бізнесу
Transporeon	Комерційна/підписка	Європейський середній та великий бізнес	Екосистема для співпраці, тендери, real-time трекінг, електронний	Залежність від кількості підключених партнерів, потреба в цифровій зрілості компанії

Продовження таблиці 1.1

1	2	3	4	5
			документооб ігг	
OpenTMS	Open Source (безкошто вна)	Невеликі компанії з технічним IT-відділом	Повна кастомізація, відкрита архітектура, гнучке розгортання	Потребує програмістів, відсутність офіційної підтримки, складність для новачків

1.3 Постановка задачі

На основі аналізу функціональності сучасних логістичних систем, а також виявлення їхніх обмежень для малого та середнього бізнесу, було сформульовано технічні вимоги до створення веб-застосунку, орієнтованого на автоматизацію основних процесів вантажоперевезень.

Метою є розробка клієнт-серверної інформаційної системи, яка дозволить логістичній компанії ефективно керувати процесами перевезення, зберіганням даних та взаємодією між учасниками перевезень. Система повинна забезпечити:

- створення, редагування та перегляд замовлень на перевезення;
- реєстрацію та управління водіями, транспортними засобами та клієнтами;
- побудову маршрутів з урахуванням координат пунктів відправлення та доставки;
- відображення поточного статусу доставки та історії замовлень;
- авторизацію користувачів з розмежуванням прав доступу (адміністратор, брокер, диспетчер, водій);

- базову інформацію по доставках (кількість, виконання, середній час).

Система має бути реалізована як клієнт-серверний застосунок. Для обробки запитів має використовується серверна частина на базі Java Spring Boot, а зберігання інформації здійснюється в реляційній базі даних PostgreSQL. Автентифікація користувачів виконується через AWS Cognito.

1.3.1 Сценарії використнння

Програмний засіб повинен підтримувати такі основні сценарії використовування:

1. Перший запуск. Перед тим як запустити програму потрібно створити адміністратора через інструмент керування HTTP запитами Postman. Після цього вже можна увійти як адміністратор і керувати даними, наприклад зареєструвати нового водія.

2. Розподілення ролей. Так як адміністратор добавив декілька користувачів із різними ролями, з’являється можливість використання програми диспетчерами, брокерами, водіями. В залежності від ролі з’являється відповідна сторінка з даними, наприклад у брокера буде панель із керування своїх вантажів у диспетчера буде список вантажів оголошених брокером та перелік водіїв, яким можна ці вантажі запропонувати. У водія буде сторінка з перевезеними вантажами та їх статусами.

2 ВИКОРИСТОВУВАНІ ТЕХНОЛОГІЇ

Проєкт реалізовано з використанням мови програмування Java, фреймворку Spring Boot, бази даних MySQL, а також архітектурного шаблону MVC. Для розробки застосовано середовище IntelliJ IDEA, для збірки – систему Maven. Зберігання користувачів реалізовано через хмарний сервіс AWS Cognito. Контроль версій здійснювався за допомогою системи Git.

2.1 Мова програмування Java

Java – це сучасна об’єктно-орієнтована мова програмування, яка вже понад два десятиліття є одним із лідерів у сфері розробки програмного забезпечення. Її розробила компанія Sun Microsystems у 1995 році (нині належить Oracle), і вона зарекомендувала себе як універсальна мова для створення високонадійних, масштабованих та безпечних застосунків [6].

Однією з ключових переваг Java є її платформонезалежність, що реалізується завдяки принципу «write once, run anywhere». Код, написаний мовою Java, компілюється в байт-код, який виконується у віртуальній машині Java (JVM). Це дозволяє запускати застосунок на будь-якій платформі, де встановлена JVM, без потреби в перекомпіляції коду. Такий підхід надзвичайно актуальний для корпоративних проєктів, які мають розподілену інфраструктуру або працюють у хмарних середовищах.

У межах розробки логістичного веб-застосунку Java була основною мовою для реалізації всієї бекенд-логіки. Зокрема:

- побудовано систему REST-контролерів для обробки HTTP-запитів;
- реалізовано об’єктну модель даних (Entity-класи);
- налаштовано сервіси для обробки логістичних сценаріїв: додавання замовлення, пошук водіїв, маршрутизація;

- створено DTO-структури для обміну даними між клієнтом і сервером;

- оброблялись сценарії валідації, обліку помилок, логування та кешування.

Java також відзначається сильною підтримкою об’єктно-орієнтованих принципів програмування (ООП), таких як наслідування, інкапсуляція, поліморфізм. Це дозволяє структурувати програму у вигляді ієрархій класів, які легко розширюються й повторно використовуються, що дуже зручно в проектах середнього та великого масштабу.

Суттєвим плюсом є і те, що Java має величезну екосистему бібліотек і фреймворків, що дозволяє не «винаходити велосипед», а користуватись готовими й перевіреними рішеннями. Зокрема, у нашому проєкті активно використовувався Spring Boot, який повністю побудований на Java і є її найпопулярнішим фреймворком для створення корпоративних веб-застосунків.

Крім того, завдяки вбудованому збирачу сміття (Garbage Collector), Java забезпечує автоматичне управління пам’яттю, що суттєво знижує ризики витоків пам’яті, що є критично важливим для стабільності довготривалих веб-систем, які постійно перебувають у роботі.

Переваги Java в контексті проєкту:

- Стабільність: ідеально підходить для довготривалих веб-застосунків;

- Безпека: розширені механізми контролю доступу, типізації та обробки помилок;

- Масштабованість: легке додавання нових функцій, API, об’єктів даних;

- Активна спільнота: тисячі безкоштовних бібліотек, форумів, гайдлайнів;

- Сумісність з хмарними сервісами (AWS, Google Cloud, Azure);

- Підтримка DevOps-практик: легко інтегрується з CI/CD інструментами, тестовими фреймворками (JUnit, Mockito).

Java була обрана в якості основної технології завдяки її перевіреній надійності, простоті розгортання у хмарних середовищах та повній сумісності зі всією екосистемою Spring – ключовим компонентом реалізованої логістичної платформи.

2.2 Середовище розробки IntelliJ IDEA

IntelliJ IDEA (рисунок 2.1) – одне з найпоширеніших середовищ розробки серед Java-програмістів, створене компанією JetBrains [7]. Воно поєднує інтелектуальне автозаповнення коду, зручну навігацію, підсвічування помилок у реальному часі та повну інтеграцію з Maven, Git і Spring Boot.

IDE забезпечила швидку роботу над проєктом завдяки:

- підтримці архітектури MVC;
- прямій роботі з базою MySQL;
- live templates і рефакторингу;
- автоматичному імпорту залежностей;
- гарячому перезапуску (DevTools);
- інтеграції з GitHub і JUnit.

Завдяки цьому IntelliJ IDEA стала основним інструментом у всіх етапах реалізації: від написання коду до тестування і розгортання.

Рисунок 2.1 – Логотип середовища IntelliJ IDEA

2.3 Система збірки Maven

Maven – це популярний інструмент автоматизації збірки проєктів на мові Java [8], який дозволяє ефективно керувати залежностями, створювати структуру застосунку та забезпечувати контроль за процесом компіляції й тестування. Він спрощує процес розробки, особливо у великих проєктах, де важливо забезпечити стабільність версій бібліотек і логіку складання всього програмного продукту.

У межах проєкту Maven використовувався для:

- підключення необхідних залежностей (Spring Boot Starter, Lombok, MySQL Driver тощо) через файл pom.xml;
- підтримки сталої структури проєкту відповідно до загальноприйнятих Java-стандартів;
- керування фазами збірки, включно з компіляцією, тестуванням і пакуванням у формат .jar;
- автоматичного запуску юніт-тестів під час кожної збірки для перевірки цілісності застосунку.

Крім того, Maven дозволив інтегрувати проєкт із зовнішніми репозиторіями залежностей, забезпечив повторюваність та переносимість середовища розробки, що значно підвищило надійність і передбачуваність процесу складання логістичного сервісу.

2.4 СУБД MySQL

У проєкті використовується MySQL – реляційна СУБД з відкритим кодом, яка забезпечує надійне зберігання інформації про користувачів, замовлення, транспорт і маршрути. Вона сумісна з Java-технологіями та добре працює зі Spring Data JPA, що значно спрощує роботу з ORM [9].

MySQL було обрано за такі переваги:

- висока швидкодія при обробці запитів;

- підтримка складної реляційної структури;
- простота у налаштуванні;
- безкоштовна ліцензія;
- стабільна робота у середовищі Spring Boot.

Система дозволяє зручно реалізовувати CRUD-операції, будувати зв’язки між таблицями та підтримує масштабування під майбутні потреби системи.

2.5 Spring Boot Framework

Spring Boot – це розширення фреймворку Spring, що спрощує розробку веб-застосунків у середовищі Java. Він дозволяє швидко запускати проєкти з мінімальною конфігурацією завдяки автоматичному налаштуванню та вбудованому веб-серверу[10].

У межах проєкту Spring Boot використовувався для реалізації REST API, через яке виконувались ключові логістичні дії: створення замовлень, призначення транспорту, реєстрація водіїв, пошук маршрутів та інше. Платформа дозволила структурувати додаток за шаблоном MVC, розділивши логіку, дані та інтерфейс API.

Використані модулі:

- Spring Web – реалізація REST-контролерів і обробка запитів;
- Spring Data JPA – доступ до MySQL через репозиторії;
- Spring Security [12] + AWS Cognito – захист маршрутів і авторизація користувачів;
- Spring Boot DevTools – гаряча перезагрузка застосунку під час розробки;
- Spring Validation – валідація введених користувачем даних.

Фреймворк забезпечив швидку розробку, модульність і простоту масштабування проєкту.

2.6 Хмарний сервіс AWS Cognito

Для зберігання користувачів та реалізації системи аутентифікації використовується AWS Cognito – безпечний та масштабований сервіс від Amazon [11]. Завдяки інтеграції через Spring Security, була реалізована підтримка авторизації, а також управління правами доступу до логістичних функцій (адміністратор, диспетчер, водій).

2.7 Шаблон архітектури MVC

Під час розробки проєкту було застосовано архітектурний підхід Model-View-Controller (MVC) [14], який дав змогу логічно розподілити відповідальність між різними частинами системи. Завдяки цьому вдалося уникнути змішування бізнес-логіки, обробки запитів і роботи з даними в одному місці, що позитивно вплинуло на підтримку коду та спрощення розробки.

Модель відповідає за структуру та зберігання даних, зокрема реалізовані сутності, що відображають об’єкти системи (користувачі, замовлення, маршрути). Також через DTO-об’єкти передаються потрібні дані між шарами без надмірної залежності від структури бази.

Контролери працюють із вхідними HTTP-запитами, формують маршрут до потрібної частини логіки та повертають результат клієнту. У свою чергу, сервісна частина виконує основні дії – перевірку, обробку та взаємодію з моделлю.

Завдяки використанню MVC структура проєкту вийшла чіткою, масштабованою та придатною до подальшого розширення без порушення цілісності системи.

2.8 Система контролю версій Git

У процесі розробки веб-системи для логістики використовувалась система контролю версій Git [13], яка є стандартом у сучасній розробці програмного забезпечення. Git надає широкі можливості для відстеження змін, спільної роботи над кодом та організації зручного циклу розробки. Репозиторій проєкту було розміщено на платформі GitHub, що забезпечило постійний доступ до коду з будь-якого місця, зручне резервне копіювання та наочний контроль за розвитком проєкту.

Однією з ключових переваг Git є можливість створення гілок (branches) – незалежних копій основного коду, в яких можна реалізовувати нову функціональність без ризику порушити основну логіку системи. У межах проєкту активно використовувалися гілки для розділення етапів розробки: реалізації контролерів, сервісної логіки, інтеграції з базою даних та налаштування авторизації. Це дозволило зосереджено працювати над окремими частинами застосунку, поступово інтегруючи їх у загальну структуру.

Git також забезпечує зручну систему коментарів до комітів, що дозволяє документувати зміни у зрозумілій формі, вказуючи, що саме було реалізовано чи змінено. Це полегшує розуміння історії проєкту, особливо при тривалому циклі розробки або залученні нових учасників до команди.

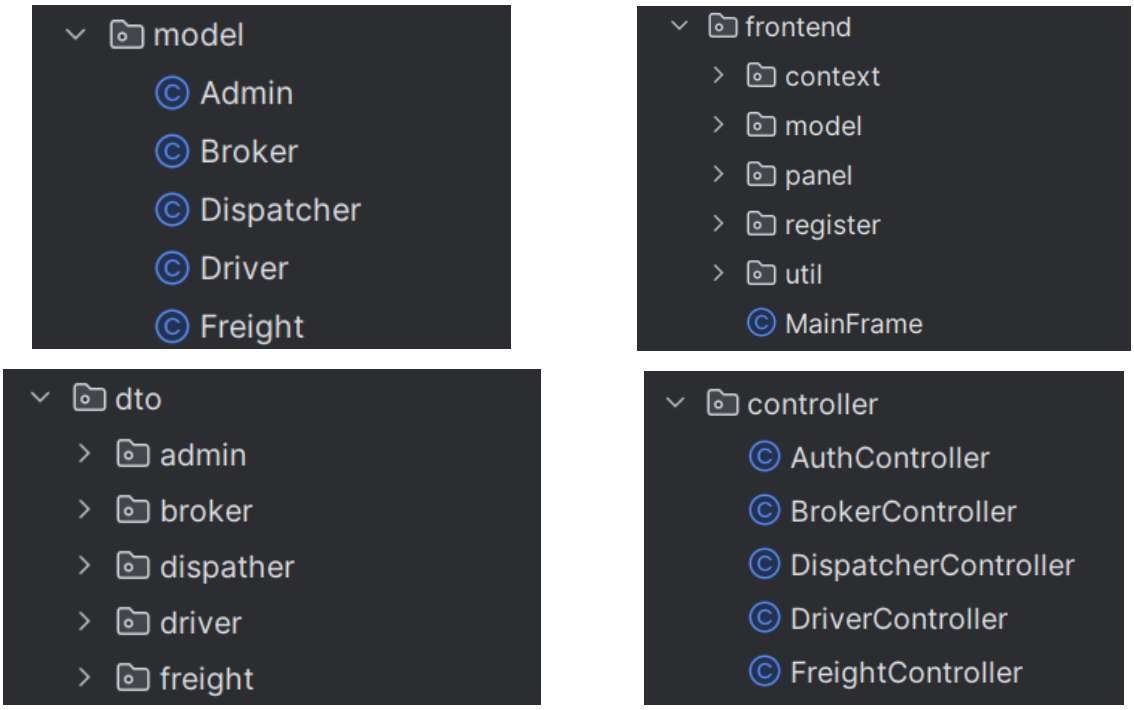
GitHub, як хостинг-платформа для Git-репозиторіїв, надає можливість створювати pull requests – запити на злиття змін, що дозволяє перевіряти код перед об'єднанням. Це сприяє підвищенню якості проєкту навіть у разі індивідуальної розробки, оскільки дозволяє структурувати внесення змін, тестування та перевірку.

Таким чином, Git запезпечив надійне керування версіями, підтримку чіткого робочого процесу та збереження історії змін. Завдяки цьому проєкт був реалізований поетапно, з можливістю безпечно повертатись до стабільних етапів і контролювати всі внесені модифікації.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

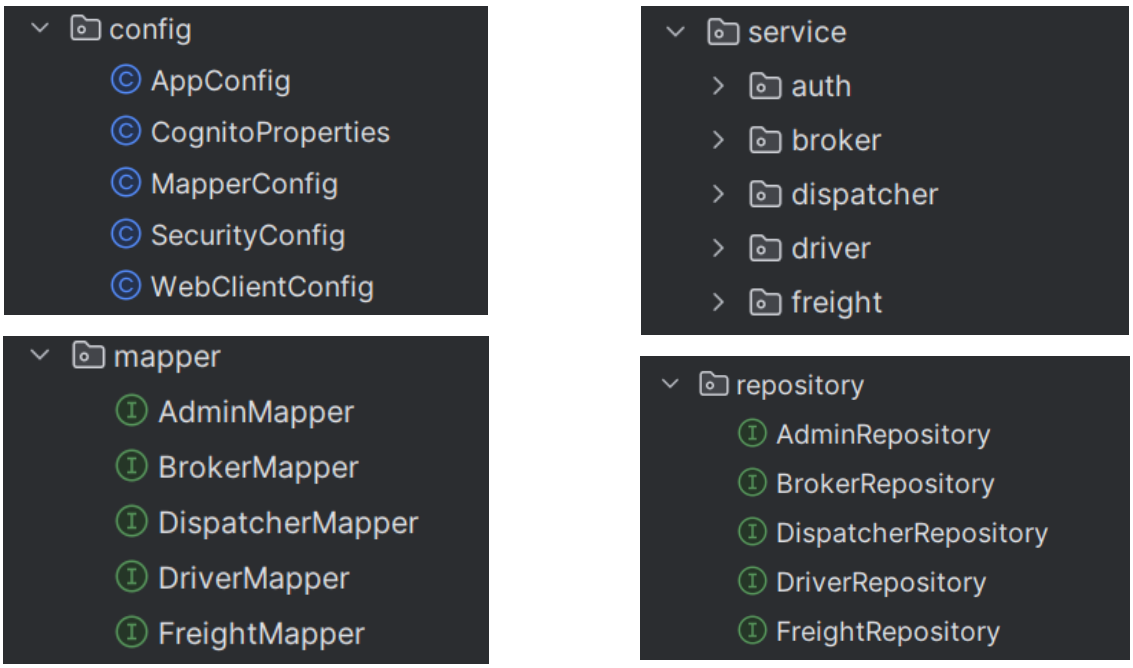
3.1 Структура проекту

Проект створено за шаблоном архітектури MVC (Model-View-Controller) з розширенням на Service Layer (рівень сервісів), який передбачає розділення всієї системи на три основні частини: модель (Model), яка відповідає за дані; (View) – представлення, тобто інтерфейс для користувача. Усі ці компоненти працюють окремо, а взаємодію між ними забезпечує контролер (Controller) (рисунок 3.1), який звертається до рівня сервісу (рисунок 3.2) та координує роботу між частинами.



а) б)

Рисунок 3.1 – Файли проекту: а) модель даних і DTO об’єкти; б) вікна інтерфейсу користувача та контролер;



в) г)

Рисунок 3.2 – в) конфігурація та маппери для гнучкої роботи з DTO; г) рівень сервісу та репозиторіїв

Виконання програми починається у класі LogisticsApplication у головному методі main(String[] args) у класі, який слугує точкою входу у Spring Boot додатках. Спочтаку встановлюється системна властивість, яка дозволяє запуск графічного інтерфейсу Swing. Потім завантажуються змінні середовища із файлу .env за допомогою бібліотеку .dotenv. Після запускається Spring Boot застосунок і створюється контекст, у якому зберігаються всі компоненти додатку та графічний інтерфейс у відповідному потоці GUI (AWT/Swing) (лістинг 3.1).

Лістинг 3.1 – Поетапна ініціалізація модулів під час запуску розробленого програмного додатку

```
@SpringBootApplication
public class LogisticsApplication {
    public static void main(String[] args) {
       System.setProperty("java.awt.headless", "false");

       Dotenv dotenv = Dotenv.load();
       dotenv.entries().forEach(entry ->
System.setProperty(entry.getKey(), entry.getValue()));
```

```
        ConfigurableApplicationContext context =
SpringApplication.run(LogisticsApplication.class, args);

        SwingUtilities.invokeLater(() -> {
            MainFrame mainFrame =
context.getBean(MainFrame.class);
            mainFrame.setVisible(true);
        });
    }
}
```

3.1.1 Модель даних

Основними сутностями додатку є користувачі, які мають особисті права і взаємодіють із системою, а також модель вантажу, що відображає логістику перевезень. Кожен з користувачів представляє сутність і таблицю у базі даних та має ім’я, пароль, email, id. Зокрема сутність вантаж має наступні властивості: місце завантаження, вивантаження, максимальна відстань до завантаження та відстань, яку потрібно проїхати водієві.

Усі класи реалізовані як JPA-сутності, які позначені анотацією @Entity, що дозволяє зберігати об'єкти певної моделі в базі даних, а також використовують анотацію @Data з бібліотеки Lombok для автоматичної генерації допоміжних методів.

Сутність Admin відповідає адміністраторам системи, які мають найвищий рівень доступу до функціоналу програми та мають найважливішу роль у програмі, наприклад створення нових користувачів (адміністратори створюються через базу даних або інструмент, який може надсилати запити на створення). Модель зберігається у таблиці admins бази даних. У моделі передбачено такі поля: id – унікальний ідентифікатор адміністратора, який генерується автоматично; email – електронна пошта, що є унікальною та використовується як логін; name – ім’я адміністратора; password – пароль для авторизації; createdAt – дата та час створення запису. Останнє поле заповнюється автоматично перед збереженням за допомогою методу onCreate, позначеного анотацією @PrePersist (лістинг 3.2).

Лістинг 3.2 – Сутність Admin

```
@Data
@Entity
@Table(name = "admins")
public class Admin {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true)
    private String email;

    @Column(nullable = false)
    private String name;

    @Column(nullable = false)
    private String password;

    @Column(name = "created_at", nullable = false, updatable =
false)
    private LocalDateTime createdAt;

    @PrePersist
    protected void onCreate() {
        this.createdAt = LocalDateTime.now();
    }
}
```

Сутність Broker відповідає брокерам – користувачам, які займаються оформленням перевезень і виступають посередниками між замовниками та виконавцями. Вони створюють заявки на перевезення, заповнюють інформацію про вантаж та передають її далі в систему. Модель зберігається у таблиці brokers бази даних. Зокрема хотілося б виділити наступні поля: mc – номер у системі Motor Carrier; cognitoSub – унікальний ідентифікатор користувача з AWS Cognito. Крім того, модель містить зв’язок з іншою сутністю freight через поле freights – це список усіх вантажів, які належать брокеру. Зв’язок реалізований через анотацію @OneToMany. Каскадна поведінка дозволяє автоматично видаляти або оновлювати вантажі при зміні брокера, а параметр orphanRemoval=true забезпечує видалення "осиротілих" записів (лістинг 3.3).

Лістинг 3.3 – Сутність Broke

```
@Data
@Entity
@Table(name = "brokers")
public class Broker {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String company;
    private String phoneNumber;
    private String email;
    private Long mc;
    private String password;
    private String cognitoSub;

    @OneToMany(mappedBy = "broker",
            cascade = CascadeType.ALL,
            orphanRemoval = true,
            fetch = FetchType.LAZY)
    private List<Freight> freights;
}
```

Сутність Dispatcher відповідає диспетчерам системи – користувачам, які відповідають за організацію та контроль процесу перевезення вантажів. Вони приймають заявки від брокерів, призначають водіїв і координують виконання доставки. Модель зберігається у таблиці dispatchers бази даних. Також присутнє особливе поле rate – тарифна ставка, яку диспетчер пропонує водієві (лістинг 3.4).

Лістинг 3.4 – Сутність Dispatcher

```
@Data
@Entity
@Table(name = "dispatchers")
public class Dispatcher {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private Long mc;
    private String company;
    private String phoneNumber;
    private String email;
    private String password;
    private BigDecimal rate;
}
```

Сутність driver відповідає водіям, які виконують перевезення вантажів у системі. Водій отримує призначені йому замовлення, виконує доставку та сповіщує диспетчера про статус. Модель зберігається у таблиці drivers бази даних. Зазначені деяйкі поля як location – поточне розташування водія; rate – тарифна ставка за яку згоден водій перевезти вантаж (лістинг 3.5).

Лістинг 3.5 – Сутність Driver

```
@Table(name = "drivers")
public class Driver {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String phoneNumber;
    private String email;
    private String password;
    private String truck;
    private String location;
    private BigDecimal rate;
}
```

Сутність Freight відповідає вантажам, які підлягають перевезенню в системі. Цей об’єкт створюється брокером і використовується для опису всіх параметрів доставки. Модель зберігається у таблиці freights бази даних. У структурі передбачено такі поля: id – унікальний ідентифікатор вантажу, що генерується автоматично; pickUpAddress – адреса завантаження; deliveryAddress – адреса доставки; milesLoaded – кількість миль з вантажем; milesEmpty – кількість миль без вантажу; totalMiles – загальна відстань перевезення; rate – ставка на перевезення. Окрім цього, модель містить поле broker, яке є зовнішнім ключем до сутності broker і реалізоване через зв’язок @ManyToOne. Це означає, що кожен вантаж належить одному брокеру, який його створив. Завантаження зв’язаної сутності виконується ліниво (lazy), що підвищує продуктивність при вибірці даних (лістинг 3.6).

Лістинг 3.6 – Сутність Freight

```
@Data
@Entity
@Table(name = "freights")
public class Freight {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String pickUpAddress;
    private String deliveryAddress;
    private Integer milesLoaded;
    private Integer milesEmpty;
    private Integer totalMiles;
    private Integer rate;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "broker_id", nullable = false)
    private Broker broker;
}
```

3.1.2 Рівень контролерів

Спочатку виконується аутентифікація у додатку, яка реалізована через контролер AuthController. Цей клас відповідає за обробку HTTP-запитів, пов’язаних із реєстрацією та входом користувачів до системи. У контролері передбачено окремі методи для кожного типу користувачів: адміністратора (лістинг 3.7), диспетчера (лістинг 3.8), водія (лістинг 3.9) та брокера (лістинг 3.10). Для кожної ролі реалізовано два основних методи – реєстрація та вхід у систему. Наприклад, методи register-admin та login-admin призначені для роботи з адміністраторами, а login-driver або register-broker – для відповідних ролей.

Лістинг 3.7 – Методи автентифікації адміністратора

```
@PostMapping("/register-admin")
public ResponseEntity<AdminResponseDto> registerAdmin(@Valid
@RequestBody AdminRequestDto requestDto) {
    return
ResponseEntity.ok(adminAuthService.registerAdmin(requestDto));
}

@PostMapping("/login-admin")
```

```
public ResponseEntity<AdminLoginResponseDto> loginAdmin(@Valid
@RequestBody AdminLoginRequestDto requestDto) {
    return
ResponseEntity.ok(adminAuthService.authenticateAdmin(requestDto)
);
}
```

Лістинг 3.8 – Методи автентифікації диспетчера

```
@PostMapping("/register-dispatcher")
public ResponseEntity<DispatcherResponseDto>
registerDispatcher(@Valid @RequestBody DispatcherRequestDto
requestDto) {
    return
ResponseEntity.ok(dispatcherAuthService.registerDispatcher(reque
stDto));
}

@PostMapping("/login-dispatcher")
public ResponseEntity<DispatcherLoginResponseDto>
loginDispatcher(@Valid @RequestBody DispatcherLoginRequestDto
requestDto) {
    return
ResponseEntity.ok(dispatcherAuthService.authenticateDispatcher(r
equestDto));
}
```

Лістинг 3.9 – Методи автентифікації водія

```
@PostMapping("/register-driver")
public ResponseEntity<DriverResponseDto> registerDriver(@Valid
@RequestBody DriverRequestDto requestDto) {
    return
ResponseEntity.ok(driverAuthService.registerDriver(requestDto));
}

@PostMapping("/login-driver")
public ResponseEntity<DriverLoginResponseDto> loginDriver(@Valid
@RequestBody DriverLoginRequestDto requestDto) {
    return
ResponseEntity.ok(driverAuthService.authenticateDriver(requestDt
o));
}
```

Лістинг 3.10 – Методи автентифікації брокера

```
@PostMapping("/register-broker")
public ResponseEntity<BrokerResponseDto> registerBroker(@Valid
@RequestBody BrokerRequestDto requestDto) {
    return
ResponseEntity.ok(brokerAuthService.registerBroker(requestDto));
```

```
}

@PostMapping("/login-broker")
public ResponseEntity<BrokerLoginResponseDto> loginBroker(@Valid
@RequestBody BrokerLoginRequestDto requestDto) {
    return
ResponseEntity.ok(brokerAuthService.authenticateBroker(requestDt
o));
}
```

На вхід методи приймають об’єкти типу DTO наприклад адміністратора (лістинг 3.11) або брокера (лістинг 3.12), які містять необхідні дані, зокрема електронну пошту, пароль, ім’я, компанію або інші параметри. Дані перевіряються на коректність за допомогою механізму валідації. Після цього запит передається у відповідний сервіс – наприклад, AdminAuthService або DriverAuthService, де виконується основна логіка обробки: перевірка наявності користувача, порівняння паролів або створення нового запису.

Лістинг 3.11 – Приклад класу AdminLoginRequestDto на вхід

```
@Data
public class AdminLoginRequestDto {
    @Email(message = "Invalid email format")
    @NotBlank(message = "Email is required")
    private String email;

    @NotBlank(message = "Password is required")
    private String password;

    public AdminLoginRequestDto() {
    }

    public AdminLoginRequestDto(String email, String password) {
        this.email = email;
        this.password = password;
    }
}
```

Лістинг 3.12 – Приклад класу BrokerRequestDto на реєстрацію

```
@Data
public class BrokerRequestDto {
    @NotNull
    private String company;
    @NotNull
    private String phoneNumber;
```

```
    @Email(message = "Invalid email format")
    @NotBlank(message = "Email is required")
    private String email;
    @NotBlank(message = "Password is required")
    @Size(min = 6, message = "Password must be at least 6
characters long")
    private String password;
    @NotNull(message = "MC is required")
    private Long mc;
}
```

У результаті користувач отримує відповідь у вигляді спеціального об’єкта з інформацією про результат операції – наприклад, токен доступу (лістинг 3.13) або дані про користувача (лістинг 3.14), які підтверджують успішну аутентифікацію. Такий підхід дозволяє централізовано керувати входом до системи та гарантує безпечну роботу з різними типами користувачів.

Лістинг 3.13 – Токен доступу

```
eyJraWQiOiJrZGtLZXlJZCIsInR5cCI6IkpXVCIsImFsZyI6IlJTMjU2In0.eyJz
dWIiOiJkaXNwYXRjaGVyQGV4YW1wbGUuY29tIiwidXNlcl9pZCI6IjEyMzQ1NiIs
InJvbGUiOiJkaXNwYXRjaGVyIiwiZXhwIjoxNzE3NjQ3NzAwLCJpYXQiOjE3MTc2
NDQxMDAsImlzcyI6ImxvZ2lzdGljcy1hcHAifQ.QlW6_vdQgN_XEkaJHrkgEdyU0
OYymdS_8HZMvlOUUDrs6IN6EnAOUv6uA0l1rKksswrOjqB4GEIFlQLxPYWAFZq29
yCQkk6FyY3I9bL4swhiSY8MfCRj_3a9vAcEkLVFXmghzXJWrYGxU4G2Y7tfA
```

Лістинг 3.14 – дані про користувача

```
  "id": 7,
  "name": "John Smith",
  "phoneNumber": "+380671234567",
  "email": "driver@example.com",
  "truck": "Volvo FH16",
  "location": "New York",
  "rate": 0.95
```

3.1.3 Рівень сервісів

Рівень сервісів у системі відповідає за реалізацію основної бізнес-логіки та взаємодію між контролерами, моделями, репозиторіями та зовнішніми сервісами. У структурі проєкту для кожного типу користувача:

адміністратора, диспетчера, водія та брокера, створено окремі сервіси, які реалізують відповідні інтерфейси. Наприклад, для адміністратора використовується сервіс AdminCognitoAuthService, який реалізує логіку реєстрації (лістинг 3.15) та автентифікації (лістинг 3.16) в Amazon Cognito, призначення ролі адміністратора, а також автентифікацію з отриманням токена доступу.

Усі необхідні залежності в сервісах впроваджуються автоматично за допомогою механізму Dependency Injection, що забезпечується Spring Framework. Завдяки цьому досягається зменшення зв'язності компонентів та спрощується їх тестування. У межах сервісу обробляються дані, перевіряється наявність користувача в системі, виконується взаємодія з зовнішніми сервісами, створюються об'єкти моделі та викликаються методи збереження у базі даних. Таким чином, сервісний рівень є центральним у побудові логіки роботи застосунку, ізолює контролери від прямої роботи з базою даних та сторонніми API, що дозволяє досягти чистої архітектури.

Лістинг 3.15 – Метод реєстраціїї адміністратора

```
@Override
@Transactional
public AdminResponseDto registerAdmin(AdminRequestDto
adminRequestDto) {
    if
(adminRepository.findByEmail(adminRequestDto.getEmail()).isPrese
nt()) {
        throw new RuntimeException("Admin with this email
already exists.");
    }

    AdminCreateUserRequest cognitoRequest =
AdminCreateUserRequest.builder()
            .userPoolId(cognitoProperties.getUserPoolId())
            .username(adminRequestDto.getEmail())
            .temporaryPassword(adminRequestDto.getPassword())
            .userAttributes(

AttributeType.builder().name("email").value(adminRequestDto.getE
mail()).build(),

AttributeType.builder().name("name").value(adminRequestDto.getNa
me()).build()
```

```
            )
            .messageAction(MessageActionType.SUPPRESS)
            .build();

    cognitoClient.adminCreateUser(cognitoRequest);

    AdminSetUserPasswordRequest setPasswordRequest =
AdminSetUserPasswordRequest.builder()
            .userPoolId(cognitoProperties.getUserPoolId())
            .username(adminRequestDto.getEmail())
            .password(adminRequestDto.getPassword())
            .permanent(true)
            .build();

    cognitoClient.adminSetUserPassword(setPasswordRequest);

    AdminAddUserToGroupRequest groupRequest =
AdminAddUserToGroupRequest.builder()
            .userPoolId(cognitoProperties.getUserPoolId())
            .username(adminRequestDto.getEmail())
            .groupName("ADMIN")
            .build();

    cognitoClient.adminAddUserToGroup(groupRequest);

    Admin admin = adminMapper.toModel(adminRequestDto);
    adminRepository.save(admin);

    return adminMapper.toDto(admin);
}
```

Лістинг 3.16 – Метод автентифікації адміністратора

```
@Override
public AdminLoginResponseDto
authenticateAdmin(AdminLoginRequestDto loginRequestDto) {
    Optional<Admin> admin =
adminRepository.findByEmail(loginRequestDto.getEmail());
    if (admin.isEmpty()) {
        throw new RuntimeException("Admin not found.");
    }

    Map<String, String> authParams = new HashMap<>();
    authParams.put("USERNAME", loginRequestDto.getEmail());
    authParams.put("PASSWORD", loginRequestDto.getPassword());

    String secretHash = calculateSecretHash(
            loginRequestDto.getEmail(),
            cognitoProperties.getClientId(),
            cognitoProperties.getClientSecret()
    );
    authParams.put("SECRET_HASH", secretHash);
```

```
    AdminInitiateAuthRequest authRequest =
AdminInitiateAuthRequest.builder()
            .userPoolId(cognitoProperties.getUserPoolId())
            .clientId(cognitoProperties.getClientId())
            .authFlow(AuthFlowType.ADMIN_USER_PASSWORD_AUTH)
            .authParameters(authParams)
            .build();

    AdminInitiateAuthResponse authResponse =
cognitoClient.adminInitiateAuth(authRequest);

    return new
AdminLoginResponseDto(authResponse.authenticationResult().access
Token());
}
```

3.1.4 Рівень доступу до даних

Рівень доступу до даних реалізовано за допомогою механізму Spring Data JPA. Для кожної сутності створено окремий репозиторій, який розширює базовий інтерфейс і надає набір стандартних CRUD операцій для роботи з базою даних. Наприклад, BrokerRepository відповідає за збереження та пошук даних для брокера у таблиці бази даних. Репозиторії дозволяють виконувати пошук за електронною поштою, зберігати нові записи, оновлювати або видаляти дані без необхідності створювати SQL-запити вручну (лістинг 3.17).

Лістинг 3.17 – Методи керування брокером без застосування SQL запитів

```
@Service
@AllArgsConstructor
public class BrokerServiceImpl implements BrokerService {
    private final BrokerRepository brokerRepository;
    private final BrokerMapper brokerMapper;

    @Override
    public List<BrokerResponseDto> findAll() {
        return brokerRepository.findAll()
                .stream()
                .map(brokerMapper::toDto)
                .toList();
    }

    @Override
```

```
    public BrokerResponseDto findById(Long id) {
        Broker broker =
brokerRepository.findById(id).orElseThrow(
                () -> new EntityNotFoundException("Can't find
broker by id: " + id)
        );
        return brokerMapper.toDto(broker);
    }

    @Override
    public BrokerResponseDto update(BrokerRequestDto requestDto)
{
        Broker broker =
brokerRepository.findByEmail(requestDto.getEmail()).orElseThrow(
                () -> new EntityNotFoundException("Dispatcher by
email: " + requestDto.getEmail() + ", doesn't exist"));

        Broker updatedBroker = brokerMapper.toModel(requestDto);
        updatedBroker.setId(broker.getId());

        Broker savedBroker =
brokerRepository.save(updatedBroker);
        return brokerMapper.toDto(savedBroker);
    }

    @Override
    public String deleteById(Long id) {
        brokerRepository.deleteById(id);
        return "Broker by id: " + id + " deleted";
    }
}
```

Доступ до репозиторіїв здійснюється безпосередньо в сервісах за допомогою автоматичного впровадження залежностей. Це дозволяє централізувати роботу з базою даних та дотримуватись принципів розділення відповідальностей. За потреби можна створювати власні запити або розширювати інтерфейси, що забезпечує гнучкість, зручність у підтримці та масштабованість програмного забезпечення (лістинг 3.18).

Лістинг 3.18 – JPA репозиторій з кастомними методами керування брокером

```
public interface BrokerRepository extends JpaRepository<Broker,
Long> {
    Optional<Broker> findByEmail(String email);
    Optional<Broker> findByCognitoSub(String cognitoSub);
}
```

3.2 Зберігання даних

Згідно з постановкою задачі, передбачається зберігання даних між сеансами використання програмного засобу. Не дивлячись на об’єм, структуру та взаємозв’язки даних у системі управління логістикою вантажоперевезень, було прийнято рішення використовувати повноцінну реляційну систему керування базами даних. Така база повинна забезпечувати надійне збереження, масштабованість, підтримку транзакцій і ефективну роботу з великим обсягом структурованих даних.

3.2.1 Використання PostgreSQL

У якості системи керування базами даних у проєкті було обрано PostgreSQL. Це сучасна об’єктно-реляційна СКБД з відкритим вихідним кодом, яка підтримує розширені типи даних, транзакційність, перевірку цілісності даних та гнучку систему індексації. PostgreSQL є оптимальним вибором для задач, які вимагають збереження складно пов’язаних об’єктів, таких як користувачі, вантажі, маршрути, ролі та їх взаємодії в логістичній системі.

Завдяки підтримці потужного механізму запитів (SQL + розширення), PostgreSQL дозволяє ефективно реалізовувати складні вибірки, фільтрацію, сортування та агрегацію даних. Крім того, підтримка транзакцій гарантує консистентність при паралельному доступі до бази з боку кількох користувачів.

На відміну від використання локальних форматів на зразок XML, що підходять лише для невеликих, слабо зв’язаних структур, PostgreSQL дозволяє централізовано керувати даними, забезпечує надійність, резервне копіювання та сумісність із різними середовищами розгортання. Це особливо важливо в багатокористувацьких веб-застосунках, де критичним є не тільки збереження даних, а й безпека, швидкодія та масштабованість (рисунок 3.3).

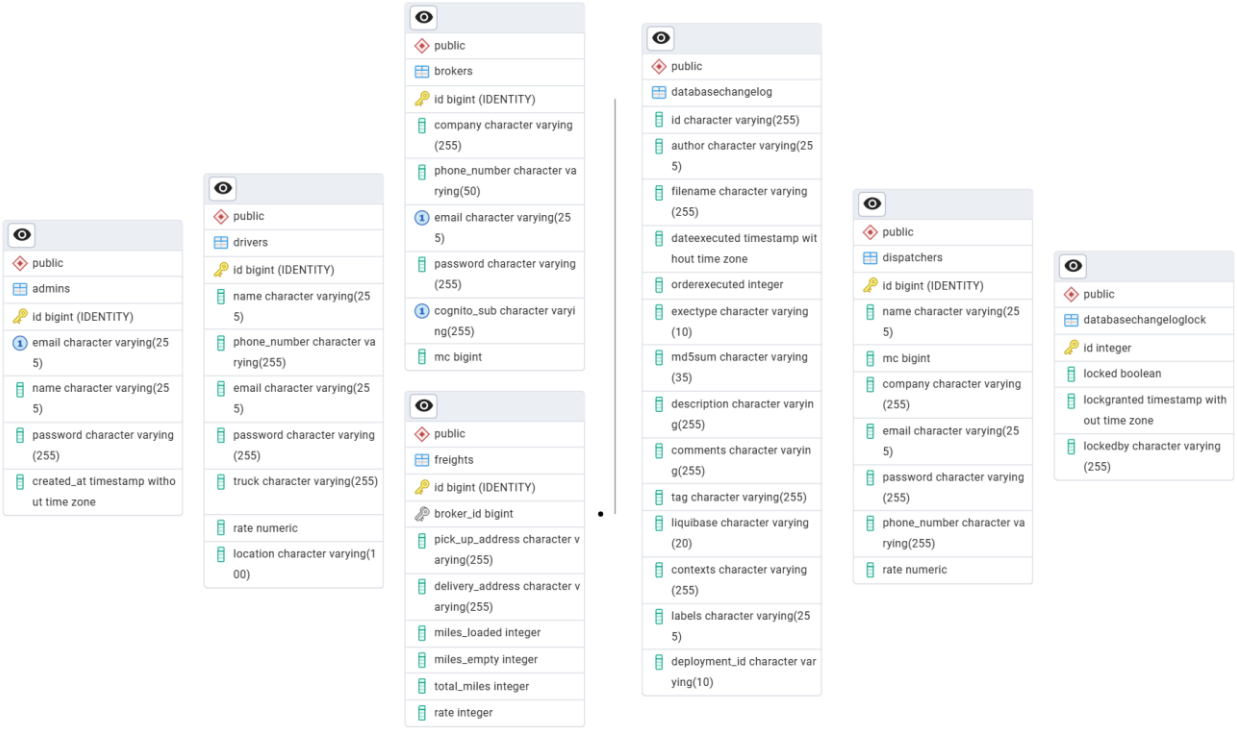


Рисунок 3.3 Схема бази даних

3.2.2 Використання AWS Cognito

У межах розробки програмного засобу було реалізовано механізм аутентифікації та авторизації користувачів із розподілом ролей. Дивлячись на вимоги до безпеки, масштабованості та інтеграції з іншими сервісами, було прийнято рішення використовувати хмарний сервіс AWS Cognito (рисунок 3.4).

Рисунок 3.4 – AWS Cognito logo

AWS Cognito – це керований сервіс компанії Amazon, який дозволяє створювати, автентифікувати та керувати користувачами у хмарному середовищі (рисунок 3.5). У рамках додатку Cognito використовується для централізованого управління обліковими записами адміністраторів, диспетчерів, водіїв і брокерів. Кожен тип користувача має свою роль, яка визначається через механізм груп у Cognito User Pool. Це дозволяє автоматично призначати права доступу та забезпечувати гнучке керування авторизацією (рисунок 3.6).

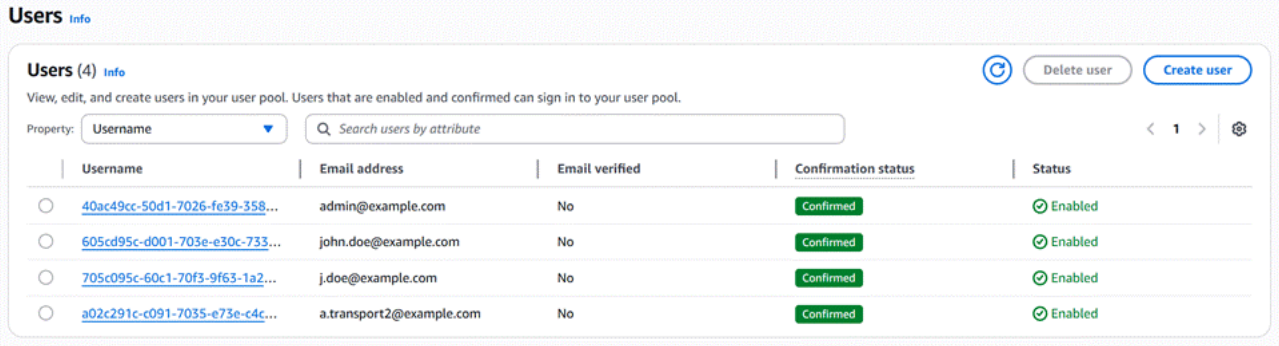


Рисунок 3.5 – AWS Cognito Users

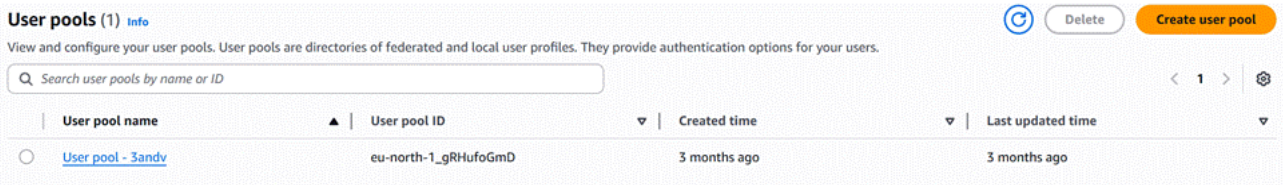
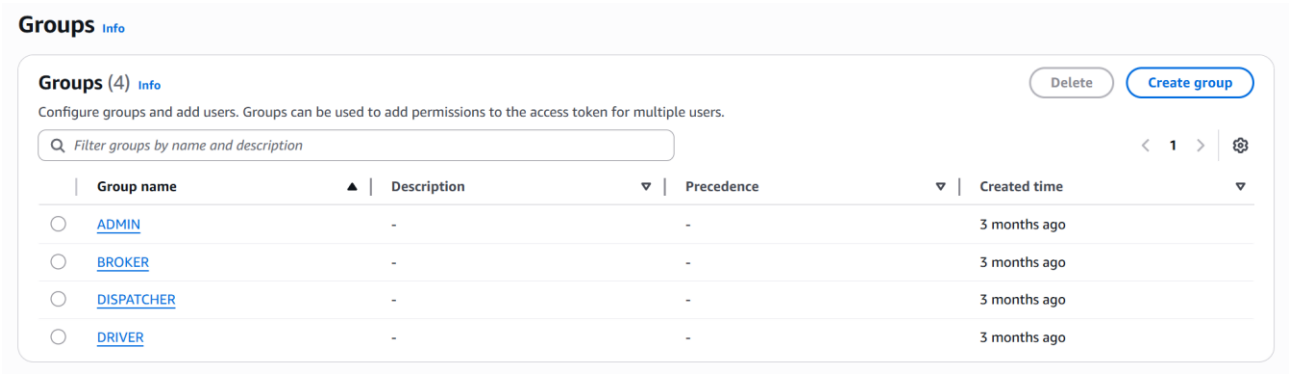
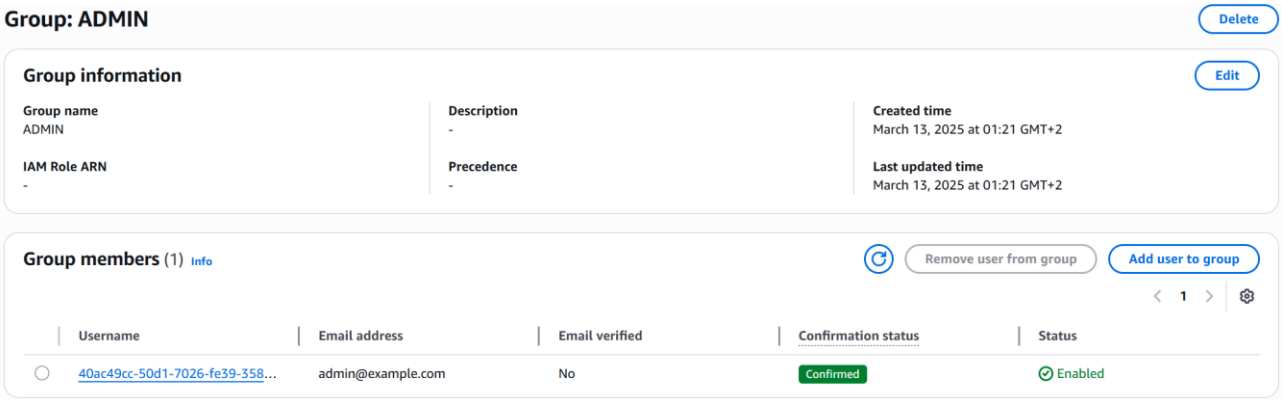


Рисунок 3.6 – AWS Cognito User Pool

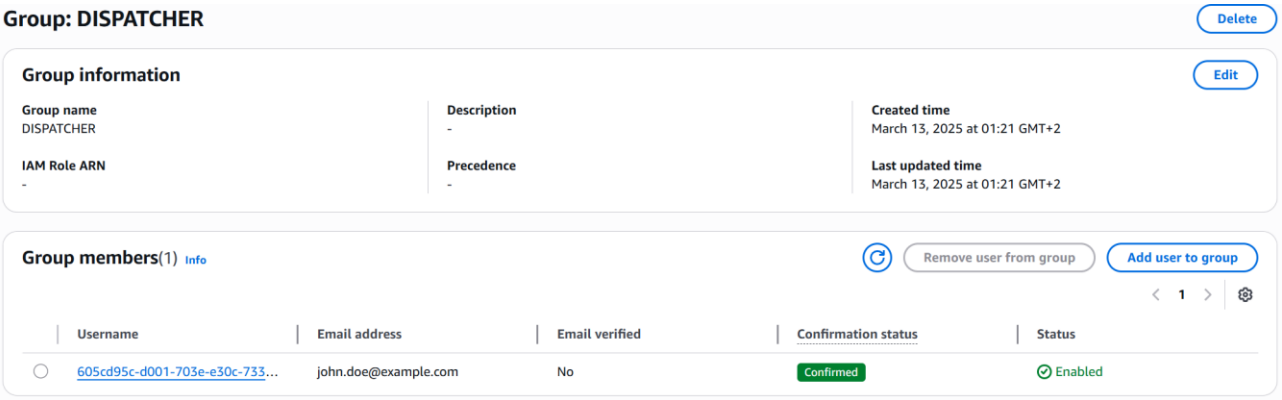
Під час реєстрації користувача, відповідний сервіс (наприклад, AdminCognitoAuthService) виконує створення облікового запису через AWS Cognito SDK, встановлює постійний пароль, та додає користувача до відповідної групи (рисунки 3.7). Усі операції з автентифікації виконуються через стандартний механізм ADMIN_USER_PASSWORD_AUTH, що дозволяє безпечно обробляти логіни та паролі. Після автентифікації користувач отримує токен доступу (JWT), який використовується для підтвердження особи під час виконання запитів до серверної частини.



а)



б)



в)

Рисунок 3.7 – а) AWS Cognito User Groups; б) AWS Cognito Group ADMIN
в) AWS Cognito Group DISPATCHER

4 ОСОБЛИВОСТІ ВИКОРИСТАННЯ ДОДАТКУ

4.1 Системні вимоги додатку

Для використання додатку необхідні:

а) Клієнтська частина (настільний застосунок на Java Swing):

1) операційна система: Windows 7 / 8 / 10 / 11, Linux або macOS;

2) встановлена Java Virtual Machine (JRE) версії 17 або вище;

3) оперативна пам’ять: не менше 512 МБ, рекомендовано 1 ГБ і більше;

4) вільне місце на диску: не менше 100 МБ;

5) підключення до Інтернету для взаємодії із сервером.

б) Серверна частина (Spring Boot API):

1) операційна система: Windows, Linux або macOS;

2) Java Development Kit (JDK) версії 17 або вище;

3) оперативна пам’ять: мінімум 1 ГБ, рекомендовано 2 ГБ;

4) вільне місце на диску: не менше 200 МБ;

5) підключення до мережі Інтернет;

6) встановлений PostgreSQL (локально);

7) наявність доступу до сервісу AWS Cognito (через API);

8) встановлена програма Postman.

4.2 Перший запуск додатку

При запуску програмного засобу буде запропоновано ввести email та password та вибрати роль користувача ADMIN, DISPATCHER, BROKER, DRIVER. Система автоматично розпізнає та перевіряє наявність існуючого користувача у базі даних та на стороні AWS Cognito. Якщо користувач не існує або неправильно обрана роль, у такому випадку виведеться

повідомлення, що користувач не існує і потрібно буде перевірити усі значення та обрати коректну роль (рисунок 4.1). Але потрібно зауважити те, що якщо в системі немає завчасно зареєстрованого адміністратора, то усі спроби виконати авторизацію будуть марні так як без зареєстрованого адміністратора не буде в наявності інших користувачів. У такому випадку потрібно використати інструмент керування REST API – Postman або базу даних з AWS Cognito. Вибираючи другий варіант вам як власнику додатку потрібно буде зареєструватися на стороні БД з AWS Cognito одночасно, так як вони між собою тісно пов’язані. У випадку Postman лише потрібно запустити додаток та надіслати запит на реєстрацію. Після цього з’явиться запис у базі даних та на стороні AWS Cognito про зареєстрованого адміністратора. І вже можна буде використовувати додаток з UI.

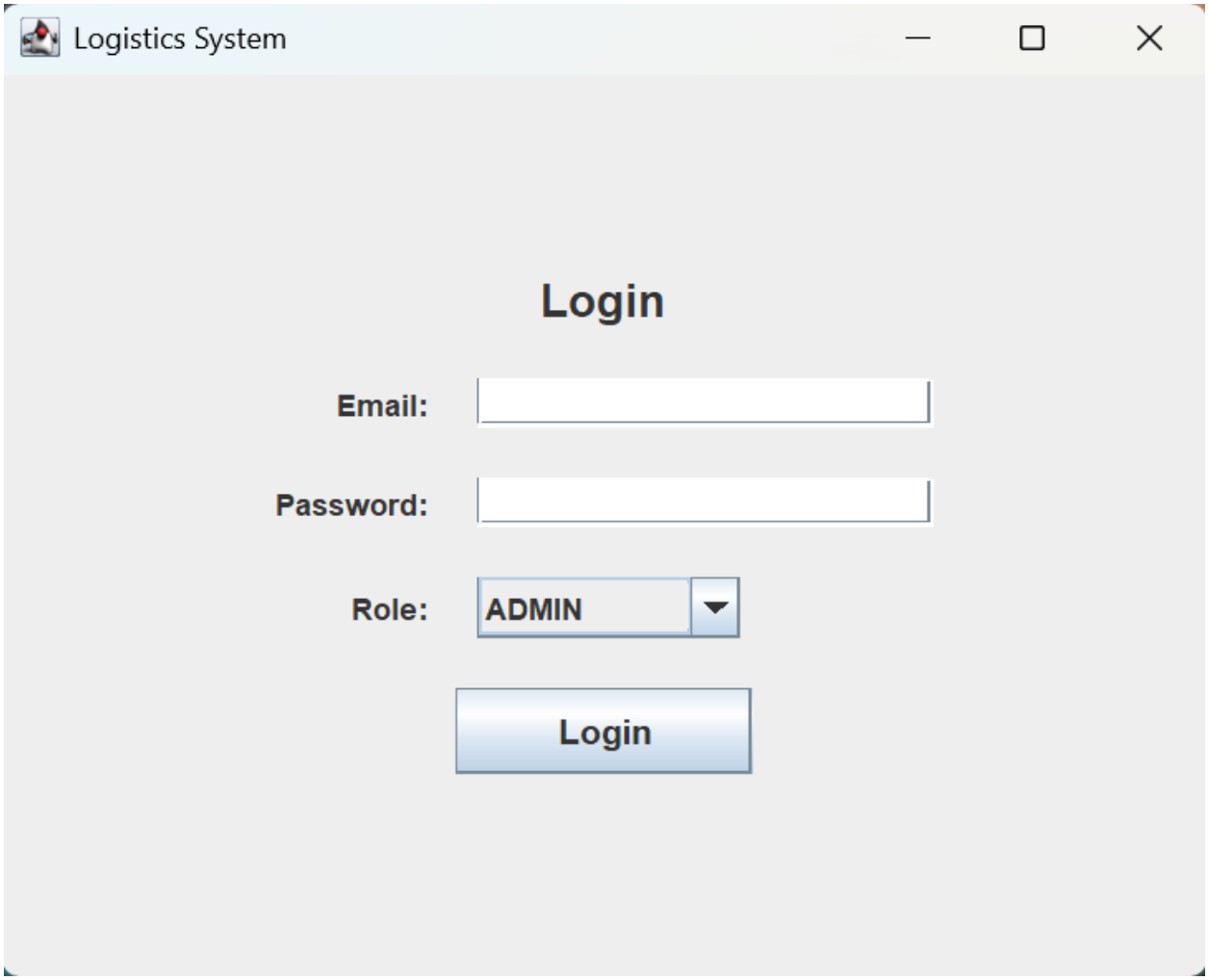


Рисунок 4.1 – Сторінка авторизації

Після введення даних буде відкрито сторінку відповідно обраної ролі (рисунок 4.2). У випадку адміністратора буде перекинуто на сторінку управління даними про вантажі та інформацією про користувачів, а також можливість повернутися до попередньої панелі. Кнопка з початком «Register» відповідає за реєстрацію користувачів, з початком «Manage» відповідає за управління вантажами та користувачами.

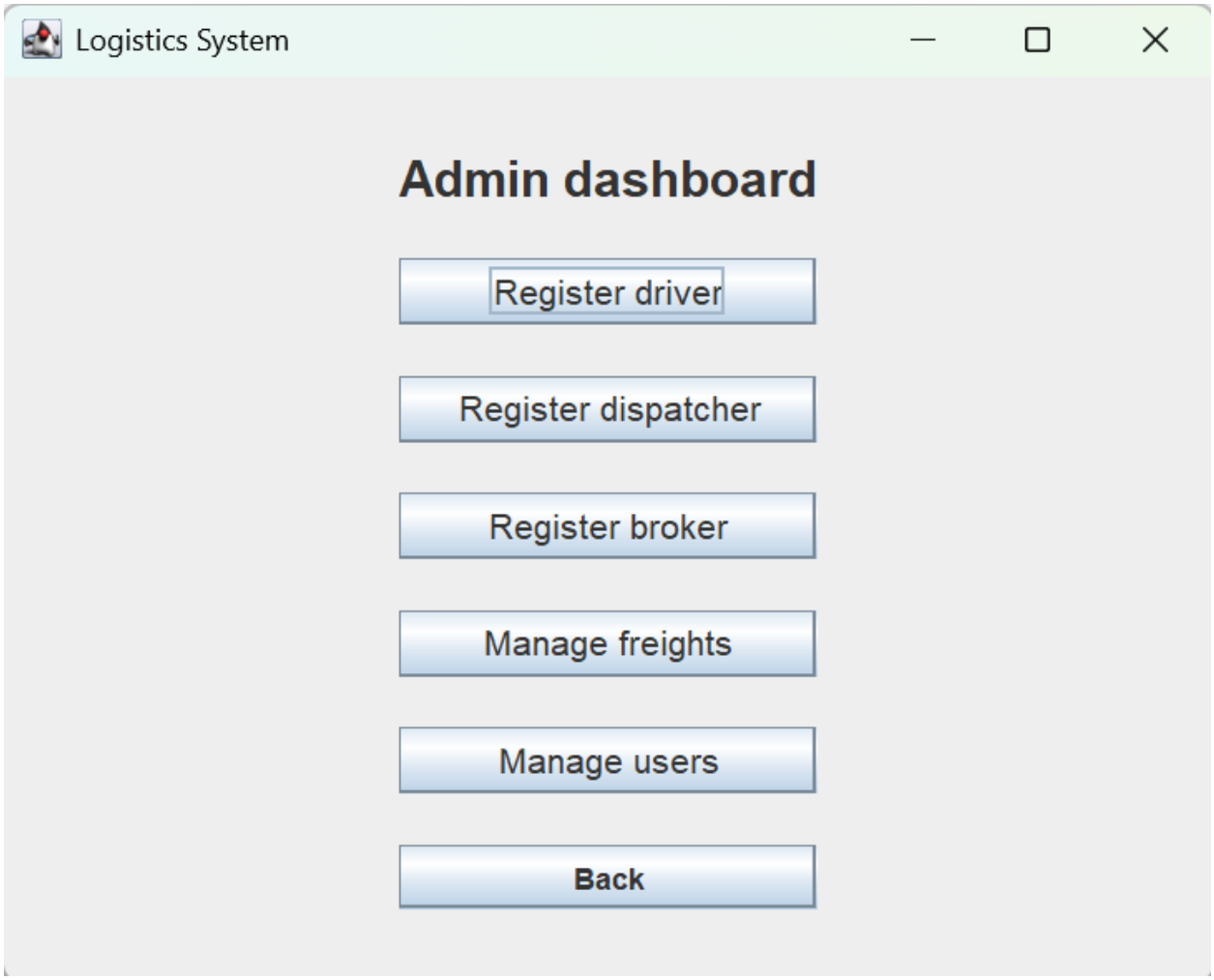


Рисунок 4.2 – Сторінка адміністратора

Натиснувши одну із кнопок відкриється відповідне вікно наприклад вікно з реєстрацією водія (рисунок 4.3). Зареєструвати нових користувачів можуть виключно адміністратори, оскільки система має бути захищена від несанкціонованого доступу, зловмисних дій та потенційного шахрайства.

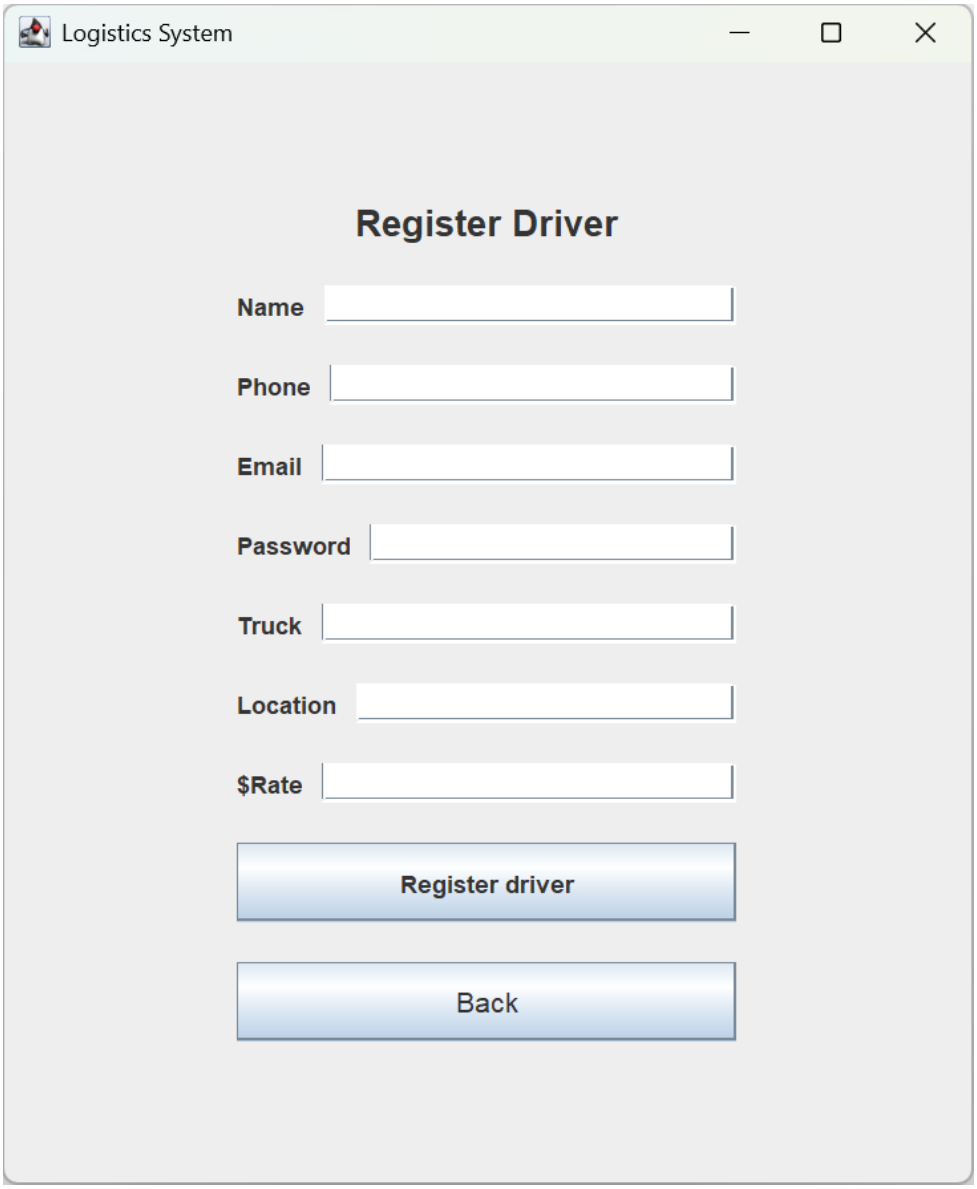


Рисунок 4.3 – Приклад панелі реєстрації водія

4.3 REST API

На сторінці адміністратора не зображено опції з реєстрацією адміністратора, бо вони створюються у базі даних або через REST API (рисунок 4.4), який керується наприклад програмою Postman. При успішній автентифікації надається токен від сервера (рисунок 4.5), який потрібно скопіювати, та передати у відповідний розділ Authentication, Auth Type, Bearer Token (рисунок 4.6).

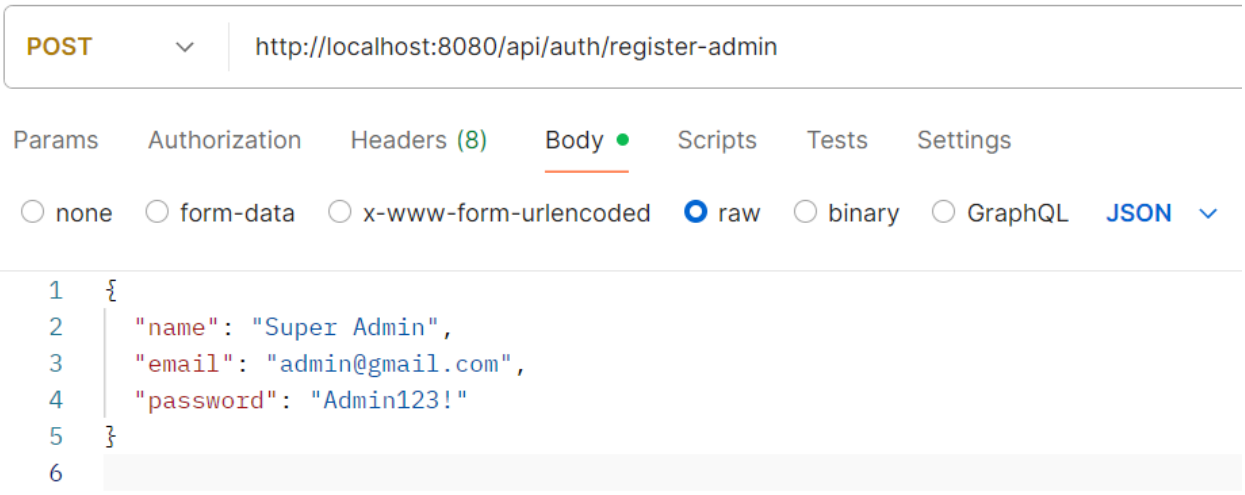


Рисунок 4.4 – Реєстрація адміністратора у Postman

Рисунок 4.5 – Токен від сервера після входу у систему

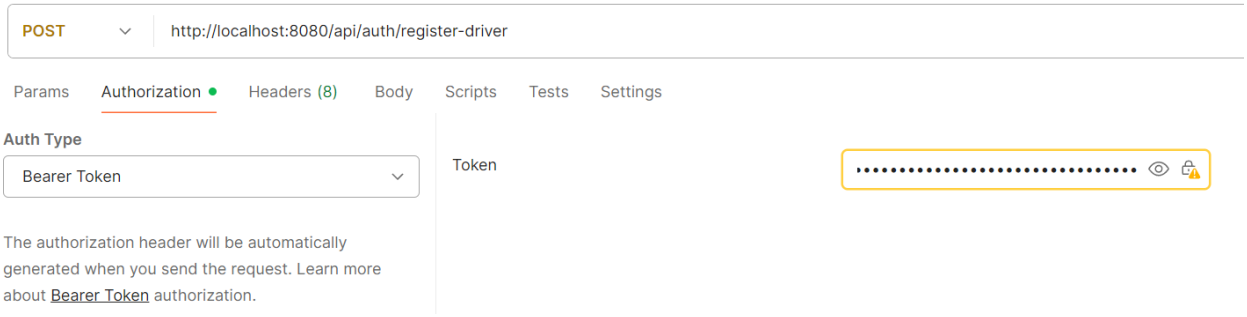


Рисунок 4.6 – Розділ передачі токену

4.4 Керування системою на стороні сервера

Є великий ряд можливостей наприклад знаходження брокера або вантажу за власним ідентифікатором для подальшого оновлення (рисунок 4.7). Це велике доповнення до системи і дуже зручно,

коли потрібно швидко перевірити чи змінити дані без зайвих кліків у UI. Також це корисно для розробників і адміністраторів, які тестують систему або працюють із запитами напряму та інтерфейс користувача не може бути використаний без зареєстрованого адміністратора. Завдяки цьому можна автоматизувати частину процесів, інтегрувати систему з іншими сервісами або працювати з нею навіть тоді, коли інтерфейс ще не готовий.

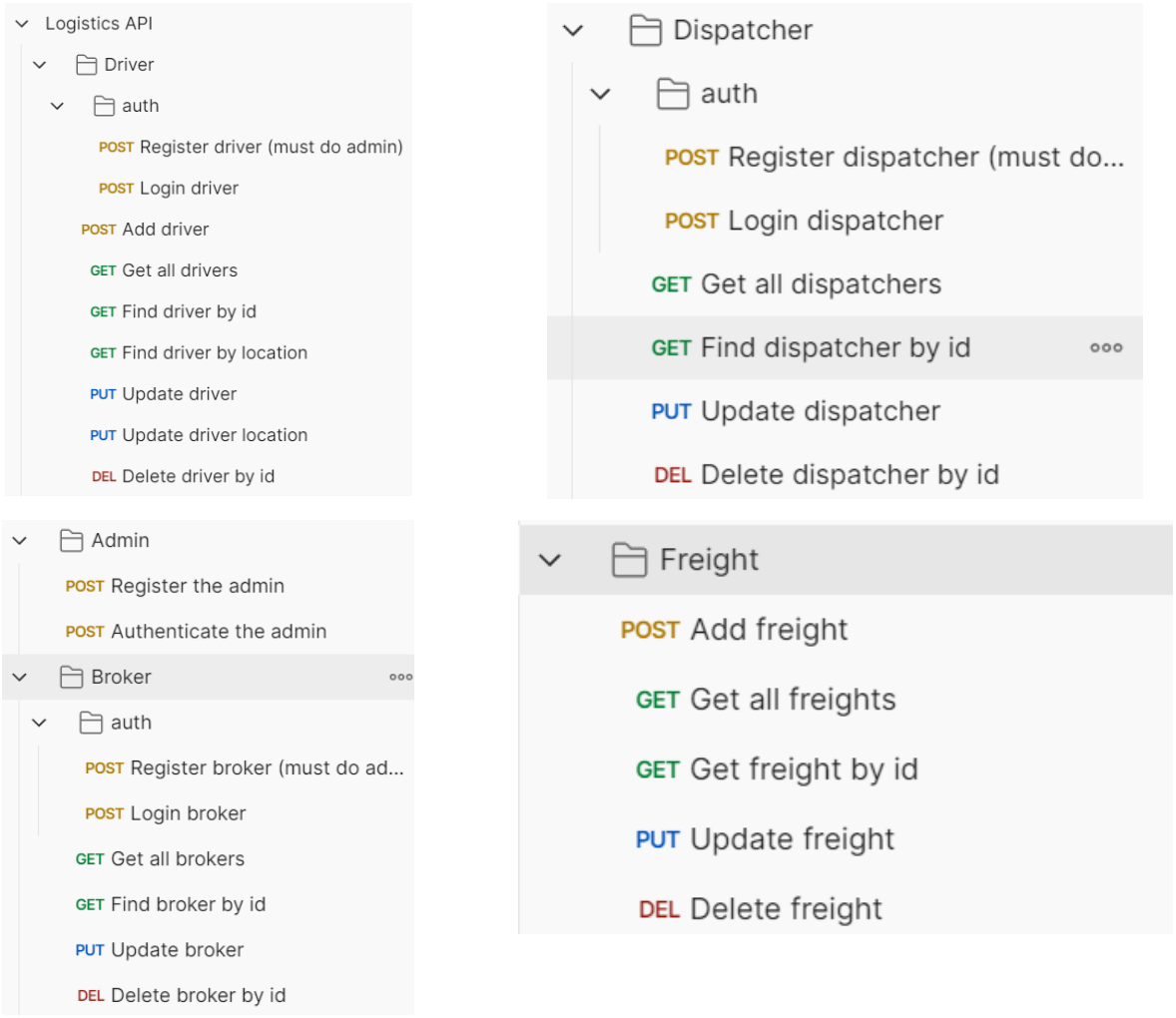


Рисунок 4.7 – Приклади запитів управління системою

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розроблено додаток на базі Java та Spring Boot, що відповідає вимогам сучасної цифрової логістики для автоматизації управління логістикою вантажоперевезень. Система підтримує рольову модель доступу, авторизацію через AWS Cognito, роботу з базою даних MySQL та організована відповідно до архітектурного шаблону MVC і дозволяє вирішувати типові задачі логістичних компаній, зокрема: створення та обробку замовлень, призначення водіїв і транспорту, збереження маршрутів, облік статусів доставки та взаємодію з користувачами через зручний REST API.

Виконані такі ключові завдання:

- проаналізовано предметну область та існуючі рішення;
- реалізовано структуру з використанням Java, Swing, Spring Boot, MySQL, Maven;
- впроваджено аутентифікацію через AWS Cognito;
- розроблено RESTFUL API для основних логістичних операцій;
- реалізовано модель збереження та обробки логістичних об'єктів;
- забезпечено модульність, масштабованість і простоту подальшої підтримки системи.

У процесі розробки було враховано специфіку галузі та потребу в цифровій трансформації логістичних процесів. У подальшому планується розробити інтерфейс для диспетчера, брокера й водія. Інтегрувати геолокаційні сервіси для побудови маршрутів у реальному часі. Впровадити модуль аналітики й статистики, підключити мобільний застосунок для водіїв та масштабувати інфраструктуру за допомогою AWS.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. SAP Transportation Management. https://www.sap.com/central-asia-caucasus/products/scm/transportation-logistics.html.
2. Oracle Logistics Cloud. https://www.oracle.com/scm/logistics/.
3. Descartes TMS. https://www.descartes.com/solutions/transportation-management/tms.
4. Transporeon. https://www.transporeon.com/en/company.
5. OpenTMS. https://github.com/fossabot/open-tms.
6. Sierra K. Head First Java. O'Reilly Media, Incorporated, 2005.
7. IntelliJ IDEA Essentials. Packt Publishing, 2014. 276 p.
8. Maven by Example / J. Casey et al. Lulu Press, Inc., 2010.
9. Nichter D. Efficient MySQL Performance. O'Reilly Media, Incorporated, 2021.
10. Learning Spring Boot. Packt Publishing, Limited, 2014.
11. Operations L. O. L. Aws : Fundamentals: Fundamentals. Logical Operations LLC, 2016.
12. Spilca L. Spring Security in Action. Manning Publications Co. LLC, 2020.
13. Git - Documentation. Git. URL: https://git-scm.com/doc
14. Späth P. Beginning Java MVC 1.0. Berkeley, CA : Apress, 2021.