Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: «Розробка програмного забезпечення для управління системою доставки вантажних перевезень з оптимізацією маршрутів»

Виконав: студент IV курсу групи 1ПІ-20б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Сіпалка І. С.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Ткаченко О. М.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. ОТ Богомолов С. В.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри Романюк О. Н.
«10» червня 2024 р.

ВНТУ – 2024

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
_____Романюк О.Н.
«12» березня 2024 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Сіпалкі Івану Сергійовичу

1. Тема роботи – «Розробка програмного забезпечення для управління системою доставки вантажних перевезень з оптимізацією маршрутів»

Керівник роботи: Ткаченко Олександр Миколайович, к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від 11 березня 2024 року № 80.

2. Строк подання студентом роботи 3 червня 2024 року

3. Вихідні дані до роботи: середовище розробки Intellij IDEA Community, фреймворк Spring Boot, високорівнева мова програмування Java, система управління базами даних PostgreSQL, Amazon Web Services.

4. Зміст розрахунково-пояснювальної записки: вступ, аналіз стану питання програмного забезпечення, порівняльний аналіз аналогів; аналіз методів розв'язання задачі, постановка задач дослідження, оптимізація методу пошуку найкоротшого маршруту, розробка прототипу інтерфейсу, розробка архітектури роботи серверної частини, розробка алгоритмів роботи системи, варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення, розробка бази даних, розробка алгоритму прокладання маршруту доставки, розробка

калькулятора для обчислення вартості доставки, метод зберігання та отримання даних про відстань між містами, тестування інтерфейсу, тестування бази даних, тестування алгоритму Дейкстри, висновки, список використаних джерел, додатки.

5. Перелік графічного матеріалу: титульний слайд; актуальність теми; мета, об'єкт та предмет дослідження; задачі дослідження, новизна, практична цінність одержаних результатів; схема алгоритму прокладання маршруту доставки, схема реєстрації та авторизації, архітектура серверної частини, тестування, висновки, апробація результатів роботи і публікації.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Ткаченко О.М., к.т.н., доцент кафедри ПЗ	13.03.24	03.06.24

7. Дата видачі завдання 13 березня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз розвитку програмного забезпечення для управління системою доставки вантажних перевезень	14.03.2024 – 24.03.2024	виконано
2	Розробка структури та алгоритмів програмного продукту	25.03.2024 – 07.04.2024	виконано
3	Варіантний аналіз і обґрунтування вибору засобів реалізації програмного забезпечення	08.04.2024 – 18.04.2024	виконано
4	Розробка системи	19.04.2024 – 19.05.2024	виконано
5	Тестування системи	20.05.2024 – 24.05.2024	виконано
6	Оформлення матеріалів до захисту БДР	01.06.2024 – 03.06.2024	виконано

Студент _____ (підпис) Сіпалка І.С. (прізвище та ініціали)

Керівник бакалаврської дипломної роботи _____ (підпис) Ткаченко О.М. (прізвище та ініціали)

Анотація

У бакалаврській дипломній роботі розроблено систему для управління доставкою вантажів з використанням сучасних технологій. Програмні модулі дозволяють забезпечити ефективний механізм моніторингу та контролю за перевезеннями. Систему розроблено з урахуванням потреб користувачів у зручному інструменті для відстеження вантажів та оптимізації логістики. Було створено необхідні моделі, які визначають основні концепції та принципи функціонування системи. Розроблено алгоритм прокладання найоптимальнішого маршруту доставки, розроблено базу даних, метод зберігання та отримання даних про відстань між містами та калькулятор вартості доставки.

Застосунок розроблено з використанням мов програмування Java та фреймворку Spring. Організацію зберігання даних забезпечено за допомогою реляційної бази даних PostgreSQL.

Такий підхід дозволяє створити систему, яка відповідає сучасним стандартам і вимогам до систем управління доставкою вантажів, та забезпечить оптимальний контроль над перевезеннями для користувачів усіх рівнів.

Abstracts

In the bachelor's thesis developed a system for managing cargo delivery using modern technologies. The software modules provide an effective mechanism for monitoring and controlling transportation. The system was developed taking into account the needs of users for a convenient tool for tracking cargo and optimizing logistics. The necessary models were created to define the basic concepts and principles of the system. An algorithm for laying the most optimal delivery route was developed, a database, a method for storing and retrieving data on the distance between cities, and a delivery cost calculator were developed.

The application was developed using Java programming languages and the Spraing framework. Data storage is organized using the PostgreSQL relational database.

This approach makes it possible to create a system that meets modern standards and requirements for cargo delivery management systems and provides optimal control over transportation for users of all levels.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. У сучасному світі ефективне управління доставкою вантажів набуває все більшої важливості для підприємств у зв'язку зі зростанням обсягів вантажопотоків та збільшенням вимог до швидкості та надійності доставки. Відповідно, розробка та впровадження систем управління доставкою стають актуальною задачею для багатьох компаній. У результаті глобалізації, діджиталізації та розвитку, як міжнародної так і національної торгівлі попит на транспортні послуги значно зріс. На протязі останньої декади перемістився в основному на автомобільний транспорт через його переваги, такі як легка доступність, гнучкість операцій і надійність. Таким чином, автомобільна логістика вважається одним із найвпливовіших факторів економічного розвитку країни [1].

Автомобільний транспорт найефективніше використовувати для перевезення невеликих вантажів на короткі відстані. Крім того, це важливий вид транспорту на початку та в кінці мультимодального транспортного ланцюга [2].

Багато наявних рішень на ринку не відповідають повністю потребам підприємств, що спеціалізуються на вантажних перевезеннях. Це може бути пов'язано з недостатньою гнучкістю, відсутністю необхідної функціональності або складністю в користуванні. Підприємствам завжди потрібен зручний та функціональний інструмент для моніторингу вантажних перевезень, який відповідає потребам сучасного ринку. Така система допоможе збільшити ефективність роботи компанії, знаходити оптимальні маршрути доставки та забезпечити зручний інтерфейс для клієнтів.

Отже, тема розробки програмного забезпечення для управління доставкою вантажних перевезень є актуальною, оскільки вона дозволяє відстежувати рух вантажу та розраховувати найкоротші маршрути.

Мета та задачі дослідження Метою бакалаврської дипломної роботи є зменшення часу, необхідного для пошуку найкоротшого маршруту доставки вантажу за рахунок розробки програмного забезпечення для управління доставкою вантажних перевезень.

Відповідно до мети роботи необхідно виконати такі задачі:

– визначити засоби реалізації програмного забезпечення для моніторингу вантажних перевезень;

– розробити калькулятор вартості доставки;

– розробити програмне забезпечення;

– удосконалити метод оптимізації на графі;

– розробити зручний графічний інтерфейс програмного забезпечення;

– провести тестування програмного забезпечення.

Об’єкт дослідження – процес управління системою вантажних перевезень.

Предмет дослідження – методи і засоби розробки програмного забезпечення для управління системою доставки вантажних перевезень.

Методи досліджень. Методи теорії графів для оптимізації маршрутів доставки; методи проектування програмного забезпечення для розробки архітектури системи, алгоритмів та програмного забезпечення; методи теорії баз даних для збереження даних про маршрути доставки та обсяги вантажів.

Новизна отриманих результатів.

1. **Удосконалено** метод пошуку найкоротшого шляху між двома вершинами графу, який відрізняється від існуючих тим, що знайдені в процесі пошуку маршрути зберігаються в пам’яті програми, що дозволило скоротити час роботи алгоритму, що реалізує даний метод.

2. **Отримав подальшого розвитку** метод зберігання та отримання даних про відстань між містами на основі формування матриці суміжностей орієнтованого зваженого графу у базі даних, який, на відміну від існуючих, представляється у програмі за допомогою ORM мапінгу, що дозволило перейти від реляційної моделі

до об'єктно-орієнтованої, чого вимагає оптимізований алгоритм пошуку найкоротшого шляху між двома вершинами графу.

Практичне значення отриманих результат Розроблено алгоритми та програмне забезпечення, яке дозволить користувачам ефективніше здійснювати управління доставкою вантажних перевезень. Крім того, зручний інтерфейс і функціонал системи дозволяють користувачам легко управляти замовленнями, стежити за станом доставок і отримувати необхідну інформацію в режимі реального часу.

Особистий внесок здобувача.

Всі наукові досягнення, які представлені у магістерській кваліфікаційній роботі, отримані автором особисто. У роботі, написаній у співаторстві, автору належить алгоритм роботи системи й оптимізація методу пошуку найкоротшого маршруту.

Апробація. Описані у роботі положення доповідалися на конференції LIII Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2024) і опубліковані в тезах доповіді цієї конференції.

Публікації. Результати роботи були опубліковані в матеріалах конференцій: LIII Науково-технічна конференція підрозділів Вінницького національного технічного університету (2024) [3].

1 АНАЛІЗ РОЗВИТКУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ УПРАВЛІННЯ СИСТЕМОЮ ДОСТАВКИ ВАНТАЖНИХ ПЕРЕВЕЗЕНЬ

1.1 Аналіз стану програмного забезпечення для управління системою доставки вантажних перевезень

На сьогоднішній день існує велика кількість програмного забезпечення для управління системою доставки вантажних перевезень. Ці програми надають широкий спектр функціональності для ефективного управління всіма аспектами доставки вантажів, включаючи моніторинг маршрутів, координацію транспортних засобів, відстеження вантажів в реальному часі та звітність.

Більшість програмного забезпечення включає в себе модулі для автоматизації процесів прийому та відправлення вантажів, оптимізації маршрутів для економії часу та грошей, а також для ведення обліку витрат та оплати послуг. Деякі програми також можуть включати інтеграцію з системами GPS для відстеження місцезнаходження транспортних засобів та управління ними в реальному часі.

Однак, не всі програмні продукти можуть відповідати всім потребам підприємств у сфері доставки вантажів. Деякі можуть бути надто складними у використанні або не мати необхідної функціональності. Також, важливою проблемою є сумісність програмного забезпечення з іншими системами, що використовуються у підприємстві.

Отже, існують багато рішень, що надають широкі можливості для ефективного управління логістичними процесами. Програмне забезпечення забезпечує автоматизацію багатьох аспектів, від прогнозування та планування маршрутів до моніторингу вантажів в реальному часі та ведення звітності.

Майбутні перспективи розвитку цього напрямку включають в себе використання штучного інтелекту для автоматизації процесів та покращення прийняття рішень, а також розширення можливостей для аналізу даних та

прогнозування витрат. Очікується, що подальше вдосконалення програмного забезпечення сприятиме підвищенню ефективності та конкурентоспроможності підприємств у галузі логістики та транспорту.

1.2 Порівняльний аналіз аналогів

Існує кілька аналогів систем для моніторингу вантажних перевезень, які використовують компанії.

Система управління транспортною логістикою підтриємств SAP Transportation Management (рисунок 1.1) – комплексне рішення, яка охоплює увесь процес логістики підтриємства та дозволяє централізовано управляти всіма перевезеннями, оптимізовувати ресурси на транспортування, а також оперативно проводити аналітику та забезпечувати ефективне планування процесів [4].

SAP Transportation Management (SAP TM) є важливим компонентом логістичного вирішення SAP, спрямованим на управління транспортними операціями та оптимізацію транспортних витрат у великих компаніях. Ця система дозволяє планувати, виконувати та контролювати транспортні процеси в реальному часі, забезпечуючи ефективність та точність виконання доставок.

Однією з ключових можливостей SAP TM є планування та виконання транспортних операцій. Система дозволяє планувати оптимальні маршрути для доставки вантажів, враховуючи різні фактори, такі як відстань, час, вага вантажу та інші. Крім того, SAP TM надає інструменти для маршрутизації та оптимізації маршрутів, що дозволяє знизити витрати на паливо та підвищити ефективність доставки.

Система також забезпечує відстеження вантажів в реальному часі, що дозволяє підприємствам ефективно контролювати та керувати доставкою. SAP TM дозволяє ефективно керувати транспортними ресурсами, включаючи водіїв, транспортні засоби та маршрути, що сприяє зниженню витрат та оптимізації використання ресурсів.

Одним з ключових переваг SAP TM є його інтеграція з іншими продуктами SAP, що дозволяє створювати єдину інформаційну систему для управління логістикою та транспортом. Крім того, система надає розширені можливості для аналізу даних та генерації звітів, що дозволяє компаніям приймати обґрунтовані рішення щодо оптимізації транспортних витрат та управління логістичними процесами.

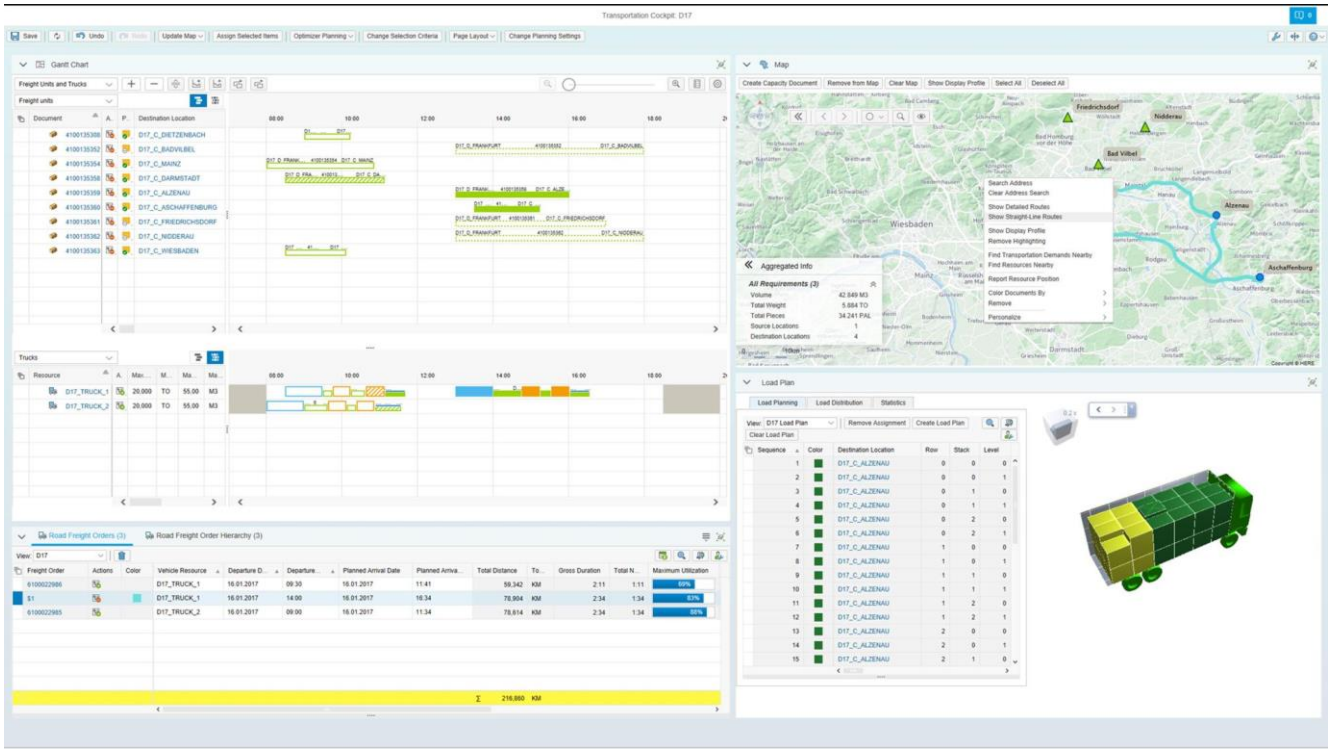

Рисунок 1.1 – Інтерфейс програмного застосунку «SAP TM»

Одними з недоліків SAP Transportation Management є високі витрати на впровадження та підтримку системи. Реалізація цієї системи може вимагати значних витрат на консультації, налаштування та навчання персоналу, що може бути складним для менших компаній з обмеженими бюджетами.

Також слід врахувати, що SAP TM є складною системою, яка вимагає високої кваліфікації для її використання та налагодження. Це може призвести до труднощів у використанні системи та потреби у спеціалізованому персоналі для її підтримки.

Окрім того, інтеграція SAP TM з іншими системами підприємства може бути складною та вимагати значних зусиль та ресурсів. Це може ускладнити впровадження системи та спричинити проблеми зі сумісністю з іншими програмами. Деякі компанії можуть виявити, що система надто потужна або складна для їхніх потреб, і можуть віддати перевагу більш простим та доступним альтернативам.

Transportation Management System, TMS (рисунок. 1.2) – логістична платформа, що використовує технології, щоб допомагати підприємствам планувати, виконувати й оптимізувати фізичне переміщення товарів, як вхідних, так і вихідних, а також забезпечувати відповідність постачання вимогам і наявність потрібної документації [5].

Однією з ключових можливостей TMS є можливість планування та виконання транспортних операцій. Система дозволяє підприємствам ефективно планувати та виконувати транспортні операції з урахуванням різних факторів, таких як вартість, час та вага вантажу. Крім того, TMS допомагає визначати оптимальні маршрути для доставки вантажів, що дозволяє знизити витрати на паливо та підвищити ефективність доставки.

Система також надає можливість відстеження вантажів в реальному часі, що дозволяє компаніям ефективно контролювати процес доставки. TMS дозволяє керувати транспортними ресурсами, такими як водії, транспортні засоби та маршрути, що сприяє зниженню витрат та оптимізації використання ресурсів.

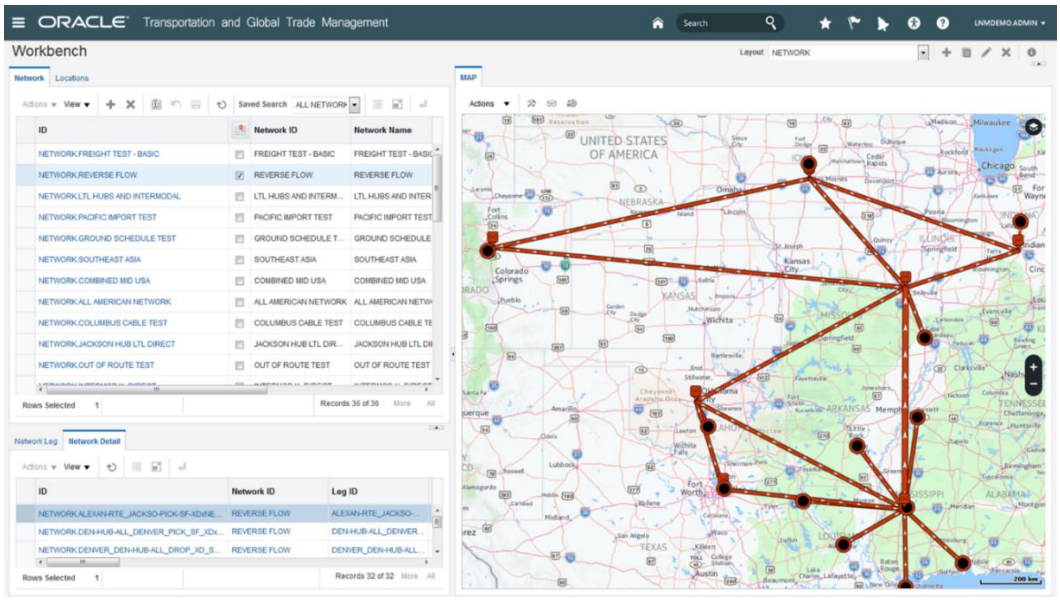

Рисунок 1.2 – Інтерфейс програмного застосунку «TMS»

Одним з основних недоліків TMS є високі витрати на впровадження та підтримку системи. Для багатьох компаній це може бути значним фактором, особливо для невеликих підприємств або тих, хто не використовує всі можливості TMS. Також важливо враховувати, що TMS є складним програмним забезпеченням, що може вимагати певного часу і зусиль для навчання персоналу і впровадження в компанії. Це може стати проблемою для компаній, які швидко потребують вирішення проблем у транспортній логістиці.

Додатково, для повноцінного функціонування TMS може бути необхідна інтеграція з іншими системами компанії, що може бути складним завданням і вимагати додаткових витрат і зусиль.

Отже, перед впровадженням TMS варто ретельно зважити всі плюси і мінуси цієї системи, а також розглянути альтернативні варіанти для управління транспортною логістикою, які можуть бути більш підходящими для конкретної компанії.

Descartes Transportation Management (рисунок. 1.3) – це програмне рішення для управління перевезеннями, призначене як для внутрішніх, так і для міжнародних ланцюгів поставок. З боку постачальника програма може

підтримувати всі контракти та відносини з постачальниками, видавати замовлення на закупівлю та відстежувати всі вхідні відправлення. По мірі надходження замовлень від клієнтів користувачі можуть налаштовувати маршрути вихідних посилок і отримувати пропозиції щодо послуг перевізників [6].

Descartes TMS надає широкий спектр функцій, включаючи планування маршрутів, управління замовленнями, відстеження вантажів, оптимізацію вантажопотоків, аналіз даних та звітність. Система дозволяє підприємствам знизити витрати на транспорт, підвищити ефективність управління та поліпшити обслуговування клієнтів.

Один з ключових плюсів Descartes TMS – це його гнучкість і можливість адаптації до потреб конкретного бізнесу.

Система може інтегруватися з іншими корпоративними системами, що дозволяє автоматизувати обмін даними і забезпечити єдиною точкою доступу до інформації про транспортні операції.

Descartes Transportation Management System є потужним інструментом для оптимізації управління транспортними процесами в компаніях. Однак, серед його недоліків можна виділити кілька ключових аспектів.

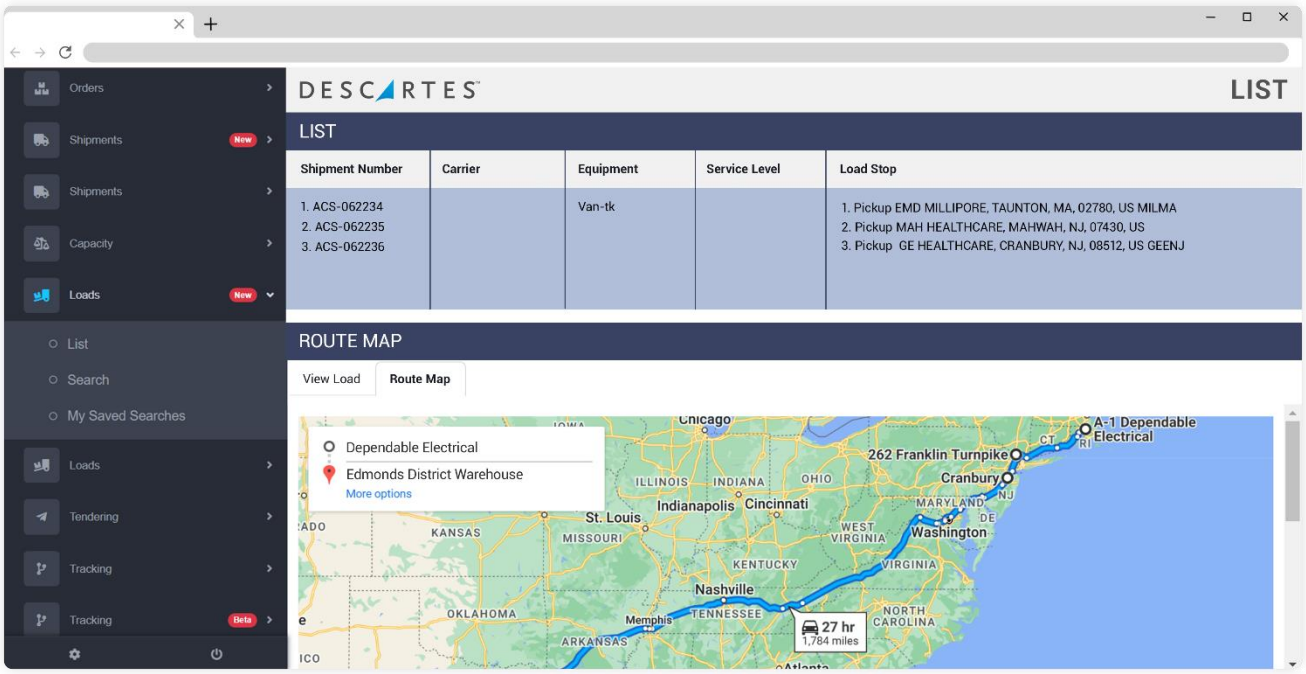

Рисунок 1.3 – Інтерфейс програмного застосунку «Descartes»

Перш за все, вартість реалізації Descartes TMS може бути високою, що робить його менш доступним для менших підприємств або тих, у кого обмежений бюджет.

Крім того, складність впровадження є ще одним фактором, що ускладнює використання системи. Інтеграція Descartes TMS з існуючими системами підприємства може вимагати значних зусиль і часу.

Також важливо враховувати необхідність навчання персоналу. Використання Descartes TMS може вимагати підготовки співробітників через складний інтерфейс та функціонал системи.

Нарешті, підтримка та обслуговування Descartes TMS можуть вимагати додаткових витрат і ресурсів у вигляді підписки на підтримку або спеціалістів з підтримки програмного забезпечення. Результати порівняння аналогів наведено в табл. 1.1.

Таблиця 1.1 – Функціональні характеристики програмних додатків

	Власна реалізація	SAP TM	TMS	Descartes
Веб сайт	1	0	1	0
Зручний інтерфейс	1	1	0	0
Калькулятор вартості	1	1	0	1
Оптимізація	1	0	1	0
Загальна оцінка	100%	50%	50%	25%

Отже, відповідно до порівняльної таблиці, розробка програмного забезпечення для управління системою доставки вантажних перевезень є доцільною.

1.3 Аналіз методів розв’язання задачі

Для розробки програмних засобів системи управління та забезпечення доставки вантажних перевезень необхідно використовувати різні алгоритми, які допоможуть оптимізувати процес доставки та забезпечити ефективне управління вантажами. У цьому розділі проаналізовано найбільш ефективні алгоритми для пошуку маршрутів доставки.

Алгоритм Дейкстри є одним із найважливіших і найбільш використовуваних алгоритмів у галузі комп’ютерних наук. Він нівелює складність задачі знайдення найкоротшого шляху у сполученому графі та дає можливість розв’язувати різноманітні задачі в таких галузях, як транспорт, маршрутизація та мовленнєві мережі. Основна мета алгоритму Дейкстри – знайти найкоротший шлях від одного зазначеного вузла до всіх інших вузлів у графі. Для досягнення цієї мети алгоритм Дейкстри використовує принцип жадібної стратегії, оптимально обираючи кожну наступну вершину у графі на основі її ваги та дистанції від початкової вершини [7].

У контексті управління доставкою вантажів, цей алгоритм може бути використаний для побудови оптимального маршруту доставки вантажу від точки відправлення до точки призначення. Його ефективність полягає у здатності знаходити найкоротший шлях від однієї точки до всіх інших у графі, що особливо важливо для доставки вантажів у багатопунктовій мережі.

Ще одним можливим варіантом для пошуку оптимального маршруту доставки є алгоритм A*. Це інформаційний алгоритм пошуку найкращого шляху, який ефективно визначає найкоротший шлях між будь-якими двома вершинами в орієнтованому зваженому графі з невід'ємними вагами ребер. Цей алгоритм є варіантом алгоритму Дейкстри. Невелика відмінність полягає в тому, що для визначення того, яку вершину досліджувати далі, використовується оціночна функція [8].

Цей алгоритм використовує евристичну оцінку відстані до кінцевої точки, що дозволяє йому швидше знаходити шлях між двома точками у великих графах з вагами. Його ефективність полягає у здатності знаходити оптимальний шлях, враховуючи якість прогнозування.

Також варто розглянути алгоритм Флойда-Уоршелла. Алгоритм Флойда-Уоршалла – це популярний алгоритм пошуку найкоротшого шляху для кожної пари вершин у зваженому орієнтованому графі [9]. Цей алгоритм дозволяє знайти найкоротші шляхи між усіма парами вершин у графі. В контексті управління доставкою вантажів, він може бути корисним для планування маршрутів доставки вантажів у складних мережах, де потрібно враховувати багато можливих шляхів і ваги на ребрах.

Порівнявши ці алгоритми, можна зробити висновок, що для задачі управління доставкою вантажів, де потрібно знаходити найкоротший шлях в дорожній мережі, найбільш підходящим буде алгоритм Дейкстри. Він дозволить ефективно знаходити оптимальний маршрут великій кількості точок доставки, що є важливим для оптимізації маршрутів доставки вантажів.

1.4 Постановка задач досліження

Враховуючи аналіз функціоналу застосунків для управління доставкою вантажів було визначено завдання, які необхідно виконати для розробки системи, а саме:

1) створення структури бази даних для зберігання інформації про клієнтів, вантажі, транспортні засоби та інші важливі дані;

2) розробка інтерфейсу для прийому та обробки замовлень на доставку, включаючи вибір маршруту, розрахунок вартості, назначення водіїв тощо;

3) створення засобів для аналізу ефективності доставки, включаючи звіти про витрати, час доставки тощо;

4) створення зручного та інтуїтивно зрозумілого інтерфейсу системи для замовлення доставки вантажів, перегляду статусу замовлень та звітності.

5) імплементація алгоритму Дейкстри для знаходження найкоротшого маршруту доставки вантажів, що дозволить зменшити час доставки та витрати на паливо.

1.5 Висновки

У першому розділі було проведено аналіз поточного стану застосунків для управління доставкою вантажів, таких як «SAP TM», «TMS» та «Descartes». Порівняння цих систем дозволило оцінити їх переваги та недоліки у порівнянні із запропонованим рішенням, що вказало на доцільність створення власного програмного рішення. На основі цього аналізу був сформований перелік завдань, які необхідно виконати для успішної розробки власної системи управління доставкою вантажів.

2 РОЗРОБКА СТРУКТУРИ ТА АЛГОРИТМІВ ПРОГРАМНОГО ПРОДУКТУ

2.1 Оптимізація методу пошуку найкоротшого маршруту

Граф – це абстрактна математична структура, яка використовується для моделювання відносин між об'єктами. Він складається з двох основних компонентів: вершин і ребер. Вершини, також відомі як вузли, представляють об'єкти, а ребра, також відомі як дуги або лінії, представляють зв'язки між цими об'єктами. Графи можна класифікувати за кількома ознаками, такими як орієнтованість і наявність ваги на ребрах.

Неорієнтовані графи характеризуються тим, що їхні ребра не мають напрямку. Якщо існує ребро між вершинами то ці вершини суміжні, і напрямок зв'язку не має значення. Такий підхід часто використовується для моделювання симетричних відносин, наприклад, дружніх зв'язків у соціальних мережах або доріг у транспортних мережах, де можна рухатися в обох напрямках. Орієнтовані графи, навпаки, мають ребра з напрямком. В таких графах кожне ребро визначається парою вершин, де перша вершина є початковою, а друга – кінцевою. Наприклад, у графі, що моделює веб-сторінки та гіперпосилання між ними, орієнтовані ребра вказують напрямок переходу від однієї сторінки до іншої.

Зважені графи мають додаткову характеристику – вагу ребер. Вага ребра може представляти різноманітні величини, такі як відстань, вартість або час. Зважені графи особливо корисні для задач, пов'язаних із оптимізацією, наприклад, знаходження найкоротшого шляху в транспортних мережах або мінімальних витрат у мережах постачання. У таких графах вага ребра між вершинами

Існує кілька способів представлення графів у комп'ютерній пам'яті. Один з них – матриця суміжності. Це двовимірний масив, де кожен елемент масиву відповідає ребру між двома вершинами. Якщо вершини суміжні, елемент масиву містить вагу ребра (або 1 для незважених графів), в іншому випадку – спеціальне

значення, яке вказує на відсутність ребра (наприклад, нуль або нескінченність). Матриця суміжності зручна для графів з невеликою кількістю вершин, де потрібно швидко перевіряти наявність ребра між будь-якими двома вершинами.

Інший спосіб представлення графів – список суміжності. У цьому підході кожна вершина має список всіх вершин, з якими вона суміжна. Це дозволяє ефективно використовувати пам'ять для розріджених графів, де більшість вершин не пов'язані безпосередньо між собою. Списки суміжності часто реалізуються як масиви або списки, де кожен елемент містить список суміжних вершин і відповідні ваги ребер.

Графи мають численні застосування в різних галузях. У комп'ютерних науках графи використовуються для моделювання комп'ютерних мереж, де вершини представляють комп'ютери або інші пристрої, а ребра – канали зв'язку між ними. У біоінформатиці графи використовуються для моделювання взаємодій між білками або генами. В економіці та логістиці графи допомагають оптимізувати маршрути доставки та мінімізувати витрати на транспортування товарів.

Використання графів для управління доставкою вантажів є потужним підходом для оптимізації маршрутів, зниження витрат і підвищення ефективності логістичних процесів. Графи дозволяють моделювати транспортну мережу, де вершини представляють пункти відправлення, призначення або проміжні пункти, а ребра – можливі шляхи між цими пунктами.

Для програмного забезпечення для управління системою доставки вантажних перевезень обрано матрицю суміжності як спосіб представлення графа через наведені переваги:

Простота реалізації та доступу до даних. Матриця суміжності дозволяє спростити доступ до інформації про ребра між вершинами. У матриці розміром $V \times V$, де V – кількість вершин, кожна комірка (i, j) містить вагу ребра між вершинами i та j. Такий підхід забезпечує доступ до ваги ребра за константний час $O(1)$, що є значною перевагою в порівнянні зі списком суміжності, де доступ до ребра може вимагати часу $O(V)$ у найгіршому випадку. Це особливо важливо для

алгоритмів пошуку найкоротших шляхів, таких як алгоритм Дейкстри, який використовується в нашому програмному забезпеченні для оптимізації маршрутів доставки.

Зручність для щільних графів. Транспортна мережа моделюється як граф, де вершини представляють різні логістичні пункти (склади, термінали, пункти доставки), а ребра – можливі маршрути між ними. У багатьох випадках така мережа є щільною, тобто більшість можливих пар вершин з'єднані ребрами. У щільних графах матриця суміжності є ефективною з точки зору пам'яті та швидкодії, оскільки кількість ребер E близька до V^2, де V – кількість вершин. У такому випадку матриця суміжності забезпечує кращу продуктивність, ніж список суміжності.

Спрощення реалізації алгоритмів. Багато алгоритмів для роботи з графами, такі як алгоритми пошуку найкоротших шляхів, максимального потоку або мінімального покриття, є більш простими для реалізації з використанням матриці суміжності. В нашій системі управління доставкою вантажів реалізація алгоритму Дейкстри для знаходження найкоротших шляхів була значно спрощена завдяки використанню матриці суміжності. Це дозволило скоротити час розробки та знизити ймовірність помилок у коді.

Візуалізація та відлагодження. Матриця суміжності є інтуїтивно зрозумілим представленням графа, яке легко візуалізувати і аналізувати. Це спрощує процес відлагодження та тестування програмного забезпечення, оскільки структура графа представлена у вигляді двовимірного масиву, де кожна комірка чітко відображає наявність або відсутність зв'язку між двома вершинами. Це дозволило нам швидше і ефективніше виявляти та виправляти помилки у програмному забезпеченні.

Можливість розширення та модифікації. У майбутньому програмне забезпечення може вимагати додавання нових функцій або змін у транспортній мережі. Матриця суміжності надає гнучкість у розширенні та модифікації графа, оскільки додавання нових вершин або ребер можна реалізувати шляхом додавання

нових рядків або стовпців у матриці. Це дозволяє легко адаптувати систему до змін у логістичній мережі без значних переписувань коду.

Використання матриці суміжності для представлення графа в програмному забезпеченні по доставці вантажів було обґрунтоване перевагами у простоті реалізації, швидкодії доступу до даних, зручності для щільних графів, спрощенні реалізації алгоритмів, а також у можливостях візуалізації, відлагодження та майбутнього розширення системи. Цей вибір забезпечив оптимальну продуктивність та гнучкість програмного забезпечення, що є ключовими факторами для ефективного управління логістичними процесами.

Порівняємо стандартний та оптимізований алгоритм Дейкстри.

Стандартний алгоритм Дейкстри для пошуку найкоротшого шляху в графі:

1. Встановлюємо відстані до всіх вершин як нескінченні, крім початкової вершини, до якої відстань 0.

2. Поки є неперевірені вершини:

- Вибираємо неперевірену вершину з найменшою відстанню.
- Позначимо її як перевірену.
- Оновлюємо відстані до всіх її сусідів, якщо знайдена менша відстань.

3. Отримуємо найкоротші відстані від початкової вершини до всіх інших вершин.

Оптимізований алгоритм Дейкстри.

1. Ініціалізація. Всі вершини отримують нескінченні відстані, окрім джерела (відстань 0). Всі вершини позначаються як непройдені.

2. Вибір вершини. Обирається непройдена вершина з найменшою відстанню.

3. Оновлення відстаней. Оновлюються відстані до сусідніх вершин, якщо знайдено коротший шлях через обрану вершину.

4. Позначення вершини. Вершина позначається як пройдена.

5. Повторення. Кроки 2-4 повторюються, доки не буде пройдено всі вершини.

Порівнюючи ці реалізації алгоритмів, стандартний алгоритм Дейкстри ефективний для графів з невеликою кількістю вершин і ребер, але повільний для великих графів.

Оптимізований алгоритм Дейкстри включає знайдені в процесі пошуку маршрути які зберігаються в пам’яті програми, що значно пришвидшує швидкість пошуку для великих графів.

Отже, для ефективного вирішення задач управління доставкою вантажів необхідно використовувати оптимізовану версію алгоритму Дейкстри. Це дозволить зменшити час виконання, витрати пам'яті та забезпечити адекватну роботу алгоритму.

2.2 Розробка прототипу інтерфейсу програмного забезпечення

Прототип сторінки входу користувача – це важлива частина розробки програмного забезпечення, яка дозволяє визначити структуру та функціонал входу в систему.

У такому прототипі можуть бути відображені основні елементи інтерфейсу, такі як поля для введення логіна та пароля, кнопки для надсилання даних, посилання на відновлення пароля або реєстрацію.

Прототип сторінки входу може бути представлений у вигляді макету інтерфейсу з мінімальним функціоналом, який дозволяє користувачеві уявити собі процес входу в систему. Такий прототип допомагає розробникам зрозуміти потреби користувачів та внести необхідні зміни до інтерфейсу до фактичної реалізації програми.

У прототипі сторінки входу важливо врахувати принципи дизайну, забезпечити зручність використання для користувачів і підвищити загальний рівень зручності та безпеки системи (рисунок 2.1).

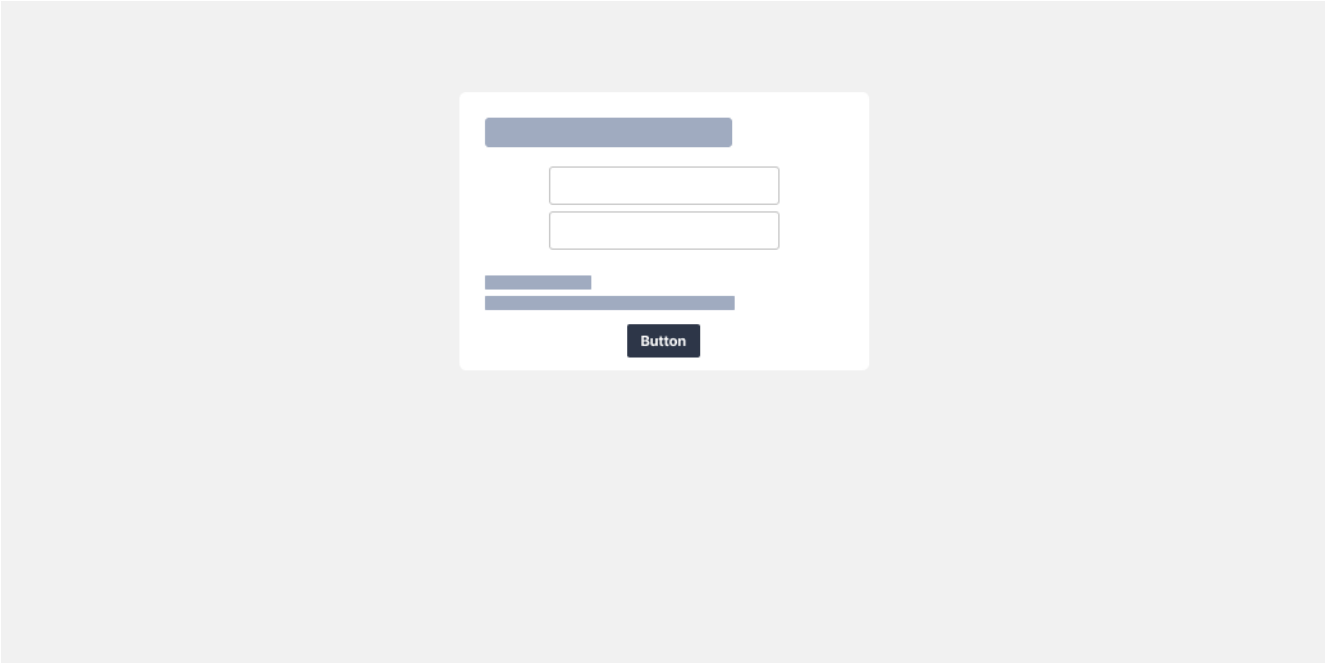

Рисунок 2.1 – Прототип сторінки входу користувача

Прототип сторінки напрямків перевезень – це концептуальне відображення інтерфейсу, що відображає можливі напрямки та маршрути доставки вантажів.

Цей прототип дозволяє користувачам отримати загальне уявлення про доступні маршрути, дистанції та вартість перевезення в різні місця.

У прототипі сторінки напрямків перевезень можуть бути відображені наступні елементи:

1. Список доступних напрямків та маршрутів з відповідними відстанями доставки.
2. Фільтри для вибору конкретного напрямку.
3. Кнопки для переходу до інших сторінок.

Прототип сторінки напрямків перевезень відіграє ключову роль у визначенні оптимального способу відображення інформації про доступні маршрути та послуги. Цей прототип допомагає зрозуміти, як краще організувати відображення інформації про напрямки перевезень, включаючи відстань, час доставки, вартість та інші параметри. Правильно спроектований прототип сторінки напрямків

перевезень дозволяє покращити користувацький досвід, забезпечуючи зручний доступ до необхідної інформації (рисунок 2.2).

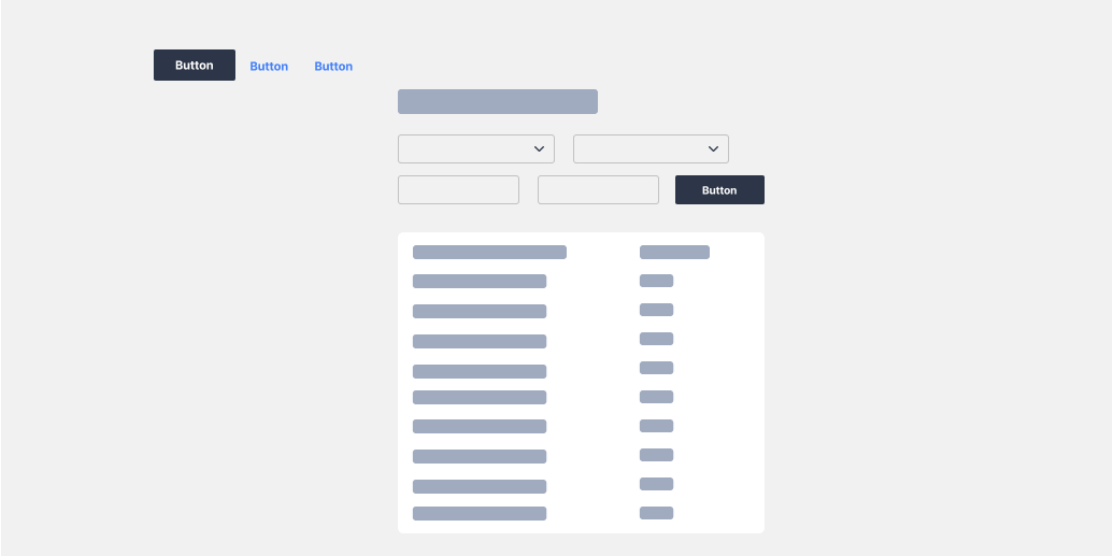

Рисунок 2.2 – Прототип сторінки напрямків перевезень

Прототип сторінки тарифів – це візуальне відображення інформації про вартість послуг доставки вантажів, що надаються компанією. Цей прототип має на меті показати користувачам доступні тарифи, їхні умови та вартість, допомагаючи зробити правильний вибір (рисунок 2.3).

Прототип сторінки тарифів допомагає користувачам швидко знайти інформацію про вартість послуг та зробити відповідний вибір залежно від їхніх потреб і можливостей.

Рисунок 2.3 – Прототип сторінки тарифів

Прототип калькулятора вартості перевезення – це інтерактивний інструмент, який дозволяє користувачам швидко розрахувати приблизну вартість доставки вантажу за певними параметрами (рисунок 2.4).

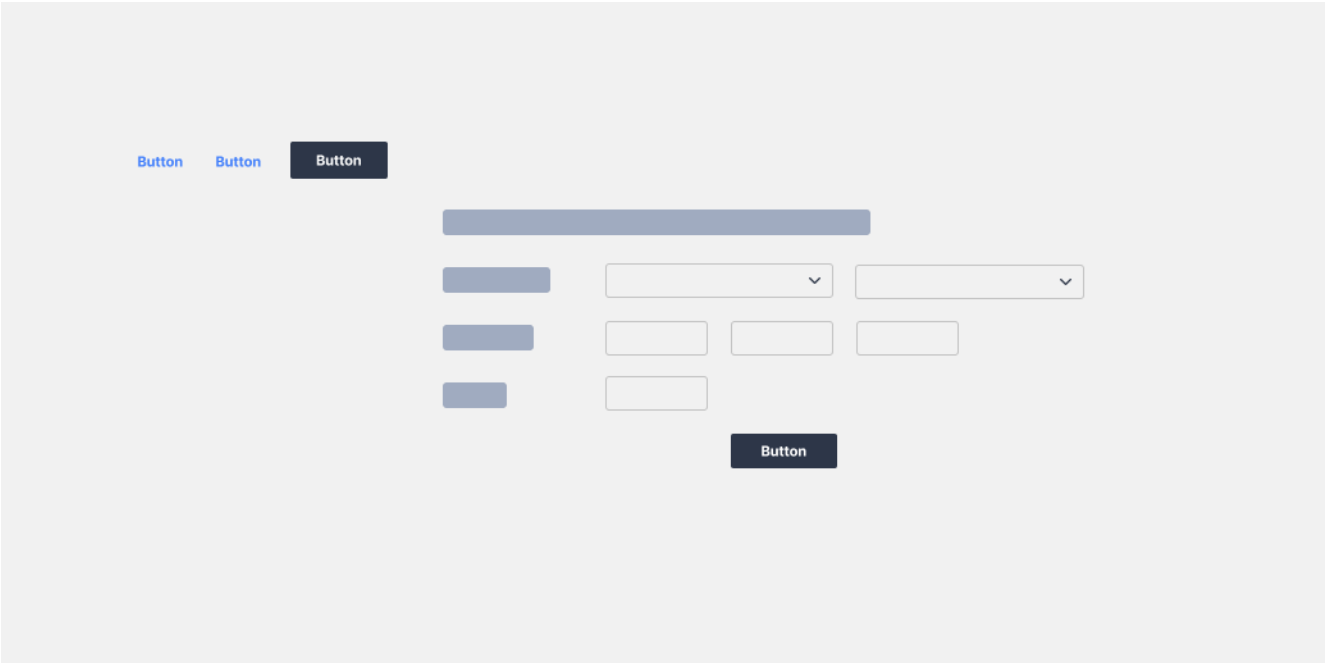

Рисунок 2.4 – Прототип калькулятора вартості перевезеннь

У прототипі калькулятора вартості перевезення можуть бути такі елементи:

1. Форма для введення параметрів вантажу, таких як вага, розміри, відстань перевезення тощо.

2. Кнопка "Розрахувати", яка запускає алгоритм розрахунку вартості перевезення на основі введених даних.

3. Поле для відображення результату розрахунку – орієнтовної вартості доставки.

Правильно спроектований прототип калькулятора допомагає зробити процес розрахунку вартості прозорим та зрозумілим для користувачів, що сприяє покращенню їхнього досвіду використання програмного забезпечення. Крім того, він дозволяє підтримувати конкурентоспроможні ціни та привертати нових клієнтів. В результаті, прототип калькулятора вартості перевезення є важливим інструментом для забезпечення успішності програмного забезпечення по доставці вантажів.

Прототип сторінки "Створити заявку" для вантажної доставки включає в себе форму, де користувач може ввести дані щодо відправлення та отримання вантажу (рисунок 2.5).

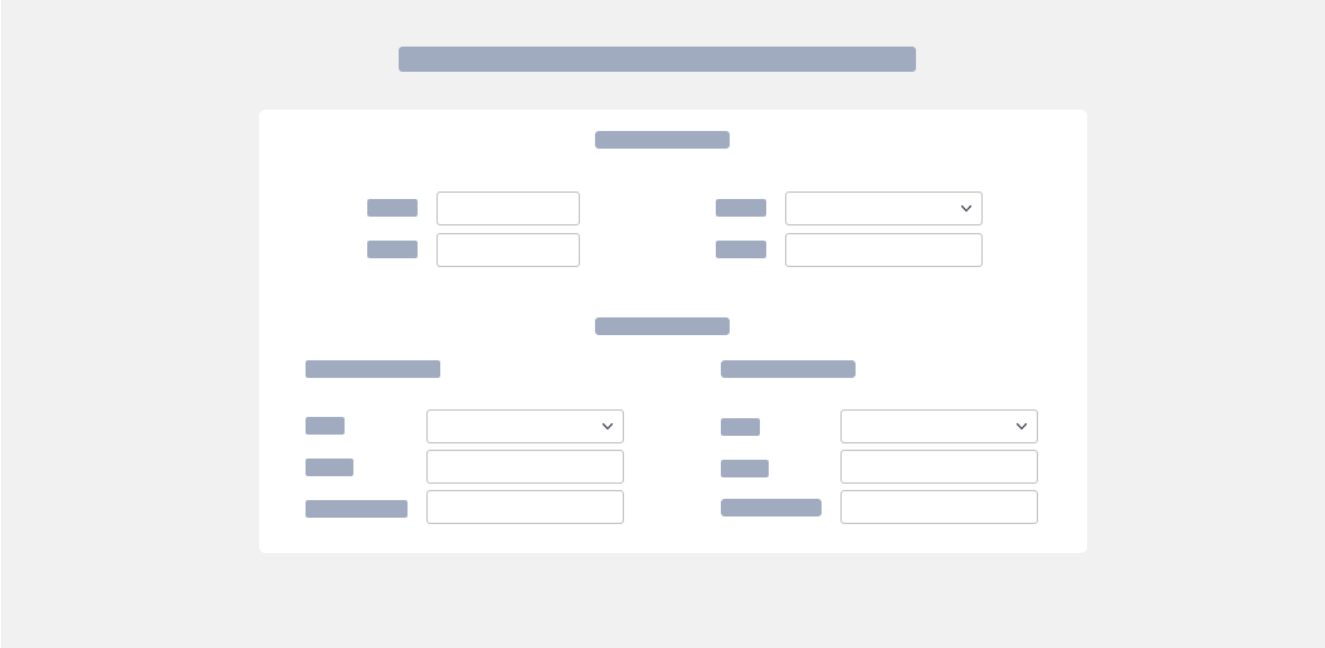

Рисунок 2.5 – Прототип сторінки «Створити заявку»

Користувачу буде запропоновано ввести інформацію, таку як адреса відправлення та отримання, тип вантажу, його розміри та вагу. Деякі поля можуть бути обов'язковими для заповнення, щоб забезпечити коректність та швидкість обробки заявки.

Прототип сторінки "Створити заявку" для вантажної доставки може включати наступні елементи:

1. Форма для введення даних відправника та отримувача, включаючи імена, контактну інформацію та адреси.
2. Можливість вибору типу вантажу та його параметрів (вага, розміри).
3. Календар для вибору дати відправлення та прибуття.
4. Кнопка "Підтвердити замовлення", щоб відправити заявку на обробку.

Цей прототип допомагає користувачам швидко та зручно створити заявку на вантажну доставку, надаючи необхідну інформацію та вибір параметрів для відправлення.

Прототип профілю користувача для системи управління доставкою вантажів може включати основну інформацію про користувача, його замовлення та історію взаємодії з системою (рисунок 2.6).

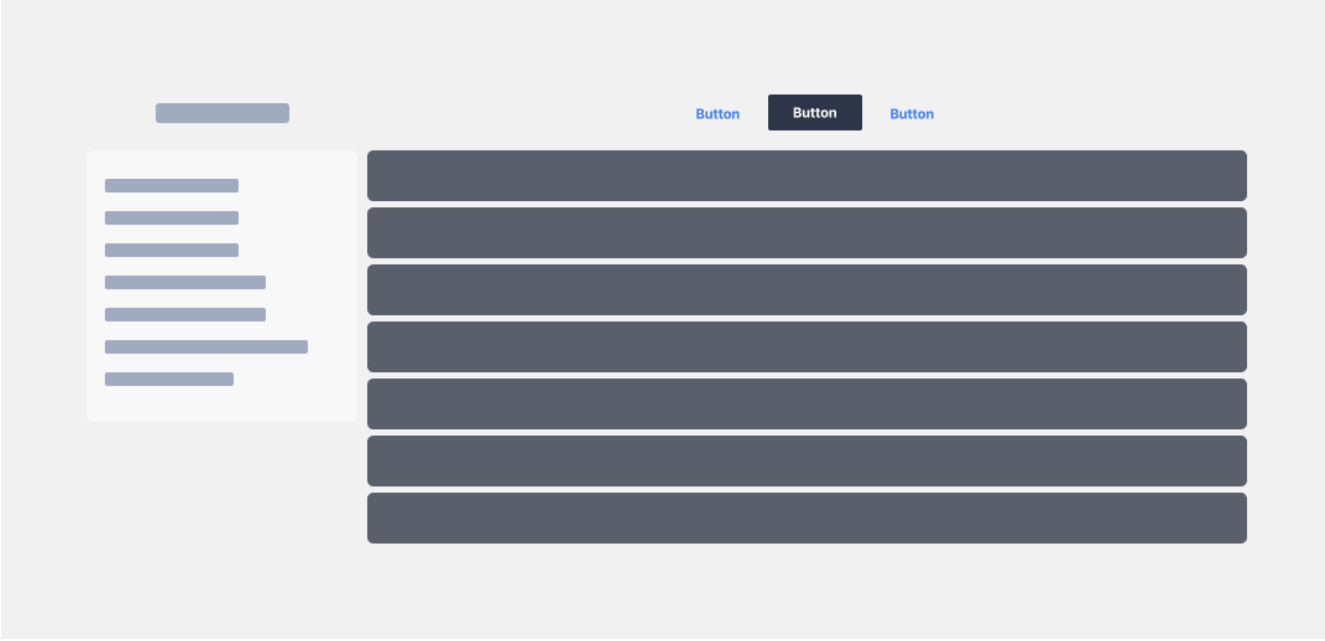

Рисунок 2.6 – Прототип профілю користувача

У розділі замовлень користувач може переглядати історію своїх замовлень, включаючи активні та завершені замовлення. Для кожного замовлення можуть відображатися деталі, такі як дата та час замовлення, маршрут доставки, статус замовлення тощо. Користувач може мати можливість відслідковувати статус свого замовлення та отримувати сповіщення про зміни.

Загальний дизайн сторінки профілю користувача може бути зрозумілим та інтуїтивно зрозумілим, з використанням чіткої навігації та зручних інтерфейсів для редагування даних та перегляду історії. Додаткові функції, такі як можливість експорту даних або збереження улюблених маршрутів, можуть бути додані для покращення користувацького досвіду.

На сторінці аналітики, доступній для адміністратора сайту з доставки вантажів, можуть бути представлені різноманітні звіти та графіки, що дозволяють відстежувати та аналізувати ключові показники (рисунок 2.7). Наприклад, адміністратор може переглядати загальну кількість відправлень за певний період, розподіл типів вантажів та графік доходів.

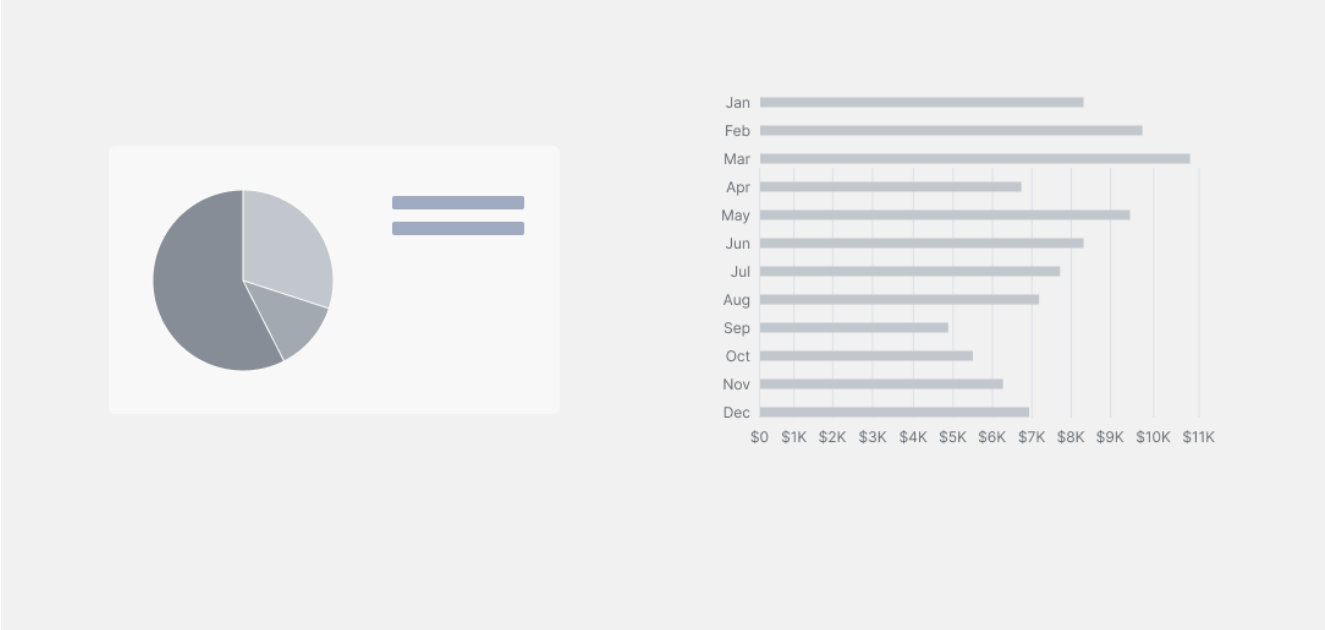

Рисунок 2.7 – Прототип сторінки аналітики

Прототип сторінки аналітики для системи управління доставкою вантажів може включати різноманітні звіти та графіки, які дозволяють адміністраторам системи аналізувати різні аспекти діяльності компанії з доставки вантажів. На цій сторінці можуть бути доступні наступні функції та компоненти:

1. Графіки та діаграми, що відображають загальну активність компанії з доставки вантажів, такі як кількість замовлень за період, загальний обсяг виконаних доставок, динаміка змін у попиті на послуги доставки тощо.

2. Відображення фінансової продуктивності компанії, такої як доходи від доставок, витрати на обслуговування, прибуток тощо. Можливість фільтрації за різними параметрами (наприклад, за місяцем або за типом послуги).

Отже, розробка графічного інтерфейсу є критично важливою для успіху будь-якого програмного продукту, оскільки від нього залежить сприйняття користувачем системи. Якісно розроблений інтерфейс дозволяє забезпечити зручність та ефективність взаємодії, покращити користувацький досвід і задоволення від використання продукту. Правильно спроектований інтерфейс сприяє залученню користувачів, збільшенню їхньої віддачі та позитивному враженню від продукту, що може значно підвищити конкурентоспроможність системи на ринку. Розробка графічного інтерфейсу була розроблена в сервісі Figma та реалізована в середовищі IntelliJ IDEA із використанням технологій Java та Spring.

2.3 Розробка архітектури серверної частини

Програмування на стороні сервера є дуже корисним, оскільки дозволяє ефективно доставляти інформацію, адаптовану для окремих користувачів, і тим самим створювати набагато кращий користувацький досвід [10].

Архітектуру серверної частини системи продемонстровано на рисунку 2.8.

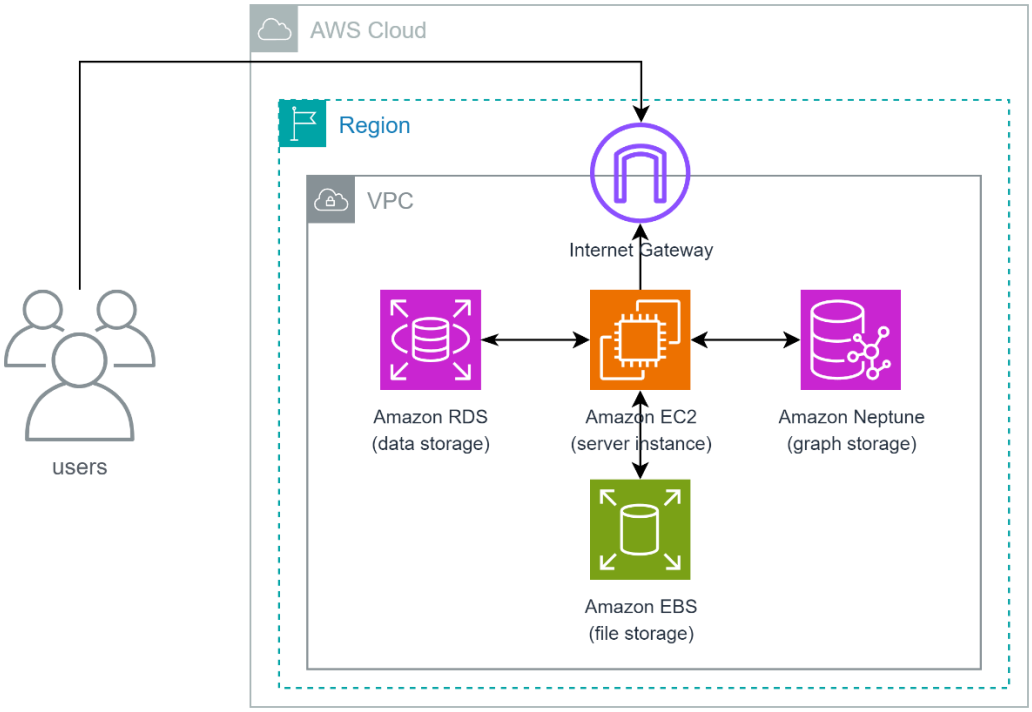

Рисунок 2.8 – Архітектура серверної частини системи

Складові системи:

Amazon Web Services – це онлайн-платформа, яка надає користувачам віртуальні обчислювальні ресурси, сховища, інфраструктуру та сервіси, що містять готовий до використання функціонал [11].

Region в термінах хмарних послуг, таких як AWS, означає географічну область, де фізично розташовані центри обробки даних (data centers). Кожен регіон може мати кілька доступних "Availability Zones" (зон доступності), які представляють собою відокремлені фізичні області з інфраструктурою обчислення та зберігання даних.

Amazon Virtual Private Cloud (Amazon VPC) – це логічно ізольований розділ хмари AWS, в якому можна запускати ресурси AWS в самостійно заданій віртуальній мережі [12], яка дозволяє ізолювати ресурси користувача у віртуальному мережевому середовищі у великому публічному хмарному інфраструктурному рішенні, такому як AWS. VPC дозволяє користувачам керувати власною мережею, визначати IP-адреси, створювати підмережі, налаштовувати

правила мережевого доступу та підключати до мережі інші хмарні ресурси, такі як сервери, бази даних та інші сервіси. Використання VPC дозволяє створювати безпечне та ізольоване мережеве середовище для роботи з хмарними послугами.

Internet Gateway (Інтернет-шлюз) – це горизонтально масштабований, надлишковий і високодоступний компонент VPC, який забезпечує зв'язок між вашим VPC та Інтернетом [13]. Він не спричиняє ризиків для доступності або обмежень пропускної здатності вашого мережевого трафіку.. Internet Gateway дозволяє ресурсам у VPC звертатися до Інтернету для виконання завдань, таких як відправка запитів на зовнішні ресурси або отримання оновлень програмного забезпечення. В той же час, Internet Gateway допомагає захищати приватні ресурси від несанкціонованого доступу з Інтернету, використовуючи правила контролю доступу (security groups або network access control lists).

Amazon Elastic Compute Cloud (EC2) – це веб сервіс, який надає розширювані обчислювальні потужності в хмарі. EC2 дозволяє користувачам орендувати віртуальні сервери (інстанси) для запуску своїх застосунків. Amazon EC2 є однією з основних служб Amazon Web Services (AWS) і забезпечує гнучке надання комп'ютерних ресурсів, тобто обчислювальних потужностей [14]. Користувачі можуть вибирати тип інстансу, операційну систему, мережові налаштування та зберігання даних, щоб відповідати вимогам їх застосунків. EC2 дозволяє масштабувати обчислювальні потужності вгору або вниз в залежності від потреб застосунку, що дозволяє ефективно використовувати ресурси і зменшувати витрати.

Amazon Neptune – це безсерверна база даних графової аналітики, яка спрощує створення інтерактивних графічних застосунків і управління ними в будь-якому масштабі [15]. Цей сервіс дозволяє легко створювати та управляти високопродуктивними графовими базами даних, що підтримують модель графових даних. Amazon Neptune надає гнучкість та масштабованість управління даними, дозволяючи виконувати складні запити та аналіз графових даних.

Amazon RDS (Amazon Relational Database Service) – це сервіс керованої реляційної бази даних від Amazon Web Services (AWS). Amazon RDS дозволяє легко налаштовувати, експлуатувати та масштабувати реляційні бази даних в хмарному середовищі без необхідності керування інфраструктурою баз даних. Він підтримує різні типи баз даних, такі як MySQL, PostgreSQL, Oracle, SQL Server та інші, і надає можливості автоматичного резервного копіювання, масштабування, моніторингу та відновлення баз даних. Отже, Amazon RDS – це простий в управлінні сервіс реляційних баз даних, оптимізований з урахуванням сукупної вартості володіння [16].

Amazon Elastic Block Store (Amazon EBS) – це простий у використанні, масштабований, високопродуктивний сервіс зберігання блоків, розроблений для Amazon Elastic Compute Cloud (Amazon EC2) [17]. Amazon EBS дозволяє прикріплювати блокові сховища до віртуальних машин, що дозволяє зберігати дані при зупинці або знищенні інстансів EC2. EBS забезпечує надійність і можливість масштабування, а також надає можливості резервного копіювання та синхронізації даних.

2.4 Розробка алгоритмів роботи системи

Для розробки засобів системи управління та забезпечення доставки вантажних перевезень потрібно розробити алгоритм авторизації та реєстрації.

Реєстрація та авторизація є важливими компонентами будь-якої системи, що вимагає ідентифікації користувачів і контролю доступу до ресурсів. Реєстрація зазвичай включає в себе створення облікового запису користувача в системі, в той час як авторизація визначає, чи має користувач доступ до конкретних функцій або ресурсів.

Процес реєстрації включає заповнення форми з обов'язковими полями, такими як ім'я, електронна пошта, пароль тощо. Після надання цих даних вони

перевіряється на валідність (наприклад, унікальність електронної пошти) і збережені в базі даних.

Після успішної реєстрації користувач може увійти до системи, використовуючи свої облікові дані.

Авторизація вимагає перевірки введеного користувачем пароля та ідентифікатора (наприклад, електронної пошти або імені користувача) на відповідність збереженим даним в базі даних.

Також забезпечено зручність для користувача під час реєстрації та авторизації. Це включає зручний інтерфейс для заповнення форм, можливість відновлення пароля в разі його забуття та інші зручні функції.

Реєстрація та авторизація відіграють ключову роль у забезпеченні безпеки та функціональності системи.

Реєстрація дозволяє користувачам створювати облікові записи, які дозволяють їм взаємодіяти з системою та зберігати свої дані. Авторизація, у свою чергу, підтверджує ідентифікацію користувача та надає доступ до особистого кабінету або функціоналу сайту.

Важливо, щоб процеси реєстрації та авторизації були максимально зручними для користувачів.

Це допомагає забезпечити високий рівень задоволення користувачів від використання сайту та підвищує його популярність серед аудиторії. Такий підхід дозволяє створити довіру між користувачами і системою, що є важливим аспектом у сучасному інтернет-бізнесі.

Згідно даного алгоритму, зображеного на рисунку 2.8, користувач має увійти в систему через логін та пароль для подальшого користування.

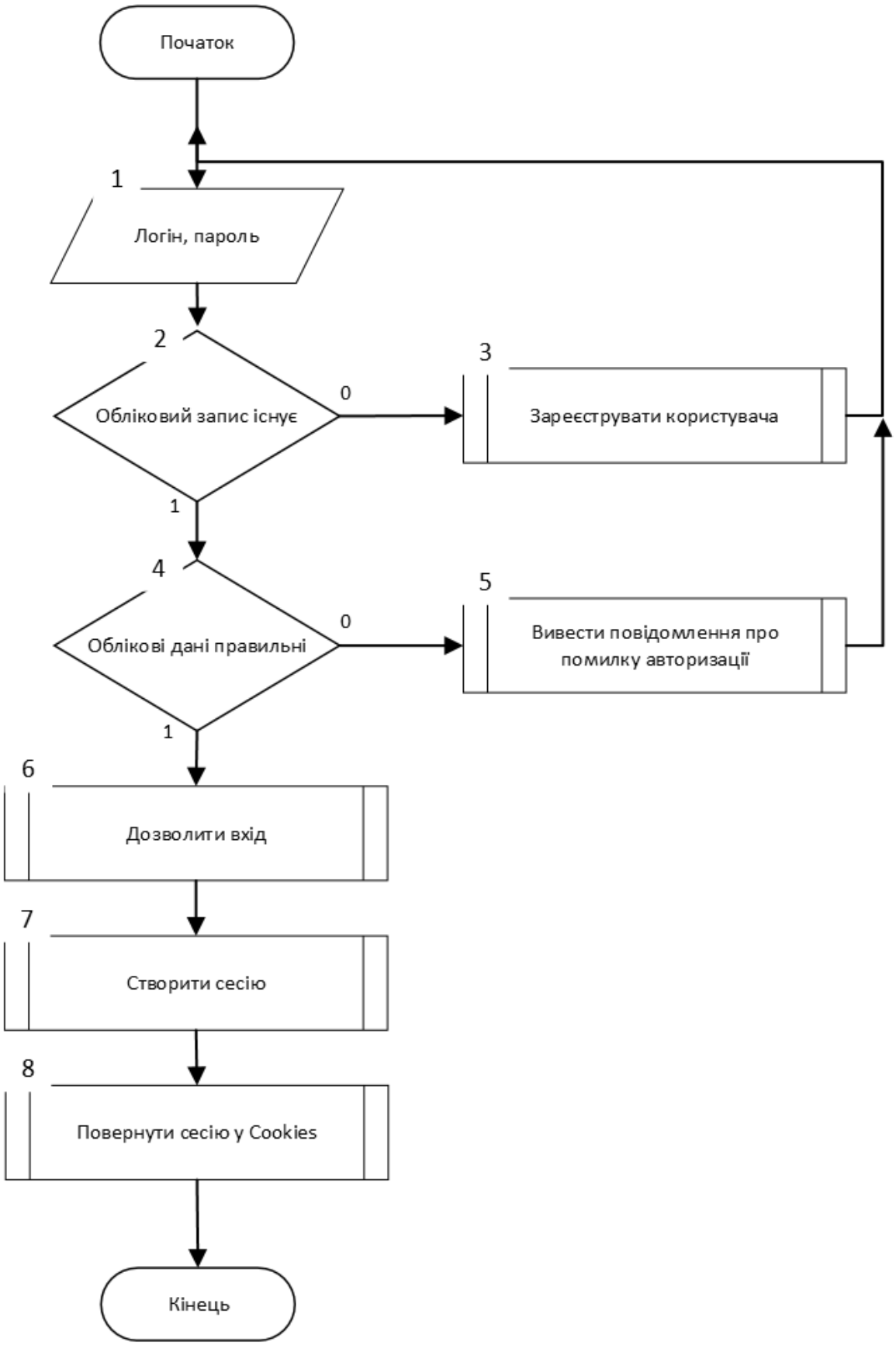

Рисунок 2.8 – Схема алгоритму роботи авторизації та реєстрації

Алгоритм Дейкстри – це один з найефективніших алгоритмів для знаходження найкоротшого шляху в напрямленому або ненапрямленому зваженому графі з однією відомою початковою вершиною.

Він може бути використаний для вирішення широкого спектру задач, таких як планування маршрутів в комп'ютерних мережах, транспортній логістиці та багатьох інших.

Основна ідея алгоритму полягає в тому, щоб у кожен момент часу мати найкоротший шлях до кожної вершини, який вже знайшли. Для цього використовуємо дві структури даних: чергу пріоритетів (зазвичай реалізовану через бінарну купу) та список відвіданих вершин.

Алгоритм починається з визначення відстаней до всіх вершин від початкової вершини як нескінченність, за винятком самої початкової вершини, відстань до якої встановлюється на 0.

Потім вершина з найменшою відстанню додається до списку відвіданих. Для кожної сусідньої вершини, яка ще не відвідана, обчислюється нова відстань і порівнюється з поточною відстанню до цієї вершини. Якщо нова відстань менше, вона оновлюється, і вершина додається до черги пріоритетів для подальшого розгляду.

Алгоритм продовжує цей процес, додаючи до списку відвіданих вершин ті, які мають найменшу відстань від початкової вершини.

Коли черга пріоритетів стає порожньою, алгоритм завершує роботу, тепер можливо відновити найкоротший шлях від початкової вершини до будь-якої іншої, використовуючи інформацію про попередників для кожної вершини, яку зберігали під час виконання алгоритму.

Використання алгоритму Дейкстри дозволяє ефективно оптимізувати ресурси та забезпечувати швидкий доступ до необхідної інформації, що робить його незамінним інструментом у багатьох сферах діяльності.

Калькулятор для розрахунку вартості доставки вантажів використовує алгоритм Дейкстри (рисунок 2.9) для знаходження найкоротшого шляху між містами.

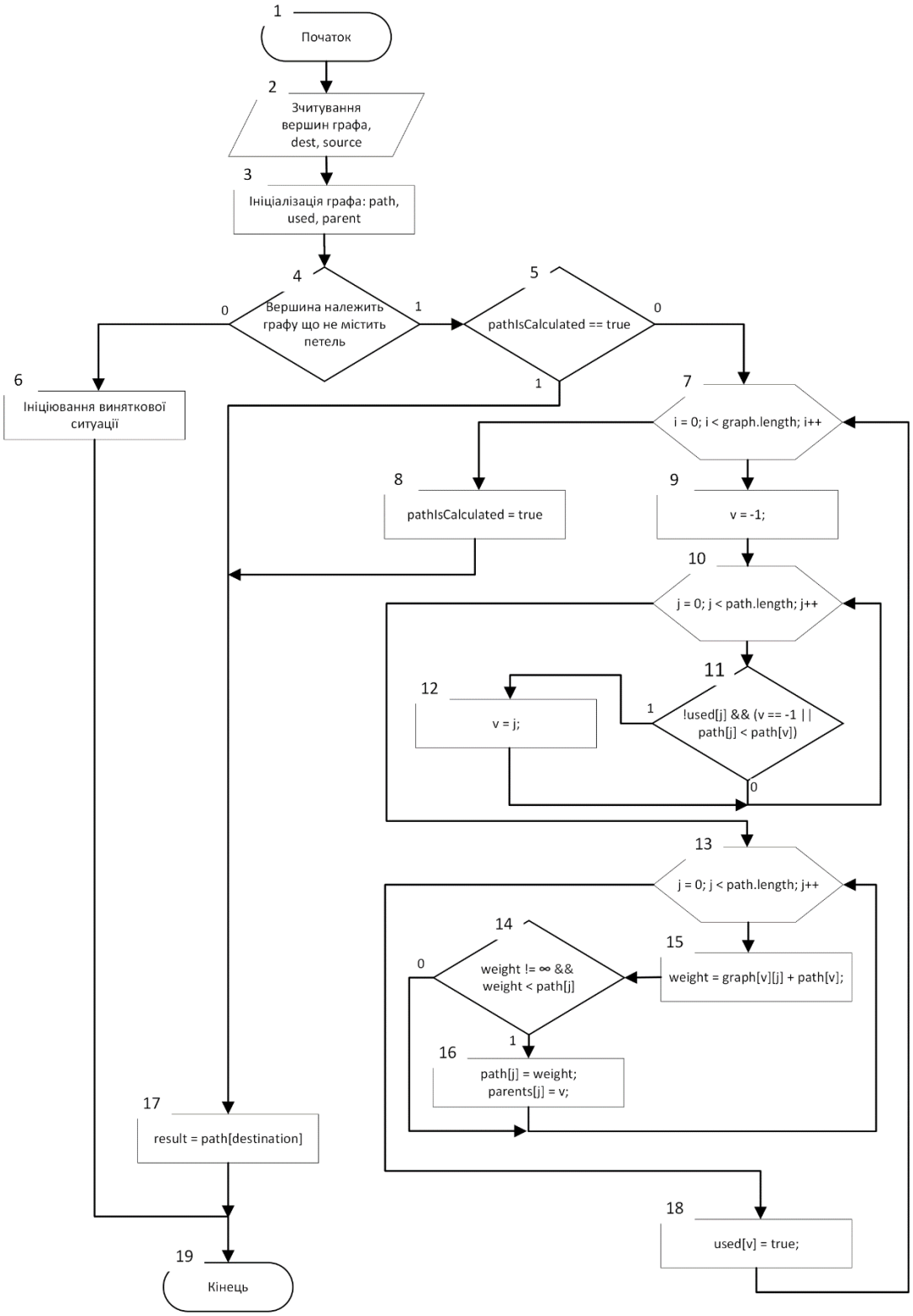

Рисунок 2.9 – Схема пошуку найкоротшого шляху за алгоритмом Дейкстри

Розглянувши дані алгоритми, можна зрозуміти, що розроблена система дає змогу користувачам реєструватися та авторизуватися в ній, а алгоритм Дейкстри розраховувати оптимальну відстань між містами.

2.5 Висновки

У другому розділі проведено значну роботу з розробки інтерфейсу системи. Інтерфейс є ключовим елементом, оскільки він забезпечує зручну та ефективну взаємодію користувача з програмним продуктом. Під час розробки були створені необхідні моделі, які визначають основні концепції та принципи функціонування системи.

Окрім цього, розроблено метод реєстрації та авторизації користувача. Також у розділі були створені блок-схеми алгоритмів роботи, які відображають послідовність дій у системі. Ці блок-схеми допомагають у розумінні логіки роботи програмного продукту та виявленні можливих покращень у процесах взаємодії з користувачем.

Окрім цього, в розділі детально описаний алгоритм Дейкстри. Цей алгоритм важливий для визначення найкоротших шляхів у графі з вагами на ребрах і може бути застосований у багатьох аспектах розробки програмного забезпечення.

3 РОЗРОБКА ПРОГРАМНИХ ЗАСОБІВ СИСТЕМИ УПРАВЛІННЯ ТА ЗАБЕЗПЕЧЕННЯ ДОСТАВКИ ВАНТАЖНИХ ПЕРЕВЕЗЕНЬ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

Для реалізації програмного забезпечення системи по доставці вантажів було проведено варіантний аналіз засобів, які найбільш відповідають потребам. У результаті аналізу було обрано наступний стек технологій та інструментів:

Java – це об’єктно-орієнтована мова програмування загального призначення з простим і зрозумілим синтаксисом, яка підходить для різних платформ. Завдяки широким можливостям, бібліотекам і портативності Java дозволяє створювати ПЗ для різних компаній і сфер: ігри, мобільні застосунки, корпоративні рішення тощо [18].

Java також відома своєю багатопоточністю, що дозволяє одночасно виконувати багато задач одночасно. Це робить її ідеальним вибором для розробки масштабованих та швидкодіючих додатків, які вимагають обробки багатьох операцій паралельно. Ще однією перевагою Java є її вбудовані механізми безпеки, такі як перевірка масивів, керування пам'яттю та відсутність вказівників. Це робить Java відмінним вибором для розробки безпечних додатків, які не вразливі на атаки зловмисників. Однією з ключових особливостей Java є її переносимість. Програми на Java можуть запускатися на будь-якому пристрої, який підтримує віртуальну машину Java (JVM), що робить її ідеальним вибором для розробки кросплатформених додатків. Це дозволяє розробникам писати код один раз і запускати його на різних пристроях без необхідності переписування.

Spring Framework – це платформа з відкритим вихідним кодом для створення веб-додатків з Java як мовою програмування. Він потужний і легкий, але простий у використанні, і забезпечує підтримку легкої розробки додатків Java [19]. Це високопродуктивний інструмент для розробки Java-програм. Він надає широкий

спектр функціональностей, що дозволяє розробникам швидко створювати надійні та ефективні додатки.

Однією з ключових особливостей Spring є підтримка інверсії керування (IoC) і вінконтейнера. Це дозволяє відокремити бізнес-логіку додатку від конфігураційних деталей, що спрощує тестування та підтримку програмного забезпечення. Крім того, Spring пропонує модель програмування аспектами (AOP), яка дозволяє відокремити кросворджування від основної бізнес-логіки додатку.

Ще однією важливою функціональністю Spring є Spring MVC, яка надає можливість створювати веб-додатки з використанням шаблону проектування MVC (Model-View-Controller). Це дозволяє розробникам ефективно розділити бізнес-логіку, представлення та управління даними у веб-додатках.

Spring також пропонує набір модулів для доступу до даних, включаючи Spring Data, який спрощує взаємодію з базами даних, і Spring Security, який надає засоби для забезпечення безпеки додатків.

Узагальнюючи, Spring Framework є потужним і гнучким інструментом для розробки Java-додатків, який дозволяє розробникам швидко створювати надійні та ефективні програми.

PostgreSQL це система управління реляційними базами даних з відкритим вихідним кодом, що використовує модель клієнт-сервер. Він використовує мову SQL для управління даними і запитів до них і може бути доступна через різноманітні протоколи, включаючи TCP / IP і доменні сокети Unix [20].

PostgreSQL – це добре відома система управління базами даних з відкритим вихідним кодом. Вона відома своєю надійністю, можливостями та широким спектром функцій, які вона пропонує.

Однією з ключових особливостей PostgreSQL є підтримка реляційних баз даних. Вона дозволяє зберігати дані у вигляді таблиць, зв'язуючи їх за допомогою ключів. Ця модель даних дозволяє зберігати дані структуровано та ефективно виконувати операції з ними.

Крім того, PostgreSQL підтримує багато різних типів даних, включаючи числа, рядки, дати, час, географічні дані та багато іншого. Це дозволяє розробникам зберігати різноманітні дані у базі даних.

PostgreSQL також відомий своєю високою продуктивністю та масштабованістю.

Вона може обробляти великі обсяги даних та високі навантаження, що робить її ідеальним вибором для великих проектів та додатків.

Отже, стек дозволяє створювати потужні та масштабовані веб-додатки, забезпечуючи високу продуктивність, безпеку та зручний інтерфейс. Використання Java разом зі Spring Framework дозволяє швидко розробити систему та забезпечувати її стабільність і масштабованість. PostgreSQL виступає в якості надійної системи управління базами даних. Загалом, цей стек технологій є добре збалансованим та підходить для високоякісної розробки.

3.2 Розробка бази даних

Для збереження даних користувачів та інформації про вантажі можна використовувати реляційні та нереляційні бази даних. Реляційні бази даних використовують таблиці з чіткою схемою та зв'язками між ними, а також мову запитів SQL. Вони підходять для застосунків, де потрібна цілісність, узгодженість і транзакційність даних, таких як MySQL, PostgreSQL.

Нереляційні бази даних не використовують реляційну модель і зберігають дані в гнучких структурах, таких як документи, ключ-значення, графи або стовпцеві сімейства. Вони не мають чіткої схеми і можуть зберігати дані з різною структурою. Нереляційні бази даних краще масштабуються для обробки великих обсягів неструктурованих даних і підходять для застосунків, де важлива висока доступність та еластичність, таких як MongoDB, Cassandra.

Для розробки серверної частини автоматизованої системи моніторингу стану дерев краще підходить реляційна база даних.

Було розроблено структуру бази даних, яка зображена на рисунку 3.1.

Таблиця delivery_receipts містить інформацію про квитанції доставки. Вона складається з таких полів:

id – унікальний ідентифікатор квитанції доставки.

formation_date – дата формування квитанції.

paid – статус оплати квитанції.

total_price – загальна вартість доставки.

application_id – ідентифікатор заявки, до якої відноситься квитанція.

manager_user_id – ідентифікатор користувача-менеджера, який обробляв доставку.

customer_user_id: – ідентифікатор користувача-замовника, який замовив доставку.

Таблиця dimensions_fares містить інформацію про тарифи за розміри вантажу. Вона складається з таких полів:

id – унікальний ідентифікатор тарифу за розміри вантажу.

dimensions_from – мінімальні розміри вантажу, до яких застосовується цей тариф.

dimensions_to – максимальні розміри вантажу, до яких застосовується цей тариф.

price – вартість доставки за вказаний діапазон розмірів вантажу.

delivery_applications – це таблиця в базі даних, яка містить інформацію про заявки на доставку Вона складається з таких полів:

id – унікальний ідентифікатор заявки на доставку.

receiving_date – дата отримання заявки.

sending_date – дата відправлення заявки.

baggage_id – ідентифікатор вантажу, який буде доставлений.

receiver_address_id – ідентифікатор адреси отримувача вантажу.

sender_address_id – ідентифікатор адреси відправника вантажу.

user_id – ідентифікатор користувача, який зробив заявку на доставку.

state – стан заявки на доставку (наприклад, "очікує на відправлення", "у процесі доставки", "доставлено").

price – вартість доставки для цієї заявки.

Таблиця user_role містить інформацію про ролі користувачів. Вона складається з таких полів:

user_id – ідентифікатор користувача, якому призначена роль.

roles – роль, яка призначена користувачеві.

Таблиця distance_fares містить інформацію про тарифи на доставку залежно від відстані. Вона складається з таких полів:

id – унікальний ідентифікатор тарифу на відстань.

distance_from – початкова відстань для тарифу.

distance_to – кінцева відстань для тарифу.

price – вартість доставки для відстані від distance_from до distance_to.

Таблиця delivered_baggage містить інформацію про доставлені багажі. Вона складається з таких полів:

id – унікальний ідентифікатор доставленого багажу.

type – тип багажу.

volume – об'єм багажу.

weight – вага багажу.

description – опис багажу.

Таблиця users містить інформацію про користувачів. Вона складається з таких полів:

id – унікальний ідентифікатор користувача.

cash – наявність грошей на рахунку користувача.

email – електронна пошта користувача.

name – ім'я користувача.

phone – телефонний номер користувача.

surname – прізвище користувача.

login – логін користувача.

address_id – ідентифікатор адреси користувача.

password – пароль користувача.

Таблиця weight_fares містить інформацію про тарифи за вагу. Вона складається з таких полів:

id – унікальний ідентифікатор тарифу за вагу.

price – ціна за вагу.

weight_from – нижня межа ваги, до якої застосовується цей тариф.

weight_to – верхня межа ваги, до якої застосовується цей тариф.

Таблиця directions містить інформацію про напрямки перевезень. Вона складається з таких полів:

id – унікальний ідентифікатор напрямку перевезень.

distance – відстань між містами у напрямку.

receiver_city_id – ідентифікатор міста-одержувача.

sender_city_id – ідентифікатор міста-відправника.

Таблиця addresses зберігає інформацію про адреси. Вона складається з таких полів:

id – унікальний ідентифікатор адреси.

house_number – номер будинку.

city_id – ідентифікатор міста, до якого належить адреса.

street – назва вулиці.

Таблиця cities містить дані про міста:

id – унікальний ідентифікатор міста.

name – назва міста.

zipcode – поштовий індекс міста.

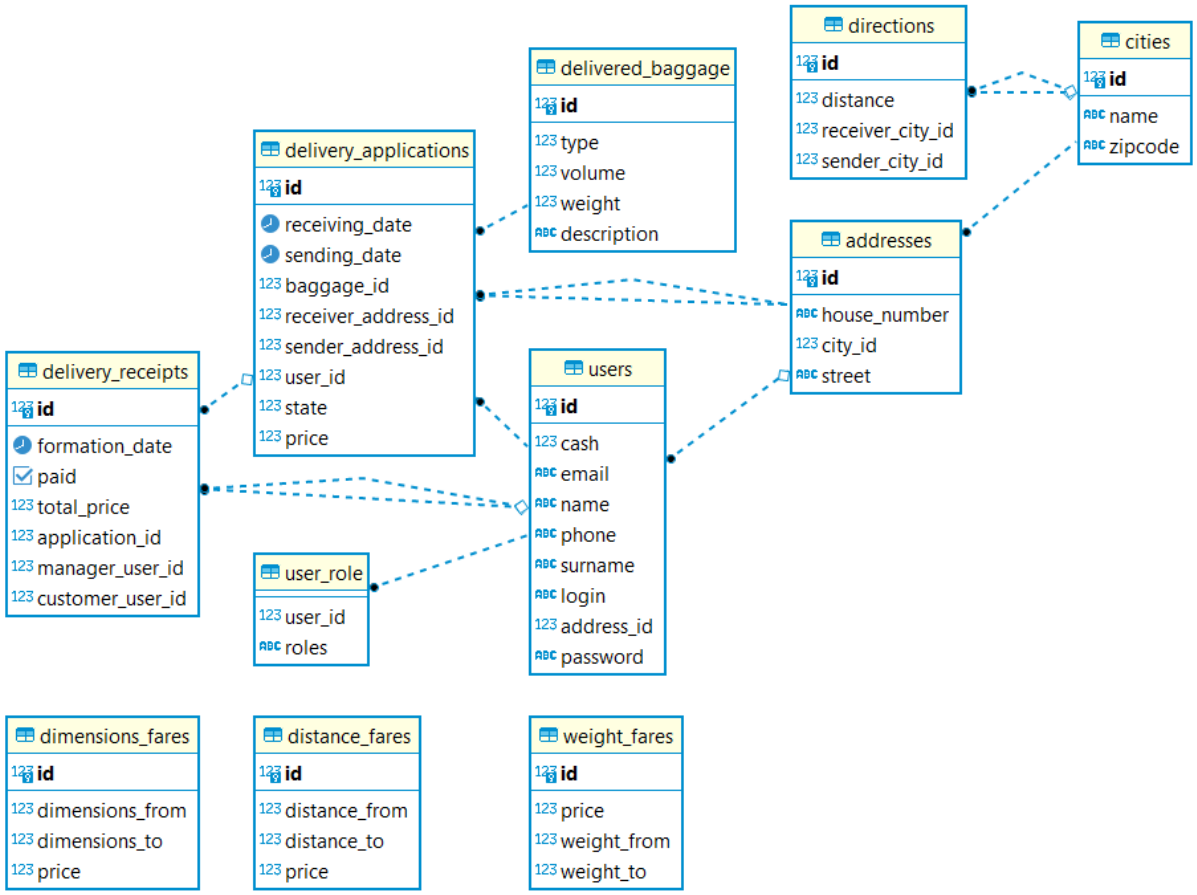

Рисунок 3.1 – Структура бази даних

3.3 Розробка алгоритму прокладання найоптимальнішого маршруту доставки

Для програмних засобів системи управління та забезпечення доставки вантажних перевезень необхідно використати алгоритм який зможе знаходити оптимальні відстані між містами. З таким завдання впорається алгоритм Дейкстри.

У алгоритмі Дейкстри для пошуку найкоротшого шляху від одного вершини до всіх інших використовується орієнтований зважений граф.

1. Орієнтований граф: Граф, де кожне ребро має напрямок. Це означає, що переходити від однієї вершини до іншої можна тільки в напрямку, визначеному ребром.

2. Зважений граф: Кожне ребро графа має вагу або вартість, яка відображається числом. Ця вага представляє вартість переходу від однієї вершини до іншої через це ребро. Наприклад, у дорожньому графі вага може представляти відстань або час подорожі між містами.

3. Найкоротший шлях: Алгоритм Дейкстри знаходить найкоротший шлях від однієї початкової вершини до всіх інших вершин у графі.

4. Динамічне програмування: Алгоритм Дейкстри базується на принципі динамічного програмування, де він поступово оновлює найкращі відомі відстані до кожної вершини, просуваючись через граф.

Код реалізує алгоритм Дейкстри для знаходження найкоротшого шляху в графі з вагами на ребрах. Основна ідея алгоритму полягає в тому, щоб поступово "розгортати" найлегший шлях від стартової вершини до всіх інших вершин графу.

Алгоритм починає з вершини 0 і на кожному кроці вибирає найлегше доступне ребро. Для кожної вершини він обчислює найкоротший шлях до кожної іншої вершини через цю вершину. Якщо знайдений шлях коротший, ніж поточний найкоротший шлях до цієї вершини, то шлях і поточні значення шляхів і батьківських вершин оновлюються.

Код використовує матрицю graph для зберігання ваг ребер графу, масив path для зберігання поточних найкоротших відстаней до кожної вершини, масив used для відстеження відвіданих вершин, та масив parents для зберігання батьківських вершин на шляху до кожної вершини.

Фрагмент коду (рисунок 3.2) відповідає за побудову найкоротшого шляху в графі від вершини-джерела до заданої вершини-призначення. Метод buildShortestRoute використовується для побудови такого шляху та повертає список вершин, які потрібно пройти для досягнення заданої вершини. Він перевіряє, чи був викликаний метод calculateMinDistance для обчислення найкоротшого шляху, і, якщо ні, викликає його для цієї вершини. Потім він викликає метод buildRoute для побудови шляху за допомогою масиву батьківських вершин.

Метод requireAbsenceSelfLoop перевіряє, чи не є вершина-призначення такою ж, як вершина-джерело, оскільки це недопустимо для побудови шляху.

Метод buildRoute використовується для побудови шляху від заданої вершини до вершини-джерела за допомогою масиву батьківських вершин. Він працює шляхом переходу по батьківським вершинам від заданої до вершини-джерела, додаючи кожну вершину до списку шляху. Після цього він перевертає список, щоб він відображав послідовність вершин від вершини-джерела до заданої вершини.

Метод buildShortestRoute використовується для побудови такого шляху та повертає список вершин, які потрібно пройти для досягнення заданої вершини. Він перевіряє, чи був викликаний метод calculateMinDistance для обчислення найкоротшого шляху, і, якщо ні, викликає його для цієї вершини. Потім він викликає метод buildRoute для побудови шляху за допомогою масиву батьківських вершин.

Метод requireAbsenceSelfLoop перевіряє, чи не є вершина-призначення такою ж, як вершина-джерело, оскільки це недопустимо для побудови шляху.

Метод buildRoute використовується для побудови шляху від заданої вершини до вершини-джерела за допомогою масиву батьківських вершин. Він працює шляхом переходу по батьківським вершинам від заданої до вершини-джерела, додаючи кожну вершину до списку шляху. Після цього він перевертає список, щоб він відображав послідовність вершин від вершини-джерела до заданої вершини.

Також орієнтований зважений граф представлено на рисунку 3.2.

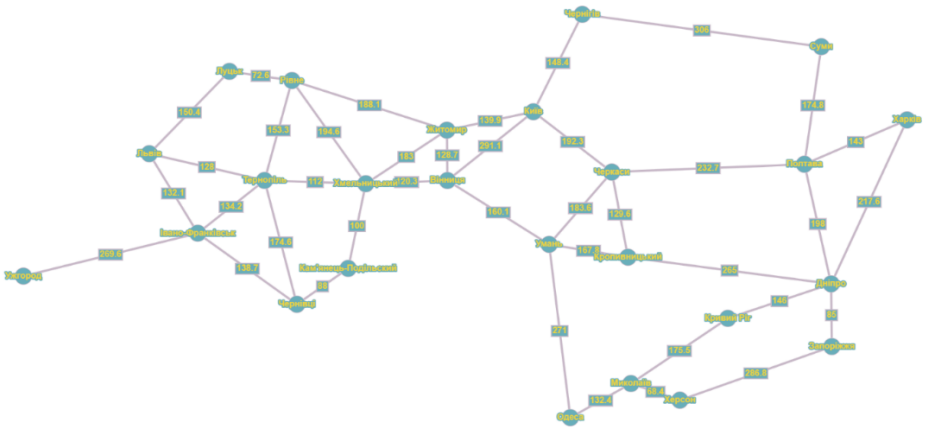

Рисунок 3.2 – Ілюстрація зваженого графу

3.4 Розробка калькулятора для обчислення вартості доставки

Клас калькулятора служить для обчислення вартості доставки. Він містить методи для розрахунку вартості доставки в залежності від відстані, ваги та об'єму вантажу, а також для розрахунку вартості доставки між двома адресами. Клас використовує різні сервіси (DistanceFareService, WeightFareService, DimensionsFareService, DirectionDeliveryService, CityService) для отримання необхідних даних і розрахунків. Методи класу дозволяють виконувати розрахунки для різних сценаріїв доставки та отримувати вартість доставки для відомих міст.

Клас також містить логіку для перевірки коректності даних, які вводить користувач. Наприклад, перевірка того, щоб міста відправлення і призначення були різними, або перевірка існування міста за його ідентифікатором (див. рисунок 3.3).

Додатково, в класі використовуються анотації Spring Framework, такі як @Service і @Autowired, для впровадження інших компонентів програми та автоматичного введення залежностей.

Також клас служить для обчислення вартості доставки. Він містить методи для розрахунку вартості доставки в залежності від відстані, ваги та об'єму вантажу, а також для розрахунку вартості доставки між двома адресами. Клас використовує різні сервіси (DistanceFareService, WeightFareService, DimensionsFareService, DirectionDeliveryService, CityService) для отримання необхідних даних і розрахунків. Методи класу дозволяють виконувати розрахунки для різних сценаріїв доставки та отримувати вартість доставки для відомих міст.

Цей клас також містить логіку для перевірки коректності даних, які вводить користувач. Наприклад, перевірка того, щоб міста відправлення і призначення були різними, або перевірка існування міста за його ідентифікатором.

Додатково, в класі використовуються анотації Spring Framework, такі як @Service і @Autowired, для впровадження інших компонентів програми та автоматичного введення залежностей.

3.5 Метод зберігання та отримання даних про відстань між містами

У даному підрозділі описано метод зберігання та отримання даних про відстані між містами на основі формування матриці суміжностей орієнтованого зваженого графу у базі даних. Цей метод забезпечує ефективний перехід від реляційної моделі до об'єктно-орієнтованої за допомогою підходу ORM мапінгу, що є необхідним для оптимізованого алгоритму пошуку найкоротшого шляху між двома вершинами графу.

Розробка бази даних.

Перший етап реалізації методу полягає у створенні структури бази даних для зберігання інформації про міста та відстані між ними.

Для цього необхідно створити дві основні таблиці:

Таблиця "Міста":

Поле id (унікальний ідентифікатор міста, первинний ключ).

Поле назва (назва міста).

Таблиця "Ребра":

Поле місто_від (ідентифікатор міста-відправника, зовнішній ключ, що посилається на id таблиці "Міста").

Поле місто_до (ідентифікатор міста-призначення, зовнішній ключ, що посилається на id таблиці "Міста").

Поле відстань (вага ребра, що представляє відстань між містами).

Формування матриці суміжностей

На основі даних, що зберігаються у таблицях бази даних, формується матриця суміжностей. Кожен елемент матриці A[i][j] представляє вагу (відстань) між містами i та j. Якщо між містами немає прямого шляху, відповідний елемент матриці встановлюється рівним нескінченності (або іншим значенням, що позначає відсутність зв'язку).

Використання ORM для мапінгу

Для забезпечення зручного доступу до даних у програмі використовується ORM (Object-Relational Mapping). Об'єктно-реляційне відображення (ORM) – це техніка, яка використовується для створення "моста" між об'єктно-орієнтованими програмами і, в більшості випадків, реляційними базами даних. Іншими словами, ORM можна розглядати як прошарок, що з'єднує об'єктно-орієнтоване програмування (ООП) з реляційними базами даних [21].

Це дозволяє мапувати таблиці бази даних на об'єкти у програмному середовищі. Використання ORM забезпечує:

Зв'язок між об'єктами міст та ребер, що представляють відстані між ними.

Можливість здійснювати складні запити до бази даних за допомогою об'єктно-орієнтованих методів.

Оптимізація алгоритму

Використання об'єктно-орієнтованих структур даних спрощує реалізацію та оптимізацію алгоритму пошуку найкоротшого шляху. У даному проекті застосовується алгоритм Дейкстри, який є ефективним для пошуку найкоротших шляхів у графах з не негативними вагами ребер. Завдяки ORM дані про міста та відстані між ними можуть бути легко інтегровані у цей алгоритм.

Запропонований метод дозволяє ефективно зберігати та отримувати дані про відстані між містами, використовуючи матрицю суміжностей та ORM мапінг. Це забезпечує зручний перехід від реляційної моделі до об'єктно-орієнтованої та сприяє оптимізації алгоритмів пошуку найкоротших шляхів.

3.6 Висновки

У третьому розділі було розроблено алгоритм прокладання найоптимальнішого маршруту доставки, розроблено базу даних, метод зберігання та отримання даних про відстань між містами та калькулятор вартості доставки. Було проведено варіантний аналіз та обґрунтовано вибір засобів реалізації розробки а саме: Java, Spring, PostgreSQL.

4 ТЕСТУВАННЯ ПРОГРАМНИХ ЗАСОБІВ СИСТЕМИ УПРАВЛІННЯ ТА ЗАБЕЗПЕЧЕННЯ ДОСТАВКИ ВАНТАЖНИХ ПЕРЕВЕЗЕНЬ

4.1 Тестування інтерфейсу

Для тестування програмного забезпечення по управлінню доставкою вантажів можна використовувати різні методики тестування, включаючи функціональне та навантажувальне тестування, тестування безпеки та користувацького інтерфейсу. Функціональне тестування включає перевірку правильності роботи функцій програмного забезпечення, таких як реєстрація, вхід в систему, створення заявок на доставку та інші операції. Навантажувальне тестування визначає, як програмне забезпечення працює при великому навантаженні, перевіряючи його швидкодію та стійкість. Тестування безпеки включає в себе аналіз на вразливості та захист від атак. Тестування користувацького інтерфейсу допомагає визначити, наскільки програмне забезпечення є зручним у використанні для користувачів.

Функціональне тестування. Цей тип тестування перевіряє, чи відповідає функціонал програмного забезпечення вимогам. Це включає перевірку реєстрації користувачів, вхід в систему, створення, редагування та видалення заявок на доставку, обробку платежів та інші операції. Мета - переконатися, що кожна функція працює правильно та відповідає вимогам специфікації.

Навантажувальне тестування. Цей вид тестування визначає, як програмне забезпечення працює під навантаженням. Це може включати тестування швидкості завантаження сторінок, відповіді сервера на велику кількість запитів та інші аспекти швидкості та стійкості програмного забезпечення при великому трафіку.

Тестування безпеки. Цей тип тестування оцінює рівень захищеності програмного забезпечення від потенційних загроз безпеці. Воно включає в себе аналіз вразливостей, перехоплення даних, атаки на автентифікацію та авторизацію, перевірку відповідності вимогам щодо безпеки.

Тестування користувацького інтерфейсу. Цей вид тестування перевіряє, наскільки зручний та легкий у використанні інтерфейс програмного забезпечення. Мета – забезпечити, щоб користувачі могли легко здійснювати необхідні дії без зайвих ускладнень та запитань.

На рисунку 4.1 зображено тестування сторінки входу користувача головної сторінки системи.

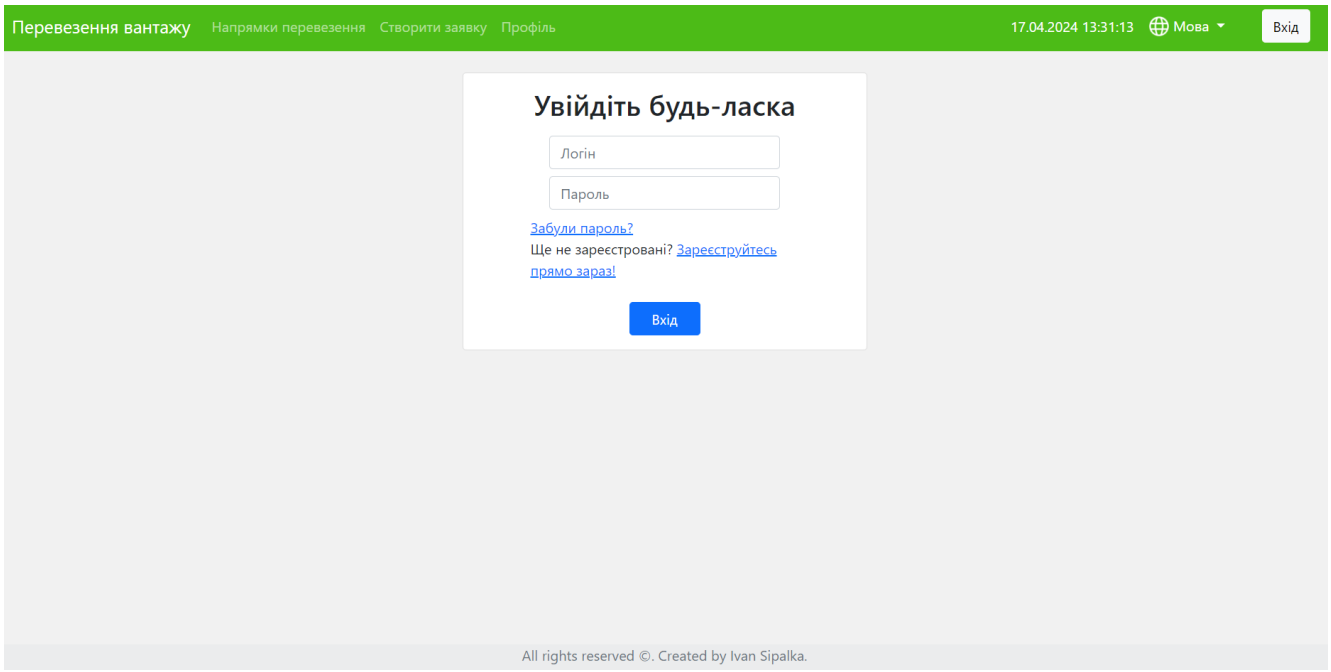

Рисунок 4.1 – Сторінка входу користувача

Перевірка поля "Логін": перевіримо, що користувач може вводити свій логін у відповідному полі. Перевіримо, чи відображається підказка або повідомлення про помилку, якщо поле залишено порожнім або введено некоректний логін.

Перевірка поля "Пароль": перевіримо, що користувач може вводити свій пароль у відповідному полі. Перевіримо, чи відображається підказка або повідомлення про помилку, якщо поле залишено порожнім або введено некоректний пароль.

Натискання на кнопку "Увійти": перевіримо, що при натисканні на кнопку "Увійти" форма входу перевіряє введені дані. Перевіримо, чи відображається повідомлення про невірний логін або пароль у випадку некоректних даних.

Забезпечення безпеки: перевіримо, що пароль введений користувачем не відображається як текст, а приховується за символами зірочками або крапками.

Перехід до відновлення паролю: перевіримо, що користувач може перейти на сторінку відновлення паролю, якщо він забув свій пароль. Перевіримо, чи працює ця функція

На рисунку 4.2 зображено тестування калькулятора вартості перевезення.

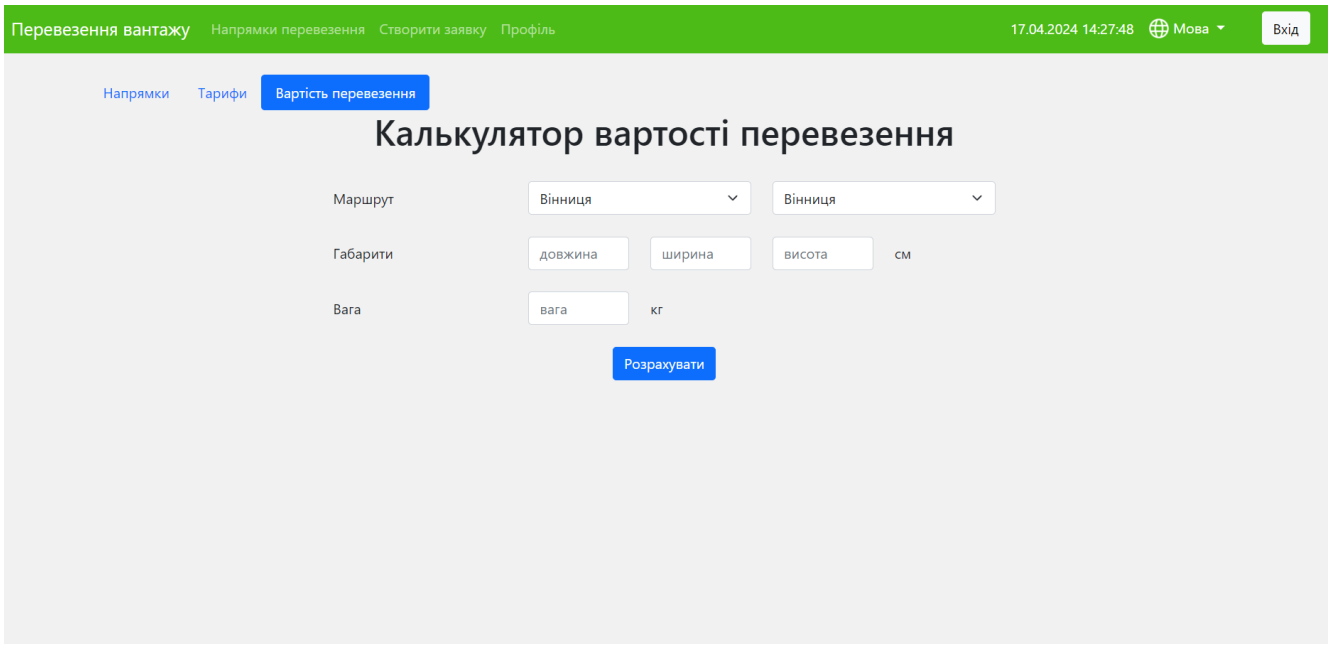

Рисунок 4.2 – Калькулятор вартості перевезення

Перевірка введення ваги. Перевіримо, що користувач може ввести вагу відправлення у відповідному полі. Перевіримо, чи відображається підказка або повідомлення про помилку, якщо поле залишено порожнім або введено некоректне значення.

Натискання на кнопку "Розрахувати". Перевіримо, що при натисканні на кнопку "Розрахувати" калькулятор коректно обчислює вартість перевезення на основі введених даних.

Перевірка результату. Перевіримо, що вартість перевезення, визначена калькулятором, відображається коректно на сторінці.

Тестування валідації. Перевіримо, як калькулятор обробляє некоректні введені дані, такі як введення тексту замість чисел або введення від'ємних значень.

Тестування динамічних оновлень цін. Якщо ціни на перевезення змінюються з часом, перевіримо, що калькулятор коректно оновлює ціни і відображає актуальну вартість перевезення.

На рисунку 4.3 зображено тестування калькулятора вартості перевезення.

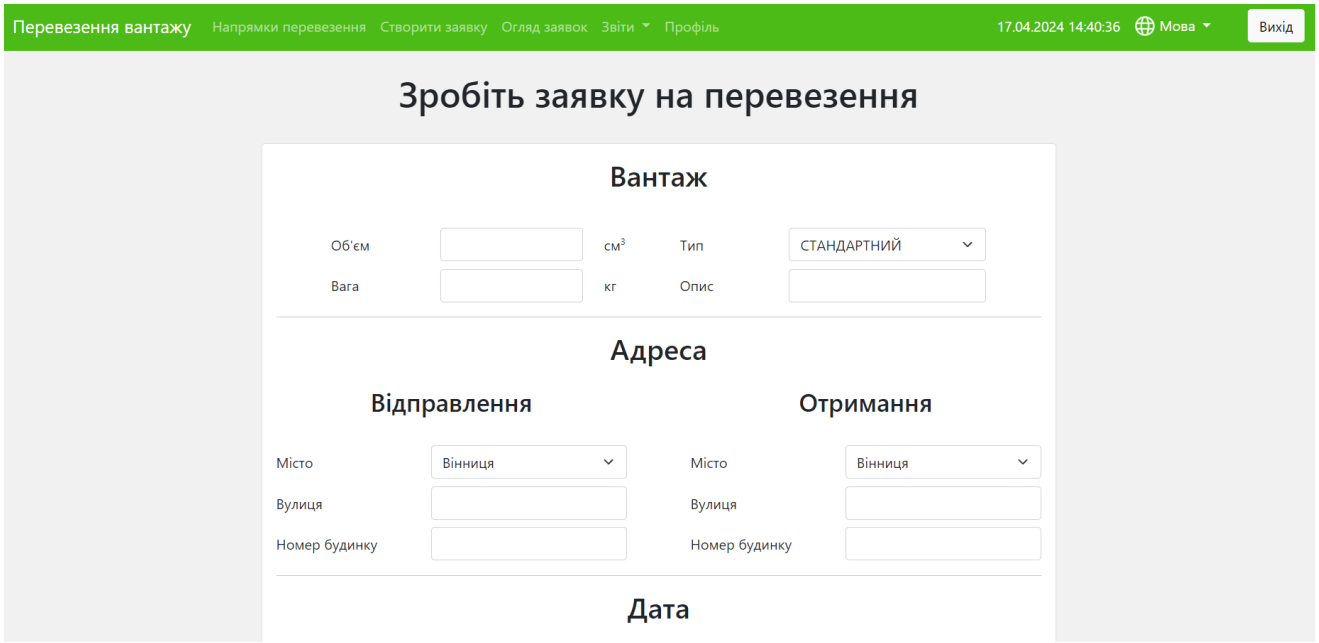

Рисунок 4.3 – Сторінка створення заявки

Перевірка введення даних. Перевіримо, чи користувач може ввести всі необхідні дані для створення заявки, такі як вага відправлення, об'єм, адреса відправника та отримувача, дата відправлення тощо.

Перевірка валідації даних. Перевіримо, як сторінка обробляє некоректні введені дані, такі як введення тексту замість чисел або введення недійсної дати.

Натискання на кнопку "Подати заявку". Перевіримо, що при натисканні на кнопку "Подати заявку" заявка створюється і зберігається в системі.

Повідомлення про успішне створення заявки. Перевіримо, що користувач отримує повідомлення про успішне створення заявки.

Перевірка зміни статусу заявки. Перевіримо, що статус заявки автоматично змінюється від "нова" до "очікує підтвердження" після створення.

Перевірка відображення історії заявок. Перевіримо, що сторінка коректно відображає історію заявок користувача, включаючи всі створені заявки та їх поточний статус.

Тестування відміни заявки. Перевіримо, чи користувач може відмінити створену заявку і як ця дія відображається в системі.

Підтримка різних типів вантажу. Перевіримо, що коректно відображається і оброблюється інформація про тип вантажу в заявці.

4.2 Тестування бази даних

Проведемо тестування бази даних за допомогою ряду тестів для перевірки правильності та ефективності її роботи.

1. Тест на з'єднання з базою даних. Переконаймося, що можна встановити з'єднання з базою даних і база даних доступна для взаємодії.

2. Тест на створення таблиць. Перевіримо, що всі таблиці, необхідні для функціонування програмного забезпечення, коректно створені і мають правильну структуру.

3. Тест на вставку даних. Додамо деякі тестові дані до таблиць і переконаймося, що вони вставлені коректно і без помилок.

4. Тест на вибірку даних. Виберемо деякі дані з таблиць і перевіримо, що вони вибираються коректно за допомогою запитів SQL.

5. Тест на оновлення даних. Оновимо деякі дані в таблицях і переконаймося, що оновлення відбувається коректно.

6. Тест на видалення даних. Видалимо деякі дані з таблиць і перевіримо, що вони видаляються коректно.

7. Тест на обмеження цілісності даних. Перевіримо, що обмеження цілісності даних, такі як унікальність ключів чи зовнішні ключі, працюють коректно і не допускають некоректних операцій з даними.

Протестуємо базу даних, яка зберігає тарифи на доставку:

Перевірка наявності таблиці dimesions_fares: переконаймося, що таблиця для зберігання тарифів за розмірами існує.

Тест на вставку даних, додамо декілька тестових записів у таблицю dimesions_fares для різних діапазонів розмірів із відповідними тарифами.

Тест на вибірку даних, виберемо деякі записи з таблиці dimesions_fares і переконаймося, що дані вибираються коректно.

Тест на видалення даних, видалимо деякі тарифи із таблиці і переконаймося, що вони видаляються коректно.

Тест на обмеження цілісності даних, перевіримо, що встановлені обмеження (наприклад, унікальність діапазонів розмірів) працюють коректно і не допускають некоректних операцій з даними.

Тест на ефективність, виміряємо час виконання запитів до таблиці dimesions_fares і переконаймося, що вони виконуються швидко та ефективно.

Тест на безпеку, переконаймося, що дані у таблиці dimesions_fares захищені від несанкціонованого доступу і можливих атак.

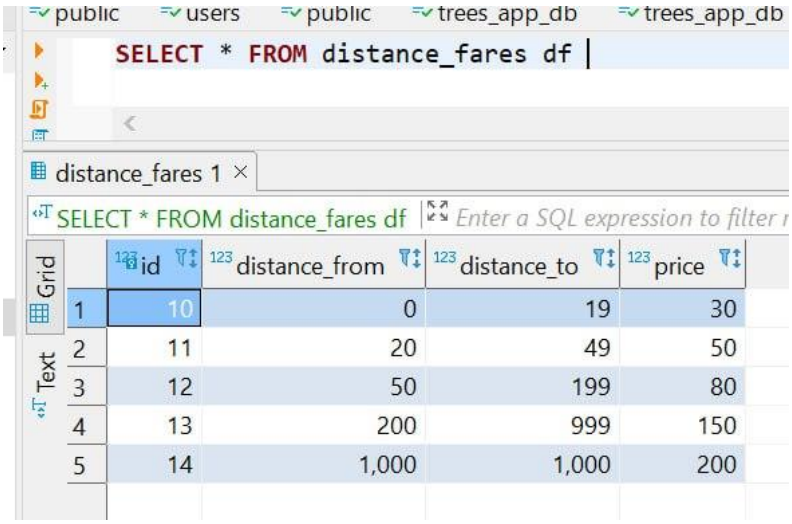

Рисунок 4.3 – Тестування бази даних тарифів

4.3 Тестування алгоритму Дейкстри

Для тестування було використано оптимізований і стандартний алгоритм, щоб порівняти продуктивність створено великий граф через метод generateLargeGraph, який генерує великий граф з 1000 вузлів. Кожне ребро має випадкову вагу між 1 і 100. Обидва алгоритми реалізовано на мові програмування Java, а їх продуктивність виміряно шляхом обчислення часу виконання у наносекундах.

```
Оптимізований алгоритм Дейкстри знайшов найкоротший шлях за 19721700 наносекунд
Найкоротший шлях: 4.46308543084327
Стандартний алгоритм Дейкстри знайшов найкоротший шлях за 26673200 наносекунд
Найкоротший шлях: 4.46308543084327
```

Рисунок 4.4 – Тестування алгоритму Дейкстри

Отже, оптимізований алгоритм працює на приблизно 25.99% швидше за стандартний алгоритм.

4.4 Висновки

У результаті тестування інтерфейсу програмного забезпечення для управління доставкою вантажів було виявлено, що всі функції працюють коректно і інтерфейс є зрозумілим та зручним для користувачів.

Щодо тестування бази даних, встановлено, що дані зберігаються правильно, SQL-запити виконуються швидко, а база даних інтегрується з іншими компонентами програмного забезпечення без проблем. Також проведено тестування алгоритму Дейкстри яке підтверджує, що алгоритм працює швидше за стандартний алгоритм.

В цілому, програмне забезпечення готове до використання і відповідає вимогам ефективності та надійності.

ВИСНОВКИ

У результаті виконання бакалаврської дипломної роботи, було реалізовано програмне забезпечення для управління доставкою вантажів. Серверна частина забезпечує функціонал реєстрації та авторизації користувача, роботу з базою даних та сховищем даних. Програмне забезпечення дозволяє переглядати ціну тарифів, що внесені у систему, використовувати калькулятор для обчислення вартості доставки, переглядати історію доставок, створювати заявки для доставки.

Для розробки було використано середовище програмної розробки IntelliJ IDEA.

Під час виконання бакалаврської дипломної роботи було проаналізовано стан програмних засобів систем управління та забезпечення доставки вантажних перевезень.

Було розглянуто основні аналоги програмного продукту та визначено їхні недоліки, порівняно з власною реалізацією, та встановлено основні задачі.

Під час аналізу технологій розробки було обґрунтовано вибір мови програмування Java та фреймворку Spring. Також було розглянуто переваги використання бази даних PostgreSQL.

У бакалаврській дипломній роботі було:

- розроблено зручний графічний інтерфейс;
- розроблено модель роботи серверної частини;
- розроблено алгоритми роботи системи;
- розроблено базу даних;
- розроблено калькулятор для обчислення вартості доставки;
- проведено тестування програмного забезпечення.

Також було розроблено схему роботи серверної частини. Крім того, було розроблено модель алгоритму роботи авторизації та реєстрації, а також алгоритм Дейкстри.

Під час тестування програмного продукту було підтверджено, що всі основні функції працюють належним чином і відповідають вимогам, визначеним у технічному завданні. Також, під час проведення тестів, були враховані різноманітні сценарії використання програми, що дозволило виявити та усунути всі виявлені помилки та недоліки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Волинець Л. Лібералізація міжнародних автомобільних перевезень – новий імпульс розвитку транспортної галузі. Економіка транспортного комплексу. 2021. Вип. 37. С. 161–176.

2. Ricci S., Abdelbary A., Elgazzar S., Bayoumi E. The Role of Road Transport Infrastructure Investmentson Logistics Performance: A Research Agenda. International Business Logistics Journal (IBL). 2021. Volume 1, Issue 2. DOI: http://dx.doi.org/10.21622/IBL.2021.01.2.016

3. Сіпалка І. С., Ткаченко О. М. Модифікація алгоритму Дейкстри для врахування специфіки управління доставкою вантажів. Матеріали конференції «LIII Науково-технічна конференція підрозділів Вінницького національного технічного університету (2024) », Вінниця, 2024. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2024/paper/view/20833

4. SAP Transportation Management [Електронний ресурс] – Режим доступу до ресурсу: http://surl.li/rwsbi.

5. Transportation Management System [Електронний ресурс] – Режим доступу до ресурсу: https://www.oracle.com/cis/scm/logistics/transportation-management/what-is-transportation-management-system/

6. Descartes Transportation Management [Електронний ресурс] – Режим доступу до ресурсу: https://www.business-software.com/product/descartes-transportation-management/

7. Алгоритм Дейкстри [Електронний ресурс] – Режим доступу до ресурсу: https://reporter.zp.ua/v-chomu-polyagaye-algorytm-dejkstry.html

8. Алгоритм A* [Електронний ресурс] – Режим доступу до ресурсу: https://www.codecademy.com/resources/docs/ai/search-algorithms/a-star-search

9. Алгоритм Флойда-Уоршелла [Електронний ресурс] – Режим доступу до ресурсу: https://www.baeldung.com/cs/floyd-warshall-shortest-path

10. Серверна частина [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Introduction

11. Amazon Web Services [Електронний ресурс] – Режим доступу до ресурсу: https://info.nic.ua/uk/blog-uk/shho-take-amazon-web-services/

12. Amazon Virtual Private Cloud [Електронний ресурс] – Режим доступу до ресурсу: https://eska.global/products/aws-amazon-vpc

13. Internet Gateway [Електронний ресурс] – Режим доступу до ресурсу: https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Internet_Gateway.html

14. Amazon Elastic Compute Cloud (EC2) [Електронний ресурс] – Режим доступу до ресурсу: https://eska.global/products/amazon-ec2

15. Amazon Neptune [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/en/neptune/features/

16. Amazon RDS [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/us

17. Amazon Elastic Block Store [Електронний ресурс] – Режим доступу до ресурсу: https://www.amazonaws.cn/en/ebs/

18. Що таке Java? [Електронний ресурс] – Режим доступу до ресурсу: https://goit.global/ua/articles/shcho-take-java-i-de-vona-vykorystovuietsia/

19. Що таке Spring Framework? [Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/uk/spring-tutorial.html

20. Мова структурованих запитів postgres [Електронний ресурс] – Режим доступу до ресурсу: https://cqr.company/ua/wiki/protocols/postgres-structured-query-language/

21. What is an ORM? [Електронний ресурс] – Режим доступу до ресурсу: https://www.freecodecamp.org/news/what-is-an-orm-the-meaning-of-object-relational-mapping-database-tools/

Додаток А – Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
« 12 » березня 2024 р.

Технічне завдання

на бакалаврську дипломну роботу «Розробка програмного забезпечення для управління системою доставки вантажних перевезень з оптимізацією маршрутів»

за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
к.т.н., доц. О.М. Ткаченко
« 12 » березня 2024 року.
Виконав:
студент гр. 1ПІ-20б І.С. Сіпалка
« 12 » березня 2024 року.

м. Вінниця – 2024 рік

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка програмного забезпечення для управління системою доставки вантажних перевезень з оптимізацією маршрутів».

Галузь застосування – логістичні перевезення.

2. Підстава для розробки.

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – наказ №80 від «11» березня 2024 р по ВНТУ про закріплення тем БДР.

3. Мета та призначення розробки.

Метою дослідження є зменшення часу, необхідного для пошуку найкоротшого маршруту доставки вантажу за рахунок розробки програмного забезпечення для управління доставкою вантажних перевезень.

Призначення роботи – розробка програмного забезпечення для для управління системою доставки вантажних перевезень.

4. Вихідні дані для проведення НДР

1. Ricci S., Abdelbary A., Elgazzar S., Bayoumi E. The Role of Road Transport Infrastructure Investmentson Logistics Performance: A Research Agenda. International Business Logistics Journal (IBL). 2021. Volume 1, Issue 2. DOI: http://dx.doi.org/10.21622/IBL.2021.01.2.016

2. Алгоритм Дейкстри [Електронний ресурс] – Режим доступу до ресурсу: https://reporter.zp.ua/v-chomu-polyagaye-algorytm-dejkstry.html

3. Серверна частина [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Introduction

4. What is an ORM? [Електронний ресурс] – Режим доступу до ресурсу: https://www.freecodecamp.org/news/what-is-an-orm-the-meaning-of-object-relational-mapping-database-tools/

5. Технічні вимоги

Двоядерний процесор з тактовою частотою від 1 ГГц. Об’єм оперативної пам’яті 4ГБ. Розмір жорсткого диску 128 ГБ. Операційна система Windows 7 або вище.

6. Конструктивні вимоги.

Програмне забезпечення має відповідати ергономічним та технічним вимогам. Текстова та графічна документація повинна відповідати стандартам України.

7. Перелік технічної документації, що пред’являється по закінченню робіт:

– пояснювальна записка до бакалаврської дипломної роботи;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз розвитку програмного забезпечення для управління системою доставки вантажних перевезень	14.03.2024 – 24.03.2024	
2	Розробка структури та алгоритмів програмного продукту	25.03.2024 – 07.04.2024	
3	Варіантний аналіз і обґрунтування вибору засобів реалізації програмного забезпечення	08.04.2024 – 18.04.2024	
4	Розробка системи	19.04.2024 – 19.05.2024	
5	Тестування системи	20.05.2024 – 24.05.2024	
6	Оформлення матеріалів до захисту БДР	01.06.2024 – 13.06.2024	

10 Порядок контролю та прийняття.

Всі етапи бакалаврської роботи контролюються науковим керівником згідно плану по виконанню роботи. Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту.

64

Додаток Б – Протокол перевірки на плагіат

ПРОТОКОЛ
ПЕРЕВІРКИ БАКАЛАВРСЬКОЇ ДИПЛОМНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка програмного забезпечення для управління системою доставки вантажних перевезень з оптимізацією маршрутів

Тип роботи: БДР

Підрозділ: кафедра програмного забезпечення, ФІТКІ

Науковий керівник: Ткаченко О. М.

Оригінальність	94,7%
Схожість	5,3%

Аналіз звіту подібності

▪ **Запозичення, виявлені у роботі, оформленні коректно і не містять ознак плагіату.**

☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цілісності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховання недобросовісних запозичень.

Особа, відповідальна за перевірку Черноволик Г.О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи Сіпалка І.С.

Керівник роботи Ткаченко О. М.

Додаток В – Лістинг програми

```
package com.epam.cargo.algorithms;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;


public class Dijkstra {

    private double[][] graph;

    private int source;

    private double[] path;
    private boolean[] used;
    private int[] parents;

    private boolean done = false;

    /**
     * Sets graph object for further work.<br>
     * Checks data validity.
     * @param graph adjacency matrix
     * @param source vertex from which we are about to calculate
     * */
    public Dijkstra(double[][] graph, int source){
        requireValidMatrix(graph);
        requirePresenceInGraph(graph, source);

        this.graph = graph;
        this.source = source;

        initializePath();
        initializeUsed();
        initializeParents();
    }

    /**
     * Calculate a minimum distance between source node all other nodes.
     * @return calculated smallest distance between source and given destination,
     * if no way exists returns Double.POSITIVE_INFINITIVE
     * @throws IllegalArgumentException when destination node doesn't belong to
the graph
     * or it's the same with source node
     * */
    public double calculateMinDistance(int destination){
        requirePresenceInGraph(graph, destination);
        requireAbsenceSelfLoop(destination);
        if (!done) {

            for (int i = 0; i < graph.length; i++) {
                int v = -1;
                //finding the vertex to which the lightest edge now leads
                for (int j = 0; j < path.length; j++) {
                    if (!used[j] && (v == -1 || path[j] < path[v])) {
                        v = j;
```

```
                    }
                }
                //relaxation
                for (int j = 0; j < graph[v].length; j++) {
                    double weight = graph[v][j];    //шлях від ребра v до ребра j

                    //if edge is present
                    if (weight != Double.POSITIVE_INFINITY) {
                        weight += path[v];
                        if (weight < path[j]) {
                            path[j] = weight;
                            parents[j] = v;
                        }
                    }
                }

                used[v] = true;
            }
        }
        done = true;
        return path[destination] != Double.POSITIVE_INFINITY ? path[destination] :
Double.POSITIVE_INFINITY;
    }

    /**
     * Build smallerRoute from source node to destination,
     * if calculateMinDistance hasn't been called, it will be called
     * @return an List with integer ordinary nodes
     * like a shortest route to go between source and destination
     * */
    public List<Integer> buildShortestRoute(int destination){
        requirePresenceInGraph(graph, destination);
        requireAbsenceSelfLoop(destination);
        if(!done){
            calculateMinDistance(destination);
        }
        return buildRoute(source, destination, parents);
    }

    private void requireAbsenceSelfLoop(int destination) {
        if (destination == source){
            throw new IllegalArgumentException("Destination node cannot be the
same with source");
        }
    }

    private static List<Integer> buildRoute(int from, int to, int[] parents){
        int v = to;
        List<Integer> route = new ArrayList<>();
        for (int i = 0; v != from && i < parents.length; i++) {
            route.add(v);
            v = parents[v];
        }
        if (v != from){
            throw new IllegalStateException("Couldn't build any route between
edges!");
        }
        route.add(v);
        Collections.reverse(route);
        return route;
    }
```

```
    private void initializeParents() {
        this.parents = new int[totalNodes()];
    }

    private void initializeUsed() {
        this.used = new boolean[totalNodes()];
        Arrays.fill(used, false);
    }

    private void initializePath() {
        this.path = new double[totalNodes()];
        Arrays.fill(path, Double.POSITIVE_INFINITY);
        this.path[source] = 0.0;
    }


    private static void requirePresenceInGraph(double[][] graph, int node) {
        if (node >= graph.length){
            throw new IllegalArgumentException("Given node doesn't belong to the
graph");
        }
    }

    private static void requireValidMatrix(double[][] matrix){
        requireSquareMatrix(matrix);
        requireOnlyPositiveValues(matrix);
    }

    private static void requireSquareMatrix(double[][] matrix){
        for (double[] row : matrix) {
            if (row.length != matrix.length) {
                throw new IllegalArgumentException("Given matrix must be only
square");
            }
        }
    }

    private static void requireOnlyPositiveValues(double[][] matrix){
        for (double[] row : matrix) {
            for (double cell : row) {
                if (cell < 0.0){
                    throw new IllegalArgumentException("Dijkstra algorithm works
only with positive values");
                }
            }
        }
    }

    private int totalNodes(){
        return graph.length;
    }
}
public enum ModelErrorAttribute {
    DUPLICATE_PASSWORD("duplicatePasswordErrorMessage"),
    LOGIN("loginErrorMessage"),
    PASSWORD("passwordErrorMessage"),
    NAME("nameErrorMessage"),
    PHONE("phoneErrorMessage"),
    SURNAME("surnameErrorMessage"),
```

```
    RECEIVING("receivingDateErrorMessage"),
    CITY_DIRECTION("invalidCityDirectionErrorMessage"),
    PAYING("payingErrorMessage"),
    ABSENT_CITY("noExistingCityMessage");

    private final String attr;

    ModelErrorAttribute(String attr) {
        this.attr = attr;
    }

    public String getAttr() {
        return attr;
    }
}
public class AddressService {

    @Value("${spring.messages.basename}")
    private String messages;

    @Autowired
    private AddressRepo addressRepo;

    @Autowired
    private CityService cityService;

    /**
     * adding an address to database
     * add given address if it is absent, otherwise assign an id to argument
without adding
     * @param address address to write in database
     * @return true if address was added, false if address already was added
     * @throws NoExistingCityException if present city of address is absent in
database
     * */
    public boolean addAddress(Address address) throws NoExistingCityException {
        if (Objects.isNull(address)){
            return false;
        }

        City city = address.getCity();
        if(Objects.isNull(city) || Objects.isNull(city.getId()) ||
Objects.isNull(cityService.findCityById(city.getId()))){
            throw new NoExistingCityException(Optional.ofNullable(city).orElse(new
City()), ResourceBundle.getBundle(messages));
        }

        Address foundAddress;
        if (!Objects.isNull(foundAddress =
addressRepo.findByHouseNumberAndCityAndStreet(address.getHouseNumber(), city,
address.getStreet()))){
             address.setId(foundAddress.getId());
             return false;
        }
        addressRepo.save(address);
        return true;
    }

    public void deleteAddress(Address address){
        addressRepo.delete(address);
    }
```

```
    /**
     * takes an Address object from database
     * @param id of Address
     * @return Address object
     * @throws java.util.NoSuchElementException if object is absent in database
     * */
    public Address findAddressById(Long id){
        return addressRepo.findById(id).orElseThrow();
    }

    public Address findByCityAndStreetAndHouseNumber(City city, String street,
String houseNumber){
        return addressRepo.findByHouseNumberAndCityAndStreet(houseNumber, city,
street);
    }
}
public class CityService {

    @Value("${spring.messages.basename}")
    private String messages;

    @Autowired
    private CityRepo cityRepo;

    /**
     * add City object to database if city with such zipcode is absent
     * @param city city for adding
     * @return true if object was successfully added, otherwise false
     * */
    public boolean addCity(City city){
        if (cityRepo.findByZipcode(city.getZipcode()).isPresent()){
            return false;
        }
        cityRepo.save(city);

        return true;
    }
    /**
     * find city in database according to the given id
     * @param id city identifier in database
     * @return found City object or null if absent
     * */
    public City findCityById(Long id){
        return cityRepo.findById(id).orElse(null);
    }

    /**
     * find city in database according given zipcode
     * @param zipcode post code of City object
     * @return found City object or null if city with given zipcode is absent
     * */
    public City findCityByZipCode(String zipcode) {
        return cityRepo.findByZipcode(zipcode).orElse(null);
    }

    /**
     * localize given city according to given bundle
     * @param city object to localize
     * @param locale locale for localization
     * @return new object localized by needed locale
```

```
     * */
    @SneakyThrows
    public City localize(City city, Locale locale){
       return localize(city, ResourceBundle.getBundle(messages, locale));
    }

    /**
     * localize given city according to given bundle
     * @param city object to localize
     * @param bundle mean for localization
     * @return new object localized by needed locale
     * */
    @SneakyThrows
    public City localize(City city, ResourceBundle bundle){
        City localizedCity = (City) city.clone();
        localizedCity.setName(bundle.getString("city."+city.getName()));
        return localizedCity;
    }

    public List<City> findAll(Locale locale) {
        ResourceBundle bundle = ResourceBundle.getBundle(messages, locale);
        return findAll().stream().map(o -> localize(o,
bundle)).collect(Collectors.toList());
    }

    public List<City> findAll(){
        return cityRepo.findAll();
    }

    public List<City> findAll(Locale locale, Sort sort){
        List<City> cities = findAll(locale);
        sortCities(cities, sort, locale);
        return cities;
    }

    //TODO take out logic into parametrized method of some utility class
    private void sortCities(List<City> cities, Sort sort, Locale locale){
        Collator collator = Collator.getInstance(locale);

        List<Sort.Order> orders = sort.get().collect(Collectors.toList());

        Comparator<City> comparator = null;

        ComparatorRecognizer recognizer = new
CityService.ComparatorRecognizer(collator);
        for (Sort.Order order:orders) {
            if (Objects.isNull(comparator)){
                comparator = recognizer.getComparator(order);
            }
            else{
                comparator =
comparator.thenComparing(recognizer.getComparator(order));
            }
        }

        if (!Objects.isNull(comparator)) {
            cities.sort(comparator);
        }
    }

    //TODO take out class into some utility class, make ComparatorRecognizer
```

```
interface
    private static class ComparatorRecognizer {
        private final Map<String, Comparator<City>> comparators;

        ComparatorRecognizer(Collator collator){
            comparators = new HashMap<>();
            comparators.put("name", new City.NameComparator(collator));
        }

        private Comparator<City> getComparator(Sort.Order order){
            Comparator<City> cmp = comparators.get(order.getProperty());
            if (!Objects.isNull(cmp) && order.isDescending()){
                cmp = cmp.reversed();
            }
            return cmp;
        }
    }
}
public class CityUtils {

    /**
     * Calculate smallest distance between cityFrom and cityTo.
     * @param cityFrom localized source city
     * @param cityTo localized destination city
     * @param directions list of all available directions between cities
     * @return number which represents smallest distance between cityFrom and
cityTo
     * if no way exists return Double.POSITIVE_INFINITY
     * */
    public static Double getMinDistance(City cityFrom, City cityTo,
List<DirectionDelivery> directions){

        Set<City> cities =
directions.stream().collect(getCityFromDirectionsCollector());
        Map<City, Integer> cityIntegerMap = new HashMap<>();

        int index = 0;
        for (City city:cities) {
            cityIntegerMap.put(city, index);
            ++index;
        }

        double[][] graph = createGraph(directions, cityIntegerMap);

        Integer source = cityIntegerMap.get(cityFrom);
        Integer dest = cityIntegerMap.get(cityTo);

        Dijkstra dijkstra = new Dijkstra(graph, source);
        return calculateMinDistance(dest, dijkstra);
    }

    /**
     * Calculate smallest distance between cityFrom and cityTo.
     * @param cityFrom localized source city
     * @param cityTo localized destination city
     * @param directions list of all available directions between cities
     * @return City.Distance object with source city, destination city, smallest
distance (rounded number to 1 decimal place)
     * and route if route doesn't exist return null
     * */
    public static City.Distance getDistance(City cityFrom, City cityTo,
```

```
List<DirectionDelivery> directions){

        Set<City> cities =
directions.stream().collect(getCityFromDirectionsCollector());

        Map<City, Integer> cityIntegerMap = new HashMap<>();
        Map<Integer, City> integerCityMap = new HashMap<>();

        int index = 0;
        for (City city:cities) {
            cityIntegerMap.put(city, index);
            integerCityMap.put(index, city);
            ++index;
        }

        double[][] graph = createGraph(directions, cityIntegerMap);

        Integer source = cityIntegerMap.get(cityFrom);
        Integer dest = cityIntegerMap.get(cityTo);

        Dijkstra dijkstra = new Dijkstra(graph, source);
        double minDistance = calculateMinDistance(dest, dijkstra);

        if (minDistance != Double.POSITIVE_INFINITY){
            List<City> route =
buildCitiesSmallestRout(dijkstra.buildShortestRoute(dest), integerCityMap);
            return new City.Distance(cityFrom, cityTo, minDistance, route);
        }

        return null;
    }

    private static double calculateMinDistance(Integer dest, Dijkstra dijkstra) {
        return Math.round(dijkstra.calculateMinDistance(dest) * 10.0) / 10.0;
    }

    private static double[][] createGraph(List<DirectionDelivery> directions,
Map<City, Integer> cityIntegerMap) {
        int size = cityIntegerMap.size();
        double[][] graph = new double[size][size];

        for (int i = 0; i < graph.length; i++) {
            Arrays.fill(graph[i], Double.POSITIVE_INFINITY);
        }
        for (DirectionDelivery d : directions) {
            int i = cityIntegerMap.get(d.getSenderCity());
            int j = cityIntegerMap.get(d.getReceiverCity());
            graph[i][j] = graph[j][i] = d.getDistance();
        }
        return graph;
    }

    private static List<City> buildCitiesSmallestRout(List<Integer> routAsInt,
Map<Integer, City> integerCityMap){
        return
routAsInt.stream().map(integerCityMap::get).collect(Collectors.toList());
    }

    private static Collector<DirectionDelivery, HashSet<City>, HashSet<City>>
getCityFromDirectionsCollector() {
        return Collector.of(
```

```
                (Supplier<HashSet<City>>) HashSet::new,
                (set, d) -> {
                    set.add(d.getSenderCity());
                    set.add(d.getReceiverCity());
                }, (cities1, cities2) -> {
                    cities1.addAll(cities2);
                    return cities1;
                }
        );
    }

}
public class DeliveredBaggageService {

    private final DeliveredBaggageRepo deliveredBaggageRepo;

    public DeliveredBaggageService(DeliveredBaggageRepo deliveredBaggageRepo) {
        this.deliveredBaggageRepo = deliveredBaggageRepo;
    }

    public Optional<DeliveredBaggage> findById(Long id){
        return deliveredBaggageRepo.findById(id);
    }

    /**
     * Save delivered baggage in the database
     * @param deliveredBaggage baggage to add
     * @throws IllegalArgumentException whereas any of mandatory attributes are
missing
     * @return true whether deliveredBaggage was saved successful, false if param
is null
     * */
    public boolean save(DeliveredBaggage deliveredBaggage) {
        if (Objects.isNull(deliveredBaggage)){
            return false;
        }
        requireValidBaggage(deliveredBaggage);
        deliveredBaggageRepo.save(deliveredBaggage);
        return true;
    }

    private void requireValidBaggage(DeliveredBaggage deliveredBaggage) {

Optional.ofNullable(deliveredBaggage.getWeight()).orElseThrow(IllegalArgumentExcep
tion::new);

Optional.ofNullable(deliveredBaggage.getVolume()).orElseThrow(IllegalArgumentExcep
tion::new);

Optional.ofNullable(deliveredBaggage.getType()).orElseThrow(IllegalArgumentExcepti
on::new);
    }

    /**
     * Updates state of already existing baggage
     * @param deliveredBaggage updating baggage object
     * @param deliveredBaggageRequest updating data
     * @return updated DeliveredBaggage object
     * */
    public DeliveredBaggage update(DeliveredBaggage deliveredBaggage,
DeliveredBaggageRequest deliveredBaggageRequest) {
```

```
        Objects.requireNonNull(deliveredBaggage, "Delivered Baggage from db cannot
be null");
        Objects.requireNonNull(deliveredBaggageRequest, "Delivered Baggage Request
cannot be null");
        if (deliveredBaggageRepo.findById(deliveredBaggage.getId()).isEmpty()){
            throw new IllegalArgumentException("Param deliveredBaggage doesn't
exist in db");
        }
        DeliveredBaggage updated =
ServiceUtils.createDeliveredBaggage(deliveredBaggageRequest);
        updated.setId(deliveredBaggage.getId());
        return deliveredBaggageRepo.save(updated);
    }

    /**
     * Gives report of profit {@link DeliveredBaggage} per each {@link
BaggageType}.
     * @return {@link List} of {@link IProfitBaggagePerType} objects
     * */
    public List<IProfitBaggagePerType> profitBaggageReport(){
        return deliveredBaggageRepo.profitBaggagePerType();
    }

}
public class DeliveryApplicationService {

    private static final Logger logger =
Logger.getLogger(DeliveryApplicationService.class);

    @Autowired
    private DeliveryApplicationRepo deliveryApplicationRepo;

    @Autowired
    private CityService cityService;

    @Autowired
    private DeliveredBaggageService deliveredBaggageService;

    @Autowired
    private UserService userService;

    @Autowired
    private AddressService addressService;

    @Autowired
    private DeliveryCostCalculatorService costCalculatorService;

    @Autowired
    private DeliveryReceiptService receiptService;

    @Value("${spring.messages.basename}")
    private String messages;

    /**
     * Make and send delivery application to the database. Calculate price before
saving
     * @param customer initiator of sending application
     * @param request data to make DeliveryApplication object
     * @return returned value of saveApplication method
     * */
    public boolean sendApplication(User customer, DeliveryApplicationRequest
```

```
request) throws WrongDataException {
        Objects.requireNonNull(customer, "Customer object cannot be null");
        Objects.requireNonNull(request, "DeliveryApplicationRequest object cannot
be null");

        ResourceBundle bundle = ResourceBundle.getBundle(messages,
LocaleContextHolder.getLocale());

        DeliveryApplication application =
ServiceUtils.createDeliveryApplication(customer, request, cityService, bundle);
        application.setPrice(calculatePrice(application));

        return saveApplication(application);

    }

    public Double calculatePrice(DeliveryApplication application) throws
NoExistingDirectionException {
        Double distanceCost =
costCalculatorService.calculateDistanceCost(application.getSenderAddress(),
application.getReceiverAddress());
        DeliveredBaggage deliveredBaggage = application.getDeliveredBaggage();
        Double weightCost =
costCalculatorService.calculateWeightCost(deliveredBaggage.getWeight());
        Double dimensionsCost =
costCalculatorService.calculateDimensionsCost(deliveredBaggage.getVolume());
        return distanceCost + weightCost + dimensionsCost;
    }
    /**
     * Save application to the database. Doesn't automatically calculate price,
takes already assigned one
     * @param application deliveryApplication to save
     * @throws NoExistingCityException any from specified addresses contains
missing in the database city
     * @throws IllegalArgumentException if user doesn't exist in the database
     * @return true if application was saved successfully, false if parameter
application is null
     * */
    public boolean saveApplication(DeliveryApplication application) throws
NoExistingCityException {
        if (Objects.isNull(application)){
            return false;
        }
        ServiceUtils.requireExistingUser(application.getCustomer(), userService);


Optional.ofNullable(application.getSendingDate()).orElseThrow(IllegalArgumentExcep
tion::new);

Optional.ofNullable(application.getReceivingDate()).orElseThrow(IllegalArgumentExc
eption::new);


deliveredBaggageService.save(requireNotNullDeliveredBaggage(application.getDeliver
edBaggage()));

addressService.addAddress(requireNotNullAddress(application.getSenderAddress()));

addressService.addAddress(requireNotNullAddress(application.getReceiverAddress()))
;
        deliveryApplicationRepo.save(application);
```

```
        logger.info(String.format("Delivery application: %s has been successfully
made", applicationLogInfo(application)));
        return true;
    }

    private DeliveredBaggage requireNotNullDeliveredBaggage(DeliveredBaggage
baggage) {
        return
Optional.ofNullable(baggage).orElseThrow(IllegalArgumentException::new);
    }

    private Address requireNotNullAddress(Address application) {
        return Optional.ofNullable(application).orElseThrow(()->new
IllegalArgumentException("Address in application cannot be null!"));
    }

    /**
     * Finds application according to the given id
     * @param id unique identifier of application in the database
     * @return found DeliveryApplication object, if no objects are found returns
null
     * */
   public DeliveryApplication findById(Long id){
        return deliveryApplicationRepo.findById(id).orElse(null);
   }

    public List<DeliveryApplication> findAll() {
        return deliveryApplicationRepo.findAll();
    }

    public Page<DeliveryApplication> findAllByUserId(Long id, Pageable pageable) {
        return deliveryApplicationRepo.findAllByUserId(id, pageable);
    }

    public List<DeliveryApplication> findAllByUserId(Long id) {
        return deliveryApplicationRepo.findAllByUserId(id);
    }

    public List<DeliveryApplication>
findAll(DeliveryApplicationsReviewFilterRequest filter){
        List<DeliveryApplication> applications = findAll();
        applications = applications.stream()
                .filter(
                        statePredicate(filter)
                                .and(baggageTypePredicate(filter))
                                .and(senderCityPredicate(filter))
                                .and(receiverCityPredicate(filter))
                                .and(sendingDatePredicate(filter))
                                .and(receivingDatePredicate(filter))
                )
                .collect(Collectors.toList());

        return applications;
    }

    private Predicate<? super DeliveryApplication>
receivingDatePredicate(DeliveryApplicationsReviewFilterRequest filter) {
        return a -> Objects.isNull(filter.getReceivingDate()) ||
a.getReceivingDate().equals(filter.getReceivingDate());
    }
```

```
    private Predicate<? super DeliveryApplication>
sendingDatePredicate(DeliveryApplicationsReviewFilterRequest filter) {
        return a -> Objects.isNull(filter.getSendingDate()) ||
a.getSendingDate().equals(filter.getSendingDate());
    }

    private Predicate<? super DeliveryApplication>
senderCityPredicate(DeliveryApplicationsReviewFilterRequest filter) {
        return a -> Objects.isNull(filter.getCityFromId()) ||
a.getSenderAddress().getCity().getId().equals(filter.getCityFromId());
    }

    private Predicate<? super DeliveryApplication>
receiverCityPredicate(DeliveryApplicationsReviewFilterRequest filter) {
        return a -> Objects.isNull(filter.getCityToId()) ||
a.getReceiverAddress().getCity().getId().equals(filter.getCityToId());
    }

    private Predicate<DeliveryApplication>
baggageTypePredicate(DeliveryApplicationsReviewFilterRequest filter) {
        return a -> Objects.isNull(filter.getBaggageType()) ||
a.getDeliveredBaggage().getType().equals(filter.getBaggageType());
    }


    private Predicate<DeliveryApplication>
statePredicate(DeliveryApplicationsReviewFilterRequest filter) {
        return a -> Objects.isNull(filter.getApplicationState()) ||
a.getState().equals(filter.getApplicationState());
    }

    public Page<DeliveryApplication>
getPage(DeliveryApplicationsReviewFilterRequest applicationsRequest, Pageable
pageable) {
        List<DeliveryApplication> list = findAll(applicationsRequest);
        return ServiceUtils.toPage(list, pageable, new
DeliveryApplicationComparatorRecognizer());
    }

    public void rejectApplication(DeliveryApplication application) {
       Objects.requireNonNull(application, "Application cannot be null");
       Optional<DeliveryReceipt> receiptOptional =
receiptService.findByApplicationId(application.getId());
       receiptOptional.ifPresent(r ->{
           if (r.getPaid()){
               throw new IllegalStateException("Cannot reject already paid
application");
           }
           else{
               receiptService.deleteById(r.getId());
           }
       });
       application.setState(DeliveryApplication.State.REJECTED);
       deliveryApplicationRepo.save(application);
       logger.info( String.format("Reject application: %s",
applicationLogInfo(application)));
    }

    public DeliveryApplication edit(DeliveryApplication application,
UpdateDeliveryApplicationRequest updated) throws NoExistingCityException,
NoExistingDirectionException {
```

```
        Objects.requireNonNull(application, "Application cannot be null");
        Objects.requireNonNull(updated, "UpdatedRequest cannot be null");

        if (!application.getState().equals(DeliveryApplication.State.SUBMITTED)){
            throw new IllegalStateException("Forbidden edit approved
applications");
        }

        if (!Objects.isNull(updated.getDeliveredBaggageRequest())){

application.setDeliveredBaggage(deliveredBaggageService.update(application.getDeli
veredBaggage(), updated.getDeliveredBaggageRequest()));
        }

        if (!Objects.isNull(updated.getSenderAddress())){
            Address updatedSenderAddress =
ServiceUtils.createAddress(updated.getSenderAddress(), cityService);
            addressService.addAddress(updatedSenderAddress);
            application.setSenderAddress(updatedSenderAddress);
        }

        if (!Objects.isNull(updated.getReceiverAddress())){
            Address updatedReceiverAddress =
ServiceUtils.createAddress(updated.getReceiverAddress(), cityService);
            addressService.addAddress(updatedReceiverAddress);
            application.setReceiverAddress(updatedReceiverAddress);
        }

        if (!Objects.isNull(updated.getSendingDate())){
            application.setSendingDate(updated.getSendingDate());
        }

        if (!Objects.isNull(updated.getReceivingDate())){
            application.setReceivingDate(updated.getReceivingDate());
        }

        application.setPrice(calculatePrice(application));

        deliveryApplicationRepo.save(application);
        return application;
    }

    /**
     * change application state to completed
     * @param application customer's delivery application
     * @throws IllegalStateException if receipt is not paid or not found
     * */
    public void completeApplication(DeliveryApplication application) {
        Objects.requireNonNull(application, "Application cannot be null");
        Optional<DeliveryReceipt> receiptOptional =
receiptService.findByApplicationId(application.getId());
        DeliveryReceipt receipt = receiptOptional.orElseThrow(()->new
IllegalStateException("Cannot complete application. No paid receipt found!"));

        if (!receipt.getPaid()){
            throw new IllegalStateException("Cannot complete application. No paid
receipt found!");
        }

        if (application.getReceivingDate().isAfter(LocalDate.now())){
            throw new IllegalStateException("Cannot complete application.
```

```
Receiving date hasn't come yet.");
        }

        application.setState(DeliveryApplication.State.COMPLETED);
        deliveryApplicationRepo.save(application);
    }

    private static class DeliveryApplicationComparatorRecognizer implements
ServiceUtils.ComparatorRecognizer<DeliveryApplication> {

        private final Map<String, Comparator<DeliveryApplication>> comparators;

        DeliveryApplicationComparatorRecognizer(){
            comparators = new HashMap<>();
            comparators.put("id", Comparator.comparing(DeliveryApplication::getId,
Long::compareTo));
        }

        @Override
        public Comparator<DeliveryApplication> getComparator(Sort.Order order) {
            Comparator<DeliveryApplication> cmp =
comparators.get(order.getProperty());
            if (!Objects.isNull(cmp) && order.isDescending()){
                cmp = cmp.reversed();
            }
            return cmp;
        }
    }

    /**
     * Make String for logger consists with applications info
     * @param application
     * @return String in format [id=(id), user=(userFullName), price=(price)]
     * */
    private static String applicationLogInfo(DeliveryApplication application){
        User customer = application.getCustomer();
        String customerFullName = String.format("%s %s", customer.getName(),
customer.getSurname());
        return String.format("[id=%1$d, user=%2$s, price=%3$f]",
application.getId(), customerFullName, application.getPrice());
    }

}


public class DeliveryCostCalculatorService {
    @Autowired
    private DistanceFareService distanceFareService;

    @Autowired
    private WeightFareService weightFareService;

    @Autowired
    private DimensionsFareService dimensionsFareService;

    @Autowired
    private DirectionDeliveryService directionDeliveryService;

    @Autowired
    private CityService cityService;
```

```
    @Value("${spring.messages.basename}")
    private String messages;

    public DeliveryCostCalculatorResponse
calculateCost(DeliveryCostCalculatorRequest calculatorRequest, Locale locale)
throws WrongDataException {
        Objects.requireNonNull(calculatorRequest, "calculatorRequest cannot be
null");

        ResourceBundle bundle = ResourceBundle.getBundle(messages, locale);

        Long cityFromId = calculatorRequest.getCityFromId();
        Long cityToId = calculatorRequest.getCityToId();
        requireDifferentCities(cityFromId, cityToId, bundle);

        City cityFrom = requireExistingCity(cityFromId);
        City cityTo = requireExistingCity(cityToId);

        City.Distance distance =
directionDeliveryService.getDistanceBetweenCities(cityFrom, cityTo, locale);

        double totalCost = calculate(distance.getDistance(),
calculatorRequest.getWeight(), calculatorRequest.getDimensions().getVolume());
        return new DeliveryCostCalculatorResponse(totalCost, distance);
    }

    public Double calculateCost(DeliveredBaggage baggage, Address sender, Address
receiver) throws NoExistingCityException {

        City from = requireExistingCity(sender.getCity().getId());
        City to = requireExistingCity(receiver.getCity().getId());

        Double distance = directionDeliveryService.calculateMinDistance(from, to);
        return calculate(distance, baggage.getWeight(), baggage.getVolume());
    }

    public Double calculateWeightCost(Double weight){
        return weightFareService.getPrice(weight.intValue());
    }

    public Double calculateDimensionsCost(Double volume){
        return dimensionsFareService.getPrice(volume.intValue());
    }

    /**
     * calculate distance cost between two addresses
     * @param sender address of sender
     * @param receiver address of receiver
     * @return cost according to fare, if direction is within one city, return 0.0
     * @throws NoExistingDirectionException if direction between addresses doesn't
exist in the database
     * */
    public Double calculateDistanceCost(Address sender, Address receiver) throws
NoExistingDirectionException {
        City from = sender.getCity();
        City to = receiver.getCity();
        if (!Objects.equals(to.getId(), from.getId())){
            Double minDistance =
directionDeliveryService.calculateMinDistance(from, to);
            if (minDistance.equals(Double.POSITIVE_INFINITY)){
                ResourceBundle bundle = ResourceBundle.getBundle(messages,
```

```
Locale.UK);
                    throw new NoExistingDirectionException(from, to, bundle);
                }
                return distanceFareService.getPrice(minDistance.intValue());
            }
            return 0.0;
        }

        private void requireDifferentCities(Long cityFromId, Long cityToId,
ResourceBundle bundle) throws WrongDataException {
            if (Objects.equals(cityFromId, cityToId)){
                throw new WrongDataAttributeException(CITY_DIRECTION.getAttr(),
bundle, INVALID_DIRECTION_SAME_CITIES);
            }
        }

        private City requireExistingCity(Long cityId) throws NoExistingCityException {
            City city = cityService.findCityById(cityId);
            if (Objects.isNull(city)){
                throw new NoExistingCityException();
            }
            return city;
        }


        private Double calculate(Double distance, Double weight, Double volume){
            double distanceCost = distanceFareService.getPrice(distance.intValue());
            double weightCost = weightFareService.getPrice(weight.intValue());
            double dimensionsCost = dimensionsFareService.getPrice(volume.intValue());
            return distanceCost + weightCost + dimensionsCost;
        }
}
```

Додаток Г – Ілюстративна частина

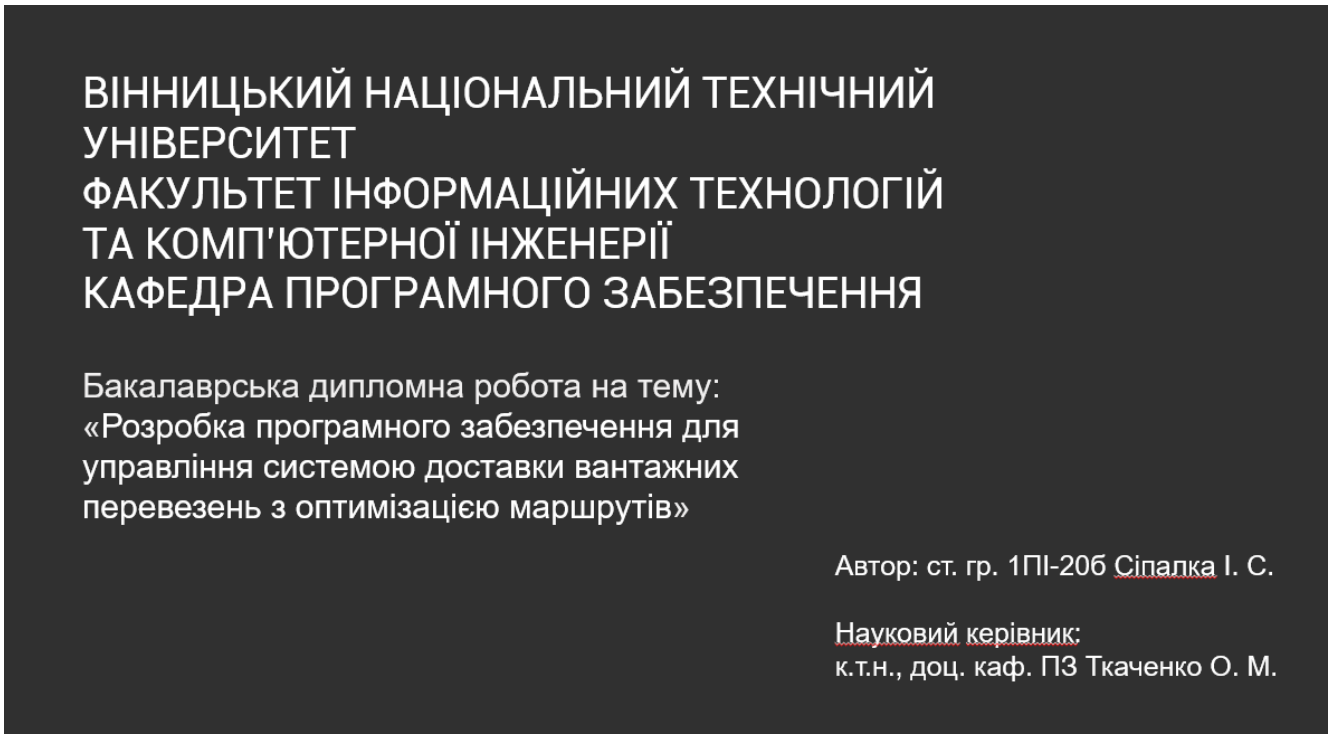

Рисунок Г.1 – Титульний слайд

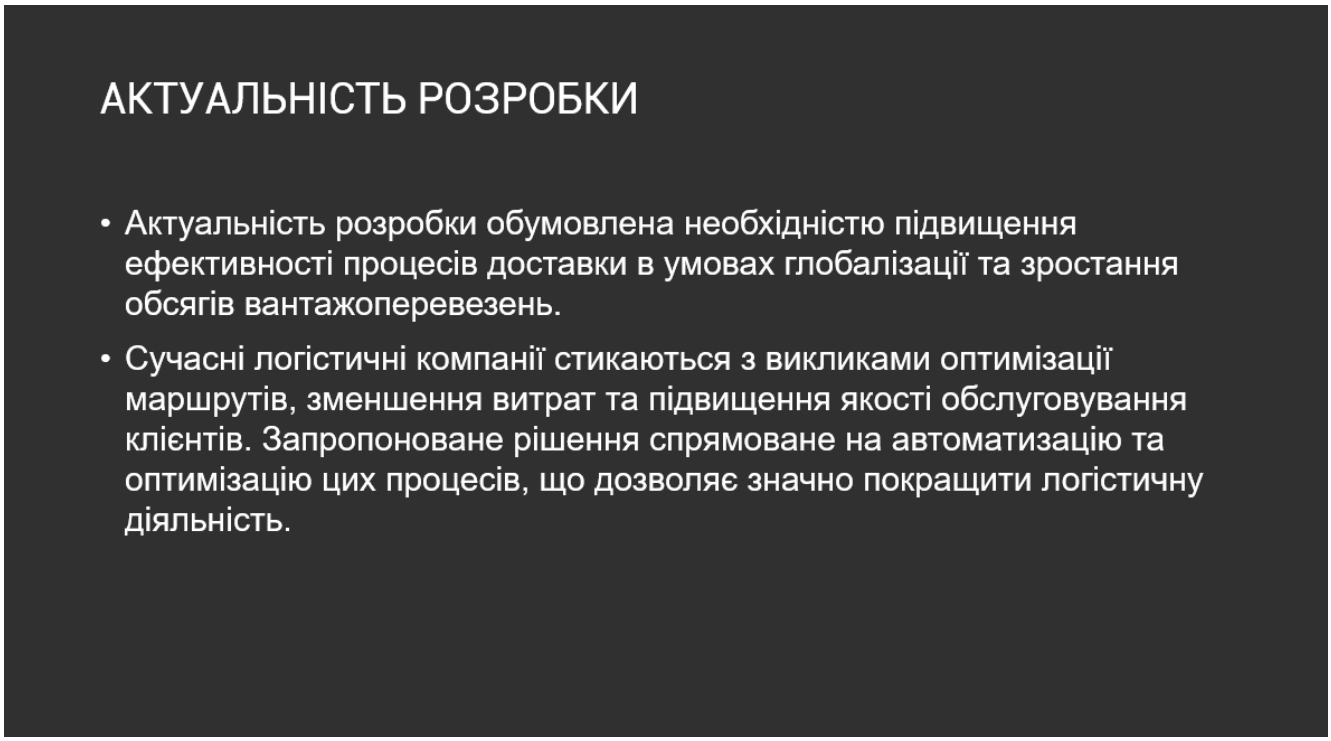

Рисунок Г.2 – Актуальність теми

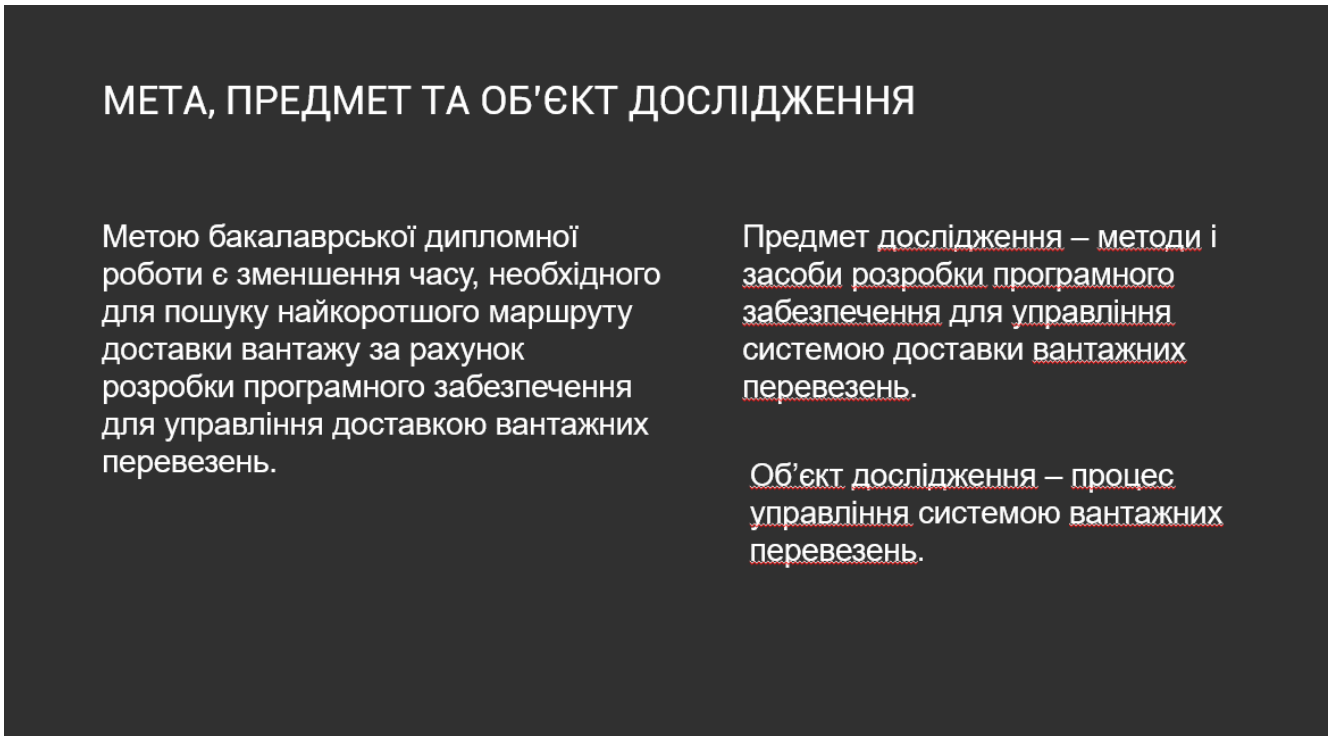

Рисунок Г.3 – Мета, об’єкт та предмет дослідження

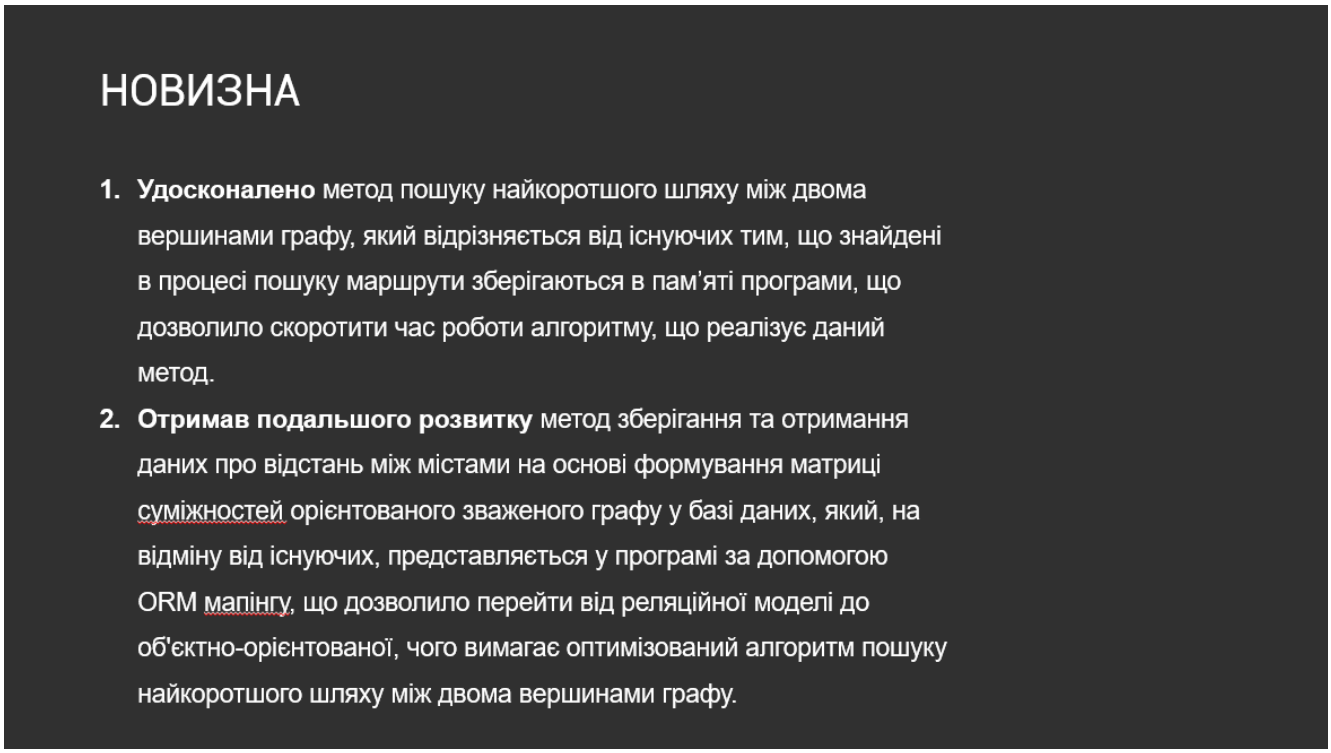

Рисунок Г.4 – Наукова новизна

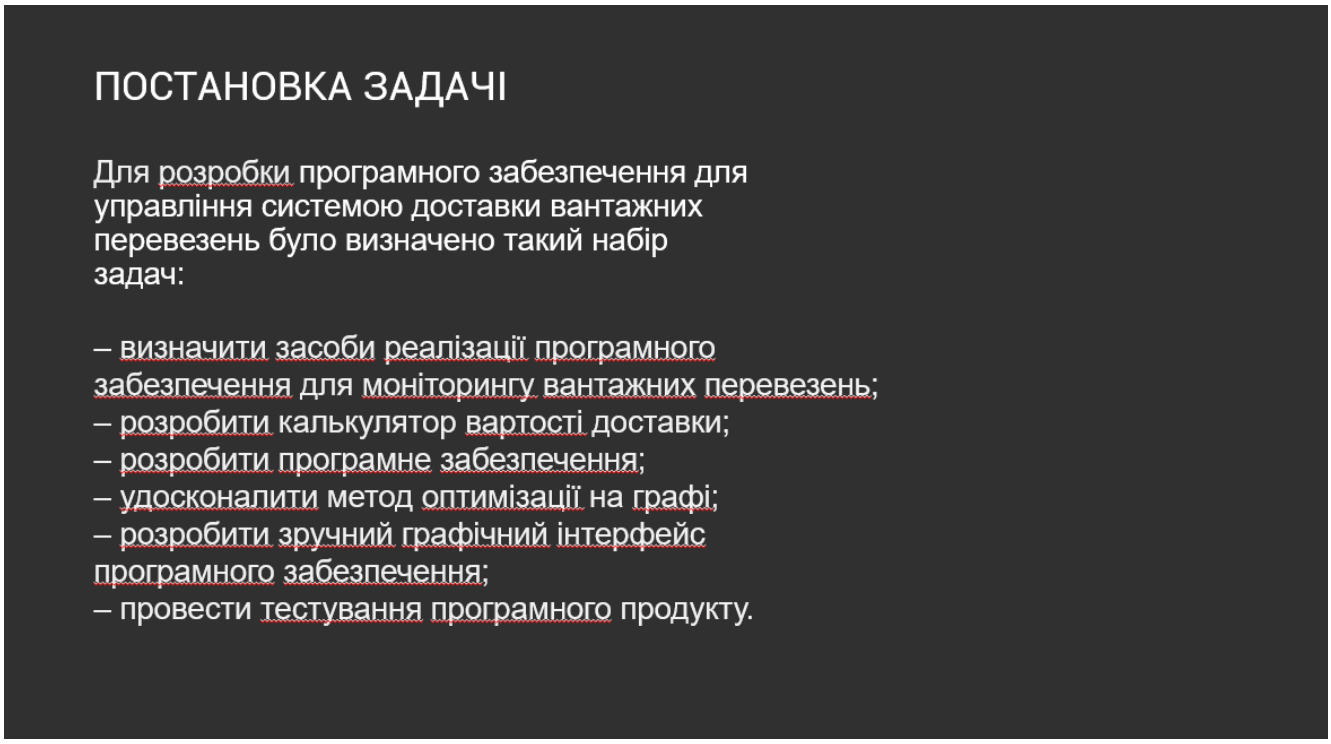

Рисунок Г.5 – Постановка задачі

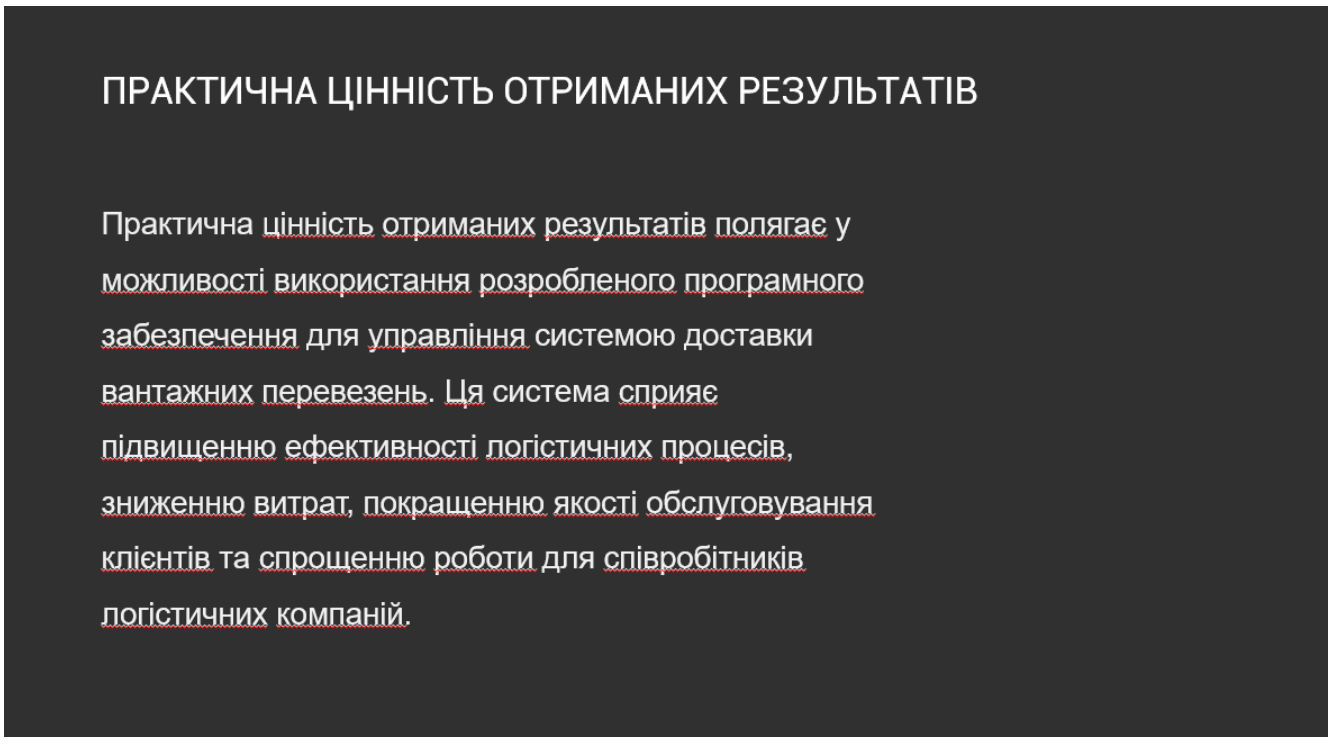

Рисунок Г.6 – Практична цінність отриманих результатів

АНАЛІЗ АНАЛОГІВ

	Власна реалізація	SAP TM	TMS	Descartes
Веб сайт	1	0	1	0
Зручний інтерфейс	1	1	0	0
Калькулятор вартості	1	1	0	1
Оптимізація	1	0	1	0
Загальна оцінка	100%	50%	50%	25%

Рисунок Г.7 – Аналіз аналогів

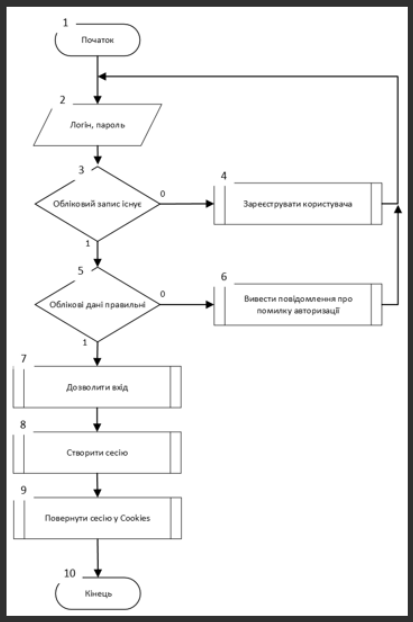

СХЕМА РОБОТИ АЛГОРИТМУ РЕЄСТРАЦІЇ ТА АВТОРИЗАЦІЇ

Рисунок Г.8 – Схема роботи алгоритму реєстрації та авторизації

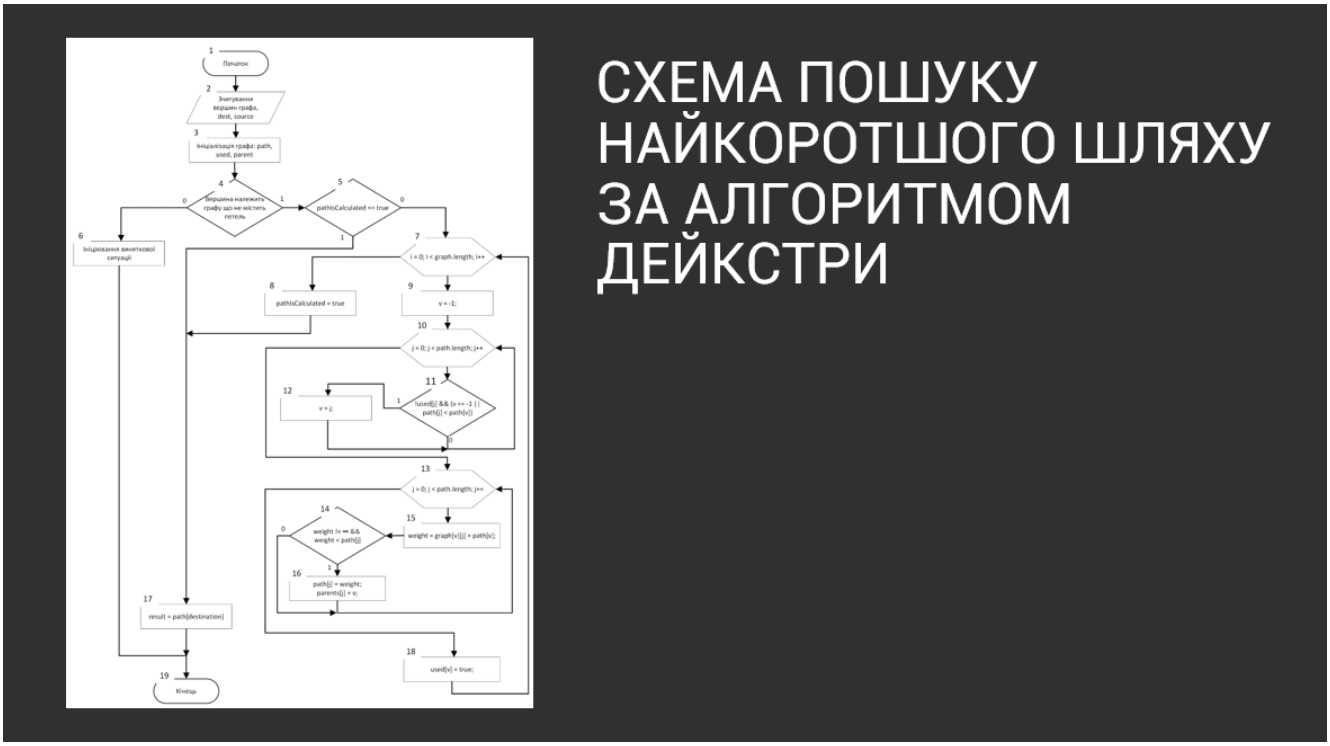

Рисунок Г.9 – Схема пошуку найкоротшого шляху за алгоритмом Дейкстри

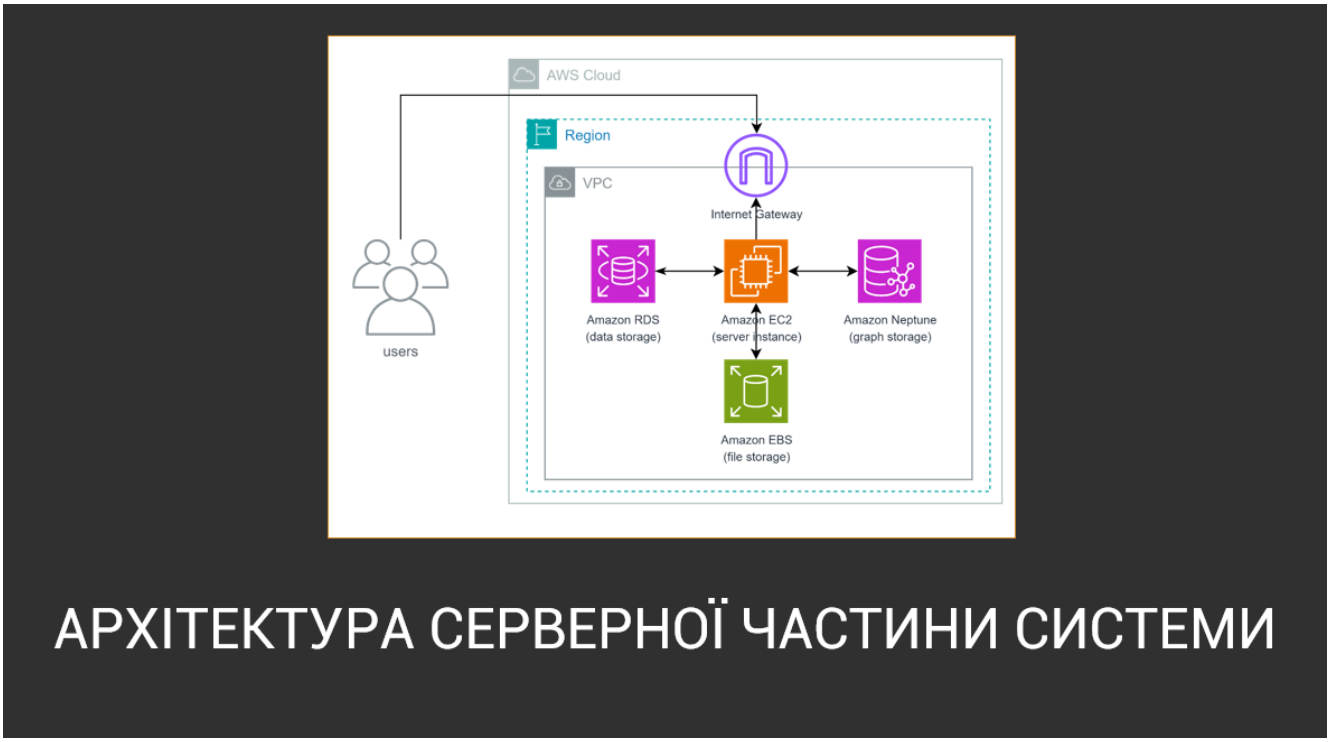

Рисунок Г.10 – Архітектура серверної частини системи

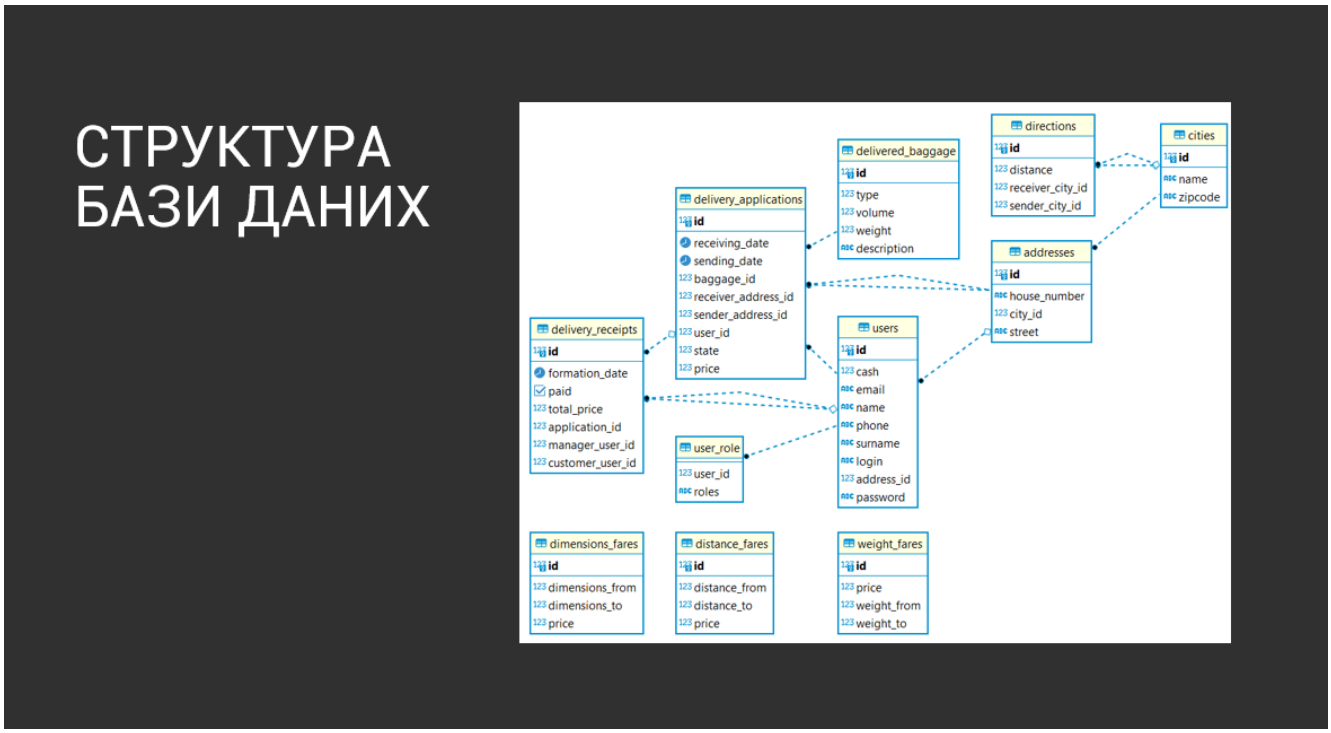

Рисунок Г.11 – Структура бази даних

Рисунок Г.12 – Ілюстрація зваженого графу

Рисунок Г.13 – Використані технології

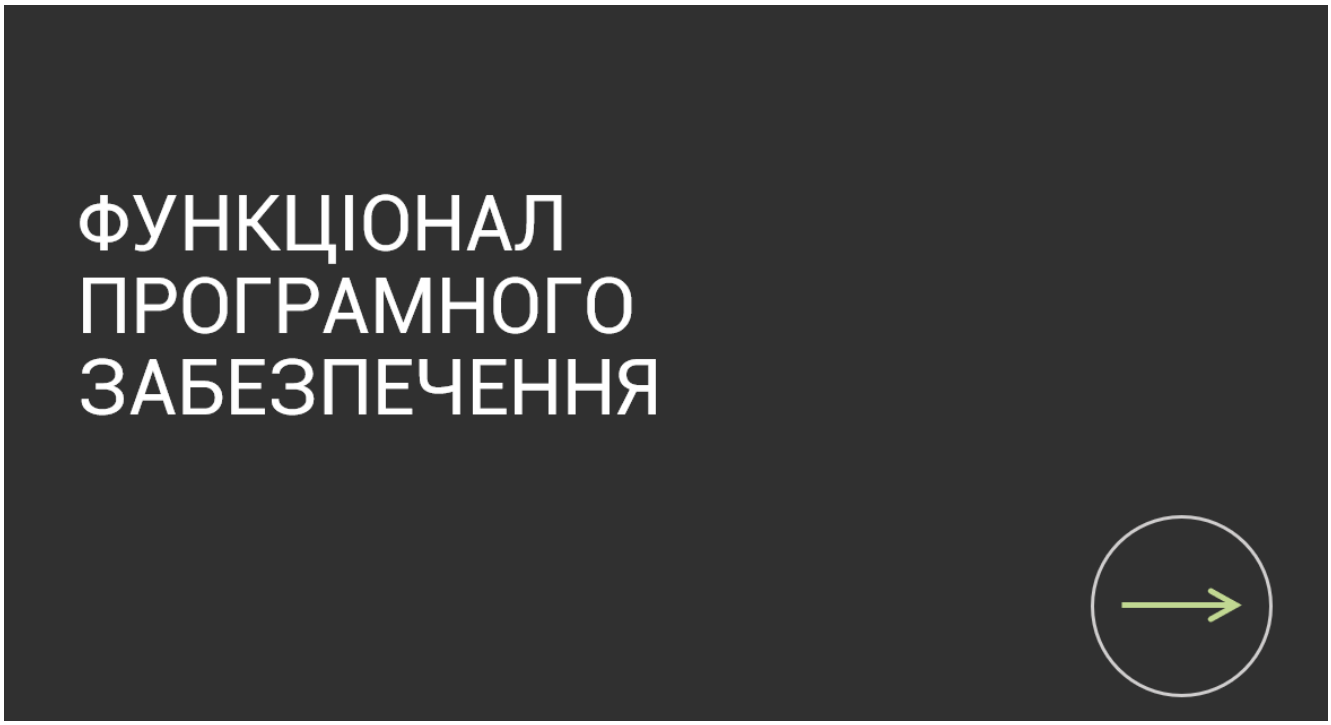

Рисунок Г.14 – Функціонал програмного забезпечення

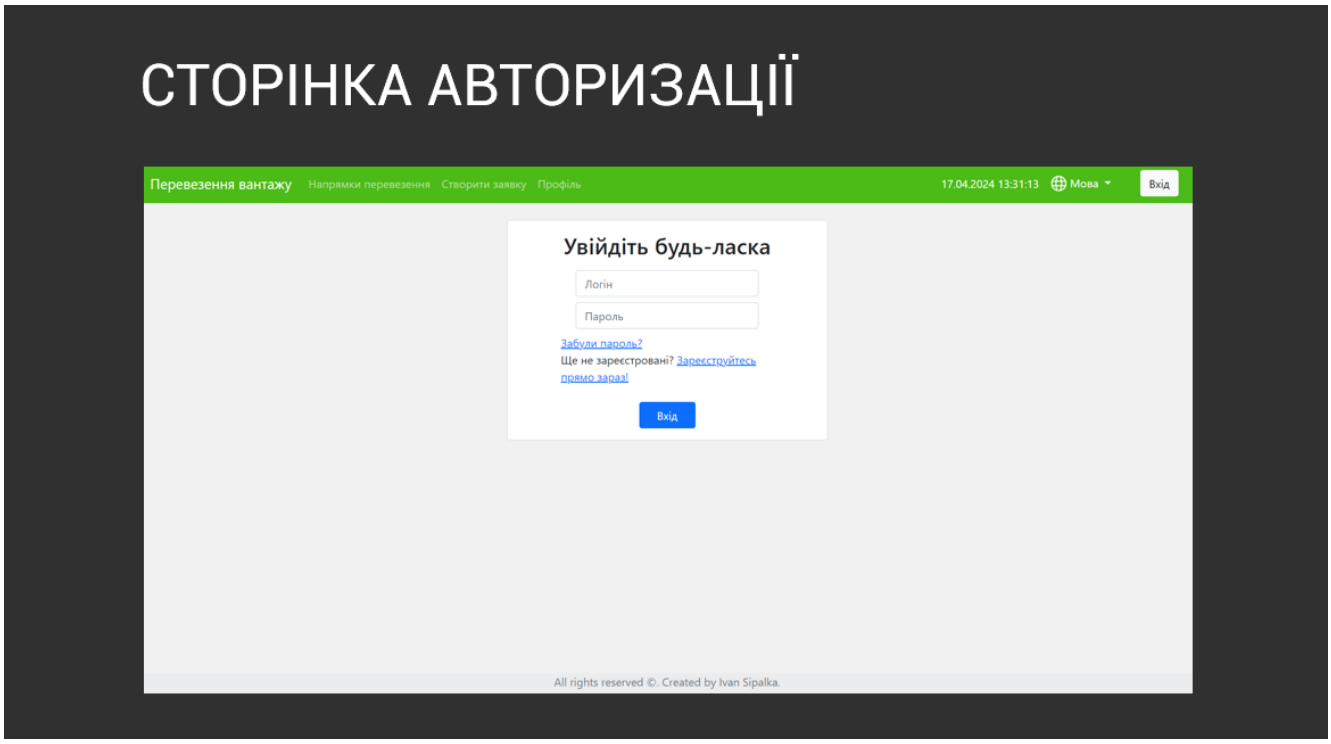

Рисунок Г.15 – Сторінка авторизації

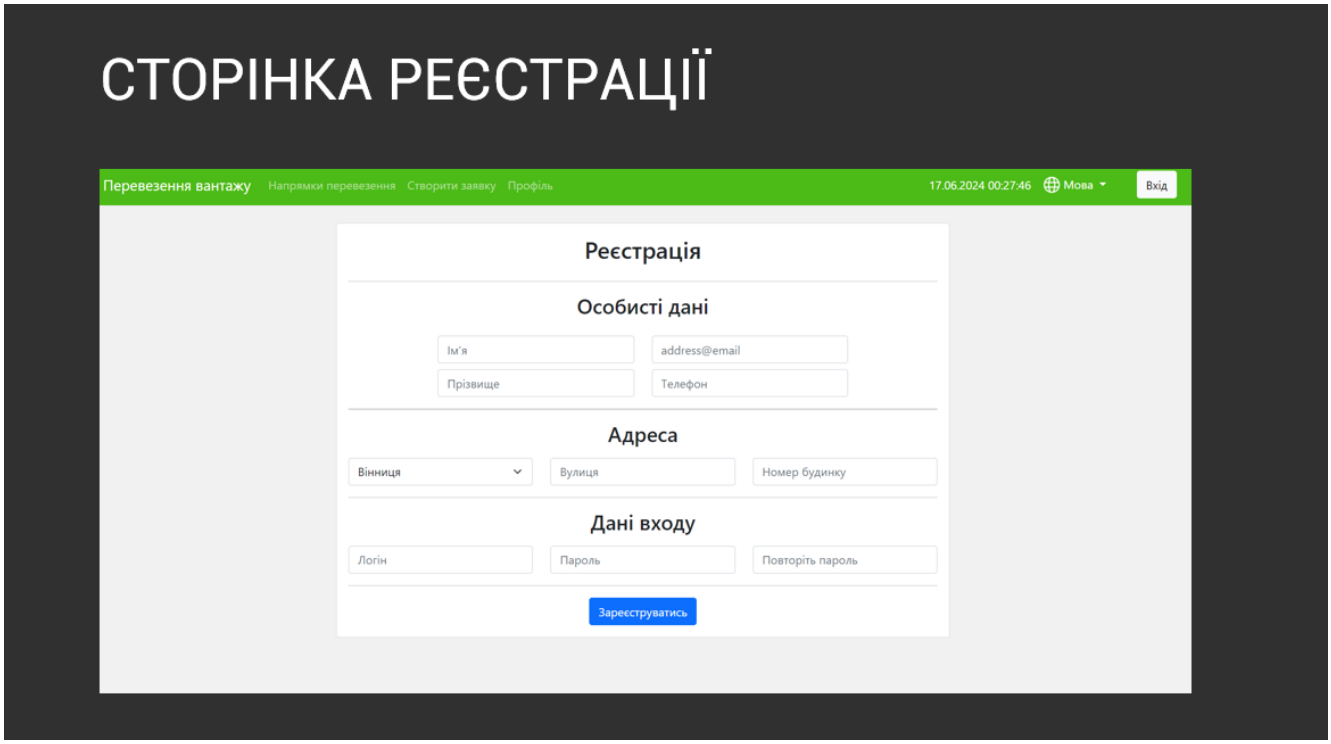

Рисунок Г.16 – Сторінка реєстрації

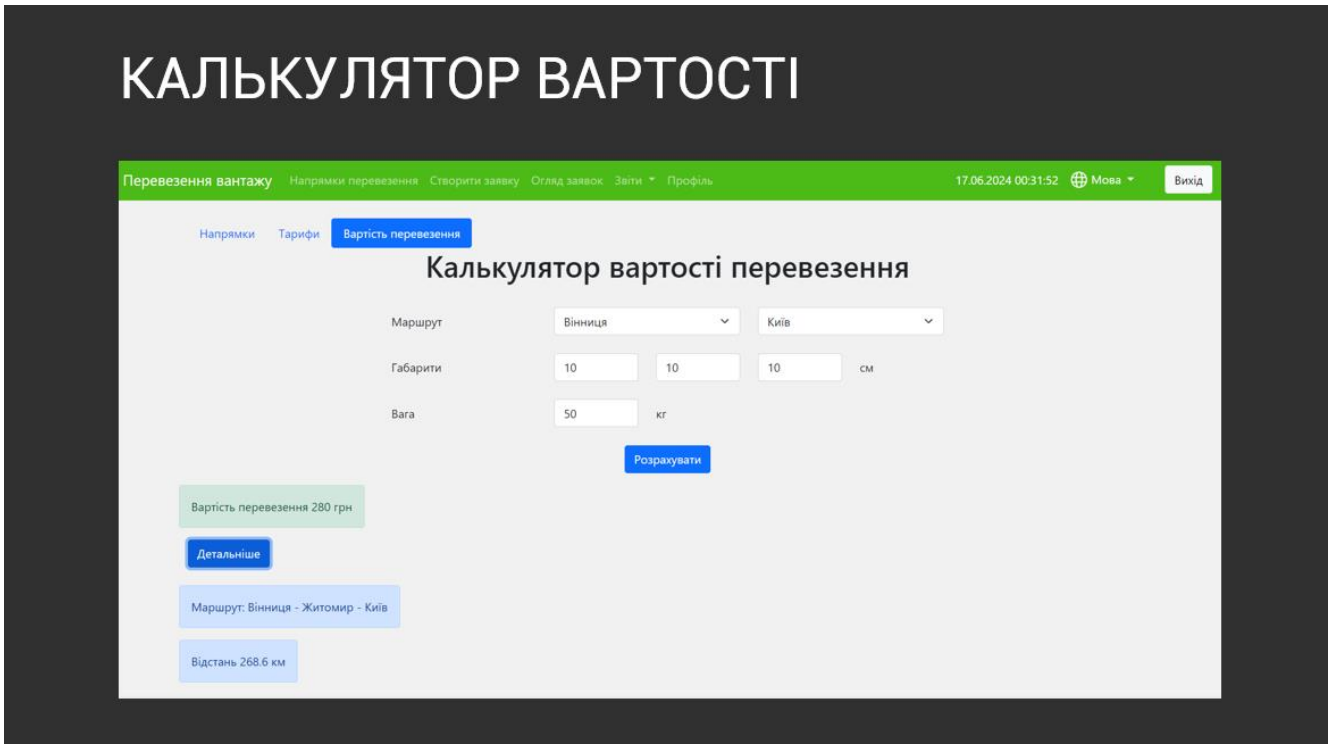

Рисунок Г.17 – Калькулятор вартості

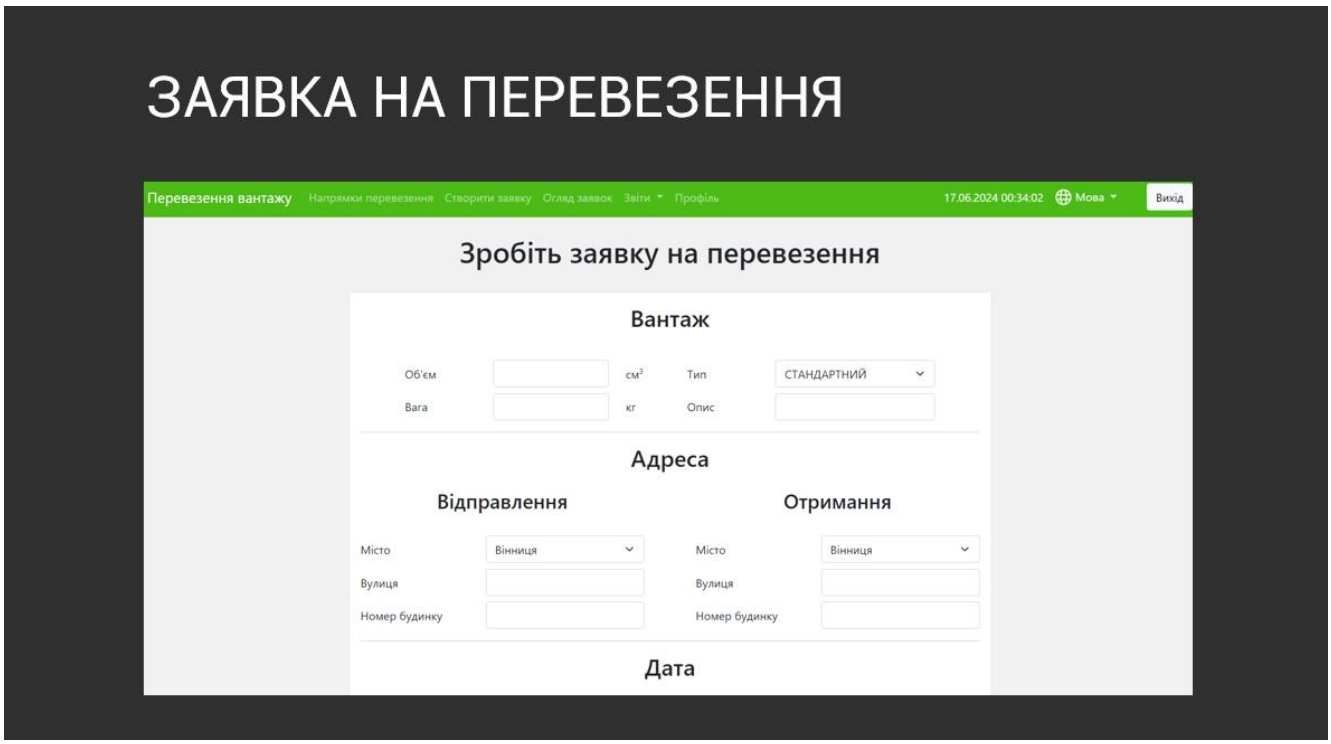

Рисунок Г.18 – Заявка на перевезення

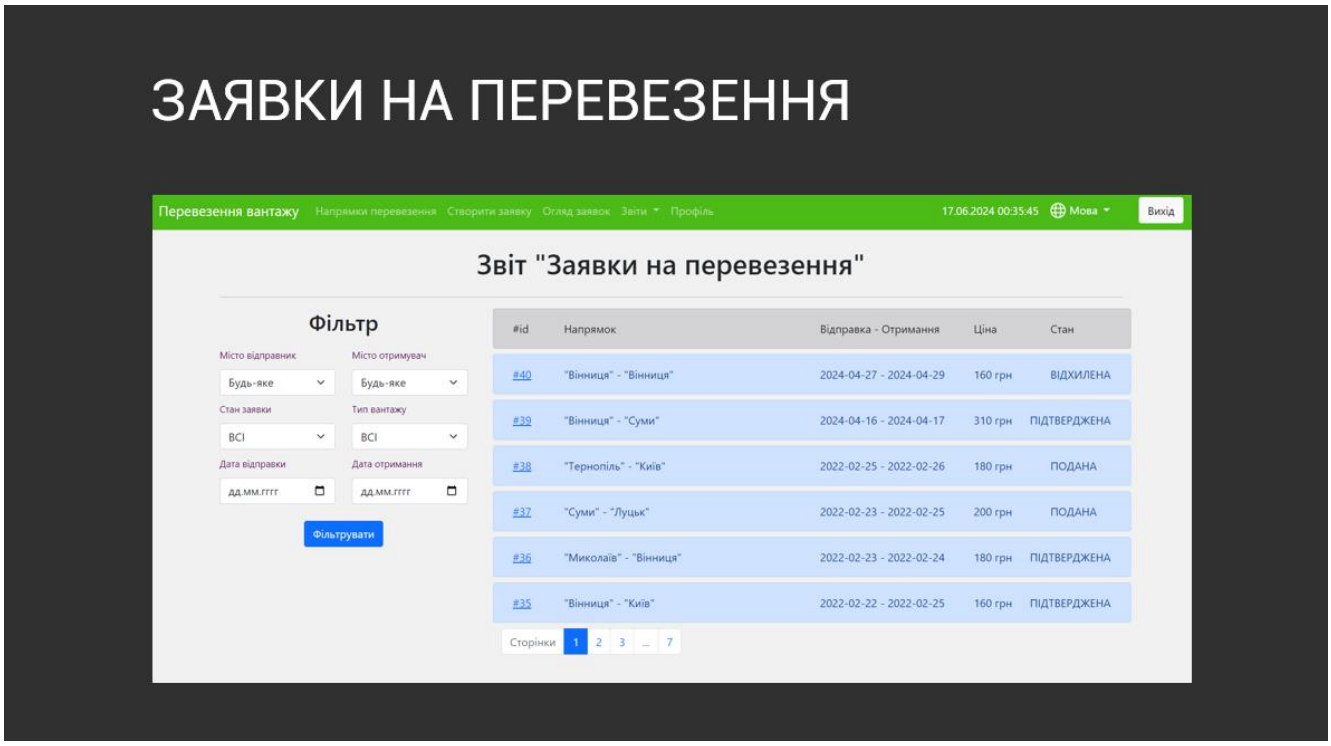

Рисунок Г.19 – Заявки на перевезення

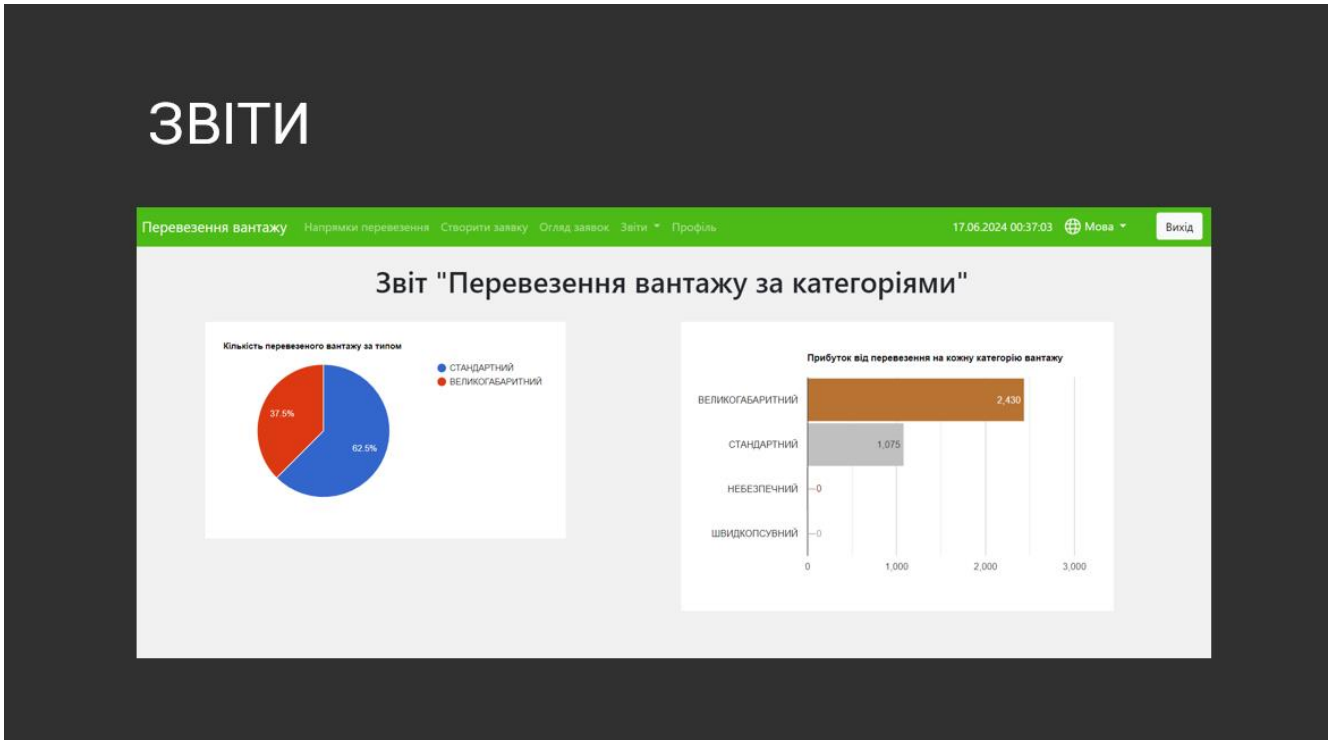

Рисунок Г.20 – Звіти

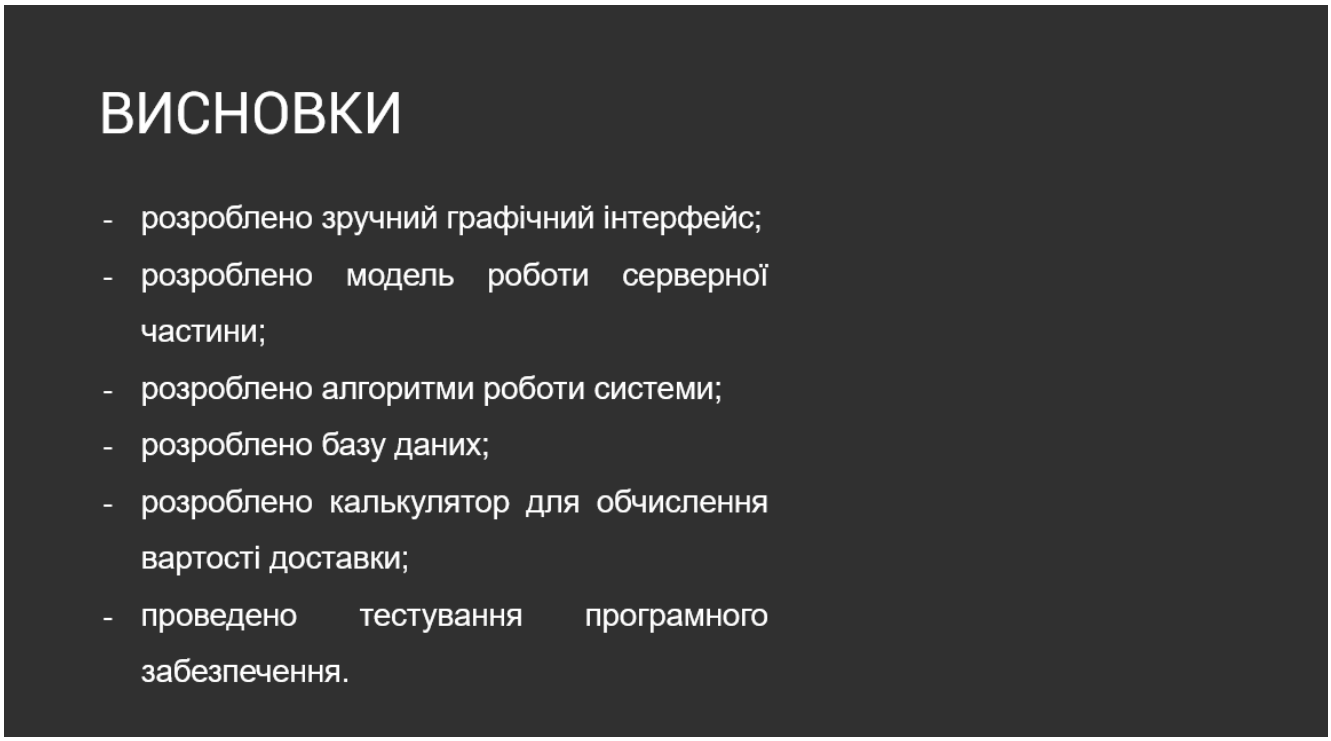

Рисунок Г.21 – Висновки