Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка методів і програмних засобів вебсистеми організованого доставлення посилок»

Виконав: студент II курсу
групи 1ПІ-24м
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Головачьов М.О.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Романюк О.В.
(прізвище та ініціали)
«10» грудня 2025 р.
Опонент: к.т.н., доц. каф. ОТ, Колесник І.С.
(прізвище та ініціали)
«10» грудня 2025 р.

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О.Н.
«10» грудня 2025 р.

ВНТУ – 2025

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

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
«25» вересня 2025 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Головачьову Михайлу Олександровичу

1. Тема роботи – Розробка методів і програмних засобів вебсистеми організованого доставлення посилок.

Керівник роботи: Романюк Оксана Володимирівна, к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від «24» вересня 2025 р. № 313.

2. Строк подання студентом роботи 10 грудня 2025р.

3. Вихідні дані до роботи: середовище розробки IntelliJ IDEA, мова розробки Java, метод пошуку координат за текстовою адресою, метод пошуку замовлень в радіусі встановленого відхилення від маршруту, метод формування рейтингу користувачів системи, метод формування статистики профілю користувача.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз стану вебсистем організованого доставлення посилок, розробка методів, моделі та алгоритмів роботи системи; розробка програмних засобів системи; тестування системи; висновки; список використаних джерел; додатки.

5. Перелік графічного матеріалу: титульний слайд; актуальність теми;

мета, об'єкт та предмет дослідження; задачі дослідження, наукова новизна, практична цінність одержаних результатів; порівняльний аналіз аналогів; використані технології, розробка методів; модель системи; загальний алгоритм роботи системи; загальна схема таблиць бази даних та їх зв'язків; сервісна топологія системи; тестування системи; демонстрація роботи; апробація результатів роботи і публікації; висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Романюк О.В., к.т.н., доцент кафедри ПЗ	26.09.2025	09.12.2025
5	Адлер О.О., к.т.н., доцент кафедри ЕПВМ	22.11.2025	08.12.2025

7. Дата видачі завдання 26 вересня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану вебсистем організованого доставлення посилок	26.09.2025 30.09.2025	виконано
2	Розробка методів, моделі та алгоритмів роботи системи	01.10.2025 17.10.2025	виконано
3	Розробка програмних засобів системи	18.10.2025 04.11.2025	виконано
4	Тестування системи	05.11.2025 21.11.2025	виконано
5	Економічна частина	22.11.2025 01.12.2025	виконано
6	Оформлення матеріалів до захисту МКР	01.12.2025 09.12.2025	виконано

Студент _____ Головачьов М.О.
(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ Романюк О.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.4:656.078

Головачьов М.О. Розробка методів і програмних засобів вебсистеми організованого доставлення посилок. Магістерська кваліфікаційна робота зі спеціальності 121 – Інженерія програмного забезпечення, освітня програма – Інженерія програмного забезпечення. Вінниця: ВНТУ, 2025. 124с.

На укр. мові. Бібліогр.: назв: 32; рисунків: 52; таблиць: 15.

У магістерській кваліфікаційній роботі подано результати дослідження програмних засобів організованого доставлення посилок. Обґрунтовано доцільність удосконалення методів організованого доставлення посилок.

Магістерська кваліфікаційна робота містить аналіз відомих підходів до організованого доставлення посилок.

Розроблено методи пошуку координат за текстовою адресою, пошуку замовлень в радіусі встановленого відхилення від маршруту, формування рейтингу користувачів системи, формування статистики профілю користувача.

Розроблено вебсистему організованого доставлення посилок, яка включає функціонал для пошуку координат за текстовою адресою користувачем, пошуку замовлень в радіусі встановленого відхилення від маршруту користувачем, формування рейтингу користувачів системи та формування статистики профілю користувача. Всі дані користувача зберігаються в базі даних PostgreSQL, що забезпечую цілісність та захищеність даних користувача, а також високу продуктивність при роботі з даними.

Ключові слова: організоване доставлення посилок, краудшипінг, Java, Spring Boot, PostgreSQL.

ABSTRACT

Holovachov M.O. Development of methods and software for a web system for organized parcel delivery. Master's qualification work in the specialty 121 – Software Engineering, educational program – Software Engineering. Vinnytsia: VNTU, 2025. 124 pages.

In Ukrainian. Bibliography: titles: 32; figures 52; tables: 15.

The master's qualification work presents the results of the study of software tools for organized parcel delivery. The feasibility of improving the methods of organized parcel delivery is substantiated.

The master's qualification work contains an analysis of known approaches to organized parcel delivery.

Methods for searching for coordinates by text address, searching for orders within a radius of a set deviation from the route, forming a rating for system users, and forming user profile statistics have been developed.

A web system for organized parcel delivery has been developed, which includes functionality for searching for coordinates by text address by the user, searching for orders within a radius of a set deviation from the route by the user, forming a rating for system users, and forming user profile statistics. All user data is stored in the PostgreSQL database, which ensures the integrity and security of user data, as well as high performance when working with data.

Keywords: organized parcel delivery, crowdshipping, Java, Spring Boot, PostgreSQL.

ЗМІСТ

ВСТУП

Актуальність роботи. Організоване доставлення посилок – це сучасний підхід до логістики дрібних відправлень, за якого переміщення посилок планується та координується через цифрові платформи з урахуванням маршрутів потенційних перевізників. Фактично це форма спільного використання транспортних ресурсів, що дозволяє поєднати потребу у відправленні посилки з уже запланованими поїздками людей чи транспорту. Така модель є розвитком тенденцій електронної комерції, економіки спільного користування та цифровізації побутових сервісів [1].

Стрімке зростання обсягів онлайн-покупок і посилкових відправлень створює значне навантаження на традиційні поштові та кур’єрські служби. Класичні логістичні моделі часто орієнтовані на великі центри обробки, стандартизовані маршрути та фіксовані тарифи, що не завжди є вигідним і зручним для пересилання невеликих або нерегулярних партій вантажів. У результаті зростає потреба у гнучкіших сервісах, здатних адаптуватися до маршрутів користувачів, скорочувати витрати часу й коштів та підвищувати ефективність використання наявної транспортної інфраструктури [2].

Поширеною неформальною практикою є передача посилок через знайомих, попутників чи оголошення в соціальних мережах. Проте такий підхід не має чітких правил, механізмів відповідальності та інструментів контролю, що призводить до конфліктних ситуацій, втрати або пошкодження відправлень.

Саме для того, щоб об’єднати всі вищезазначені засоби та спростити користувачеві процес планування маршрутів, пошуку релевантних перевезень, прозорого узгодження умов і контролю виконання доставки, потрібно використовувати готовий програмний застосунок. Тому актуальною є розробка вебсистеми організованого доставлення посилок.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою магістерської кваліфікаційної роботи є підвищення рівня доставлення посилок шляхом розробки та впровадження вебсистеми організованого доставлення посилок, що дозволить підвищити якість доставлення посилок.

Основними завданнями дослідження є:

– розробити модель, методи та алгоритми роботи вебсистеми організованого доставлення посилок;

– розробити функціонал для пошуку координат за текстовою адресою;

– розробити функціонал для пошуку замовлень в радіусі встановленого відхилення від маршруту;

– розробити функціонал для формування рейтингу користувачів системи;

– розробити функціонал для формування статистики профілю користувача;

– провести тестування розробленої системи.

Об’єктом дослідження є процес розробки вебсистеми організованого доставлення посилок.

Предметом дослідження є методи та засоби розробки вебсистеми організованого доставлення посилок.

Методи дослідження. У процесі досліджень використовувались методи дослідження:

– методи розробки вебсистеми, доступної з будь-якого сучасного браузера для реалізації вебсистеми організованого доставлення посилок;

– методи розробки інтерфейсу вебсистеми організованого доставлення посилок для забезпечення зручної взаємодії користувача з системою;

– методи автоматичного обчислення аналітики за вхідними даними користувача для можливості ведення статистичних досліджень;

– методи роботи з шаблонізатором Thymeleaf та елементами HTML, CSS та Javascript для реалізації графічного інтерфейсу системи.

Наукова новизна отриманих результатів.

1. Подальшого розвитку отримав метод пошуку координат за текстовою

адресою, який, на відміну від відомих, використовує багатоетапну нормалізацію та валідацію текстових даних, інтеграцію з зовнішнім геокодувальним сервісом Geocode API, а також кешування отриманих результатів у PostgreSQL за допомогою PostGIS через Spring Data JPA, що забезпечує підвищену точність визначення координат за неповними або неточними адресами, зменшує кількість помилкових збігів і скорочує час оброблення повторних запитів.

2. Подальшого розвитку отримав метод пошуку замовлень в радіусі встановленого відхилення від маршруту, який, на відміну від відомих, ґрунтується на векторизованому поданні маршруту у вигляді polyline, сформованому за допомогою маршрутизатора OSRM, розрахунку мінімальної відстані від точки замовлення до будь-якого сегмента маршруту з урахуванням допустимого радіуса відхилення та можливості ранжування замовлень за додатковими просторово-часовими критеріями, що забезпечує виявлення релевантних замовлень уздовж траєкторії руху та мінімізацію зайвих об’їздів.

3. Подальшого розвитку отримав метод формування рейтингу користувачів системи, який, на відміну від існуючих, реалізований у вигляді багатофакторної агрегувальної моделі на рівні СУБД PostgreSQL, де за допомогою спеціально спроєктованих SQL-запитів виконується підрахунок кількості виконаних замовлень, усереднення оцінок за відгуками та обчислення показників активності з подальшим нормуванням цих величин до єдиної шкали та їх зваженим підсумовуванням; результати передаються у застосунок через Spring Data JPA у вигляді окремої проєкції рейтингу, що забезпечує гнучкий механізм формування рейтингового бала без дублювання логіки на рівні інтерфейсу та спрощує подальше розширення моделі оцінювання.

4. Подальшого розвитку отримав метод формування статистики профілю користувача, який, на відміну від традиційних підходів, використовує багаторівневе агрегування операційних даних у PostgreSQL із застосуванням групувальних обчислень, а також подальшу публікацію цих даних у вигляді REST-ендпоінтів і візуалізацію у вебінтерфейсі за допомогою Thymeleaf та бібліотеки Chart.js, що забезпечує наочне подання індивідуальної ефективності,

підтримує прийняття рішень щодо покращення сервісу й дає змогу адаптувати роботу системи до реальних моделей поведінки її учасників, шляхом аналізу статистики.

Практична цінність отриманих результатів. Практичне значення отриманих результатів полягає в тому, що на основі отриманих у магістерській кваліфікаційній роботі теоретичних положень запропоновано алгоритми та програмні засоби для підвищення рівня доставлення посилок шляхом розробки та впровадження вебсистеми організованого доставлення посилок, що дозволить підвищити якість доставлення посилок.

Особистий внесок здобувача. Усі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто. У наукових роботах [3] [4], опублікованій у співавторстві, автору належать такі результати: загальний алгоритм роботи системи, таблиця порівняння аналогів, обгрунтування використаних інструментів.

Апробація матеріалів. Описані у магістерській кваліфікаційній роботі положення доповідалися на V Всеукраїнської науково-технічної конференції молодих вчених, аспірантів і студентів «Комп'ютерні ігри та мультимедіа як інноваційний підхід до комунікації – 2025» та на Міжнародній науково-практичній інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ – 2025».

Публікації. Результати роботи опубліковані в тезах доповіді на V Всеукраїнської науково-технічної конференції молодих вчених, аспірантів і студентів «Комп'ютерні ігри та мультимедіа як інноваційний підхід до комунікації – 2025» [3] та на Міжнародній науково-практичній інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ – 2025» [4].

Структура та обсяг роботи. Магістерська кваліфікаційна робота складається зі вступу, п’яти розділів, висновків, списку використаних джерел, що містить 32 найменування, 4 додатків. Робота містить 52 ілюстрації, 15 таблиць.

1 АНАЛІЗ СТАНУ ВЕБСИСТЕМ ОРГАНІЗОВАНОГО ДОСТАВЛЕННЯ ПОСИЛОК

1.1 Аналіз стану вебсистем організованого доставлення посилок

В сучасному світі, в час швидкого розвитку інформаційних технологій, дуже важливу роль відіграє доставка посилок. Поштові компанії є дуже потужним інструментом на рівні з мережею Інтернет та телефонією, який забезпечує доступ до товарів та послуг незалежно від місцезнаходження клієнта, який користується послугами доставки.

Традиційні служби доставки, незважаючи на високий рівень розвитку інфраструктури, стикаються з низкою проблем: значні часові витрати, складнощі логістики, потреба в централізованих складах. Саме через модель використання централізованих складів, або так званих терміналів, з’являється підвищена вразливість до зовнішніх загроз, зокрема шкоди внаслідок військових операцій чи стихійних лих.

Існуючі методи організації доставки умовно можна розділити на три групи:

1. Централізовані логістичні моделі. Використовуються традиційними поштовими операторами та великими кур'єрськими службами. Базуються на складській інфраструктурі та транспортних вузлах. Клієнт може обрати будь-яке зручне для нього поштове відділення. Це, безумовно, має переваги, такі як масштабованість та стандартизація. Але з іншого боку, вони залежать від централізованих вузлів, які мають вразливість до їх пошкодження та високу вартість.

2. Краудшипінгові моделі доставки – концепція краудсорсингу для персоналізованої доставки вантажів та відправлень. Краудшипінг можна розглядати як взаємодії людей, які використовують соціальні мережі для спільних взаємовигідних дій та обміну послугами і активами задля більшого блага громади, а також для своєї особистої вигоди. Передбачають залучення приватних осіб до транспортування посилок «по дорозі». Основним інструментом є веб-платформи, які координують взаємодію між відправником та виконавцем. Такі

моделі сприяють децентралізації, збільшуючи швидкість та знижуючи вартість доставки. Також великою перевагою є значно менша ймовірність пошкодження посилки через стихійні лиха та інші зовнішні фактори на централізованих вузлах зберігання посилок.

3. Гібридні моделі. Поєднують елементи централізованої інфраструктури та краудшипінгового підходу. Наприклад, використання пунктів видачі разом з окремими перевізниками. Ця модель є компромісом для існуючого бізнесу, щоб сумістити переваги обох моделей з мінімальними витратами на переобладнання робочого процесу [5].

Оскільки гібридна модель більше відноситься до рішень бізнесу та конкретних вимог тієї чи іншої компанії, в цій роботі буде розглядатися виключно краудшипінгова модель.

Ідея базується на використанні децентралізованого підходу до логістики, де ключову роль відіграє не стільки інфраструктура, скільки взаємодія між користувачами. На відміну від традиційних сервісів, що вимагають обов'язкового проходження посилки через центральні склади або сортувальні термінали, запропонована модель використовує можливість передачі відправлень між учасниками, які вже мають власні маршрути руху. Таким чином, основна ідея полягає в поєднанні необхідності доставки з існуючими транспортними потоками, що існують незалежно від логістичних компаній. Аспект децентралізації має особливу цінність. У випадках, коли традиційна логістика порушується – наприклад, через обмеження транспортних коридорів, військові операції або стихійні лиха – краудшипінгова система доставки залишається працездатною. Відсутність великих терміналів і складів знижує ризик одночасної втрати великих обсягів відправлень, а географічно розподілена структура учасників забезпечує більшу стійкість до зовнішніх загроз.

Зрозуміло, що цей метод сам по собі не зможе впоратися з усіма потребами в доставці через їх високий обсяг у наш час. Однак його використання як альтернативи значно зменшить навантаження на традиційні

служби доставки.

В багатьох компаній є клієнтські системи як у вигляді вебсистеми, так і у вигляді мобільного застосунку. Часто в них дуже схожий перелік функцій – відстеження статусу посилки, список актуальних замовлень та маніпуляції з замовленням, такі як переадресація, відмова тощо. Інколи є можливість оплати онлайн та створити накладну онлайн. Мобільний застосунок є більш зручним, якщо користувачі частіше для доступу до ресурсу використовують телефони чи планшети, але вебсистема є більш універсальним рішення, яка поєднує в одному клієнті можливості використовувати ресурс з багатьох платформ. Також потрібно підмітити, що за умови правильного підходу до мобільної адаптивності вебсистеми, користувачі не отримають дискомфорту при перегляді з телефону або планшету.

Одним з можливих рішень є створення вебсистеми організованої доставки посилок, де роль кур'єра виконує звичайний користувач, який пересувається маршрутом, що співпадає з кінцевою адресою доставки і може забрати відправлення з собою.

Отже, актуальною є розробка вебсистеми організованого доставлення посилок.

1.2 Аналіз аналогів розроблюваної вебсистеми

Раціональне проєктування вебсистеми організованого доставлення посилок неможливе без критичного огляду наявних моделей і платформ. Такий аналіз слугує підставою для визначення мінімально необхідного функціоналу, формальних інваріантів домену та нефункціональних вимог, які повинні бути задоволені з першої ітерації розвитку системи. Порівняння різних підходів дозволяє також уникнути технологічної імітації, зосередившись на тій комбінації можливостей, що забезпечує максимальну корисність для користувача за помірної складності реалізації й прийнятних операційних ризиків.

TravelPost [6] – це двосторонній додаток, де можна як запропонувати

передати посилку, так і попросити її доставити. Він позиціонувався як C2C-маркетплейс краудшипінгу, що поєднує мандрівників, які прагнуть частково компенсувати витрати на поїздки, та відправників, яким потрібна швидша або дешевша доставка невеликих відправлень, з акцентом на крос-кордонні перевезення. У публічних профілях сервіс фігурує як український стартап, заснований у 2017 році, із визначенням «crowdshipping community» та заявленою місією скорочення вартості й часу міжнародних пересилань для звичайних користувачів, де традиційні перевізники інколи виявляються надто повільними або дорогими.

Логотип та інтерфейс додатка зображені на рисунку 1.1.

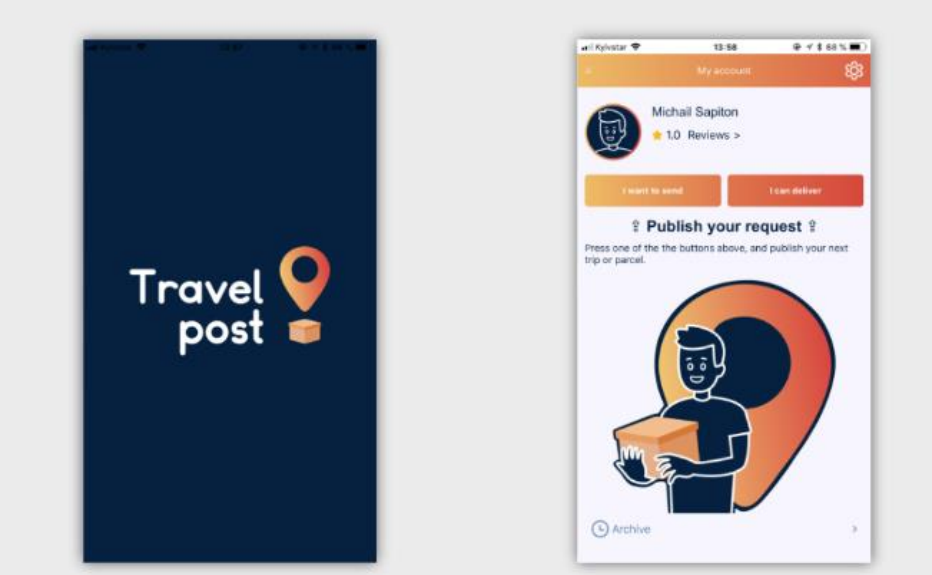

Рисунок 1.1 – Логотип та інтерфейс застосунку TravelPost

Сильні сторони такого підходу логічно випливають із зазначених принципів. Перш за все, це потенціал часової та маршрутної гнучкості на нетипових напрямах, особливо для транскордонних доставлень, де традиційні ланцюги створюють затримки на консолідаційних етапах. Далі, економія на «порожніх» або частково завантажених поїздках мандрівників створює можливість конкурентної ціни для дрібних відправлень і «рідкісних» пар походження-призначення. Нарешті, низький поріг входу для учасників може швидко масштабувати пропозицію в густонаселених коридорах, де достатньо

мандрівників із релевантними маршрутами, а цифровий маркетплейс бере на себе лише координацію та комунікацію сторін. Сукупно це формує привабливу ціннісну пропозицію для сегментів, які не покриваються ефективно класичною мережею.

Fleetli [7] – веб-сайт та мобільний додаток, що поєднує людей, які хочуть відправити посилку, з тими, хто готовий взяти її з собою в подорож, що дозволяє значно заощадити на доставці та компенсувати дорожні витрати. Ключова ідея – монетизація вільної місткості в уже запланованих поїздках; відправник формує запит із параметрами маршруту і типом вантажу, а мандрівники переглядають відповідні пропозиції та узгоджують передачу. Комунікація доповнюється рейтинговими індикаторами та історією взаємодій, що покликані знизити інформаційну асиметрію між сторонами. Офіційні матеріали сервісу акцентують на «простоті» і «економічності» обміну, а також на підборі надійних виконавців за рахунок базових соціальних сигналів довіри.

Інтерфейс застосунку зображений на рисунку 1.2.

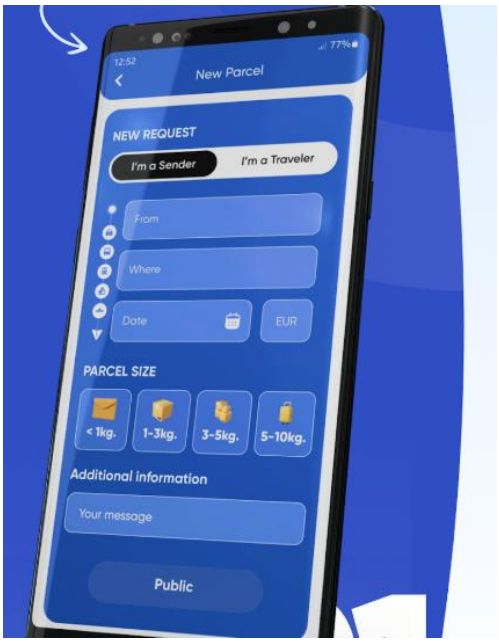

Рисунок 1.2 – Інтерфейс застосунку Fleetli

Переваги моделі Fleetli полягають у її наближеності до «природного» сценарію поїздки: платформа не створює новий ланцюг транспортування, а

під’єднується до вже існуючих маршрутів користувачів, що відкриває можливість зниження граничних витрат і прискорення адресних доставлень на нетипових напрямах. Наявність рейтингових індикаторів і транзакційної історії формує мінімальний каркас довіри без громіздких процедур перевірки особи.

GoGoBag [8] – сервіс, що допомагає скоротити викиди CO_2, об'єднуючи попутників та перевізників, які можуть взяти посилку у свою мандрівку. Це відповідальний підхід до перевезень, адже таким чином логістичним компаніям не потрібно здійснювати рейси. GoGoBag розвиває сервіс швидких відправлень в Україні та між Україною і країнами ЄС, поєднуючи відправників із кур’єрами або подорожніми. В описі сервісу підкреслюється можливість обрати виконавця за маршрутом, датою і ціною, отримувати відстеження в реальному часі та організовувати міжнародні пересилання з гнучкими умовами щодо масово-габаритних параметрів. Окремо комунікуються «прості кроки» оформлення міжнародного відправлення, що знижує бар’єр входу для побутових користувачів.

Інтерфейс застосунку GoGoBag зображено на рисунку 1.3.

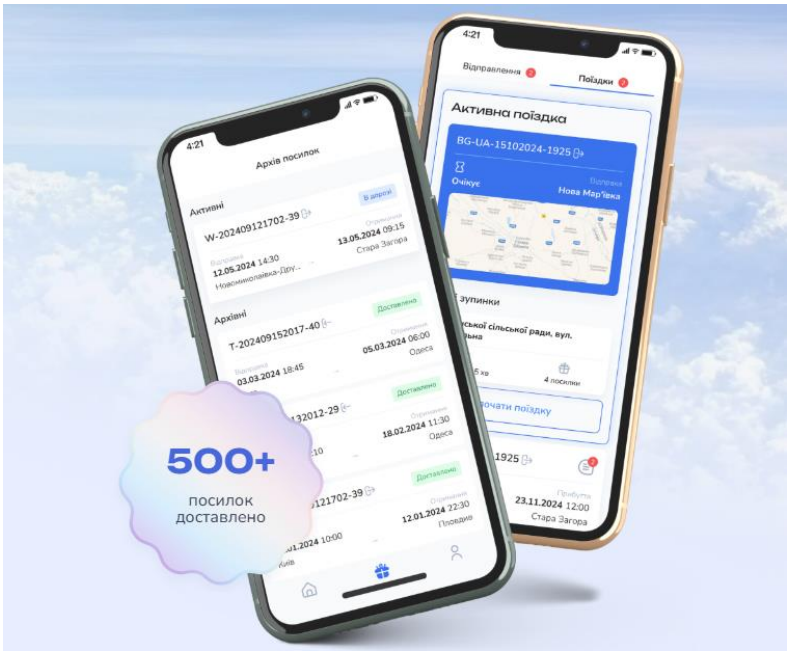

Рисунок 1.3 – Інтерфейс застосунку GoGoBag

Переваги рішення проявляються у поєднанні зрозумілого користувацького

шляху з базовим інструментарієм контролю: можливістю вибору кур'єра за релевантними параметрами, трекінгом і прозорими ціновими орієнтирами для різних категорій відправлень. Орієнтація на міжнародні маршрути з України формує помітну додану цінність для діаспори та транскордонних відправників.

Nova Post [9] – міжнародна група компаній, яка надає приватним та бізнес-клієнтам повний спектр логістичних та пов'язаних з ними послуг. представляє класичного оператора з розгалуженою мережею відділень, поштоматів і кур'єрських сервісів для B2C та B2B сегментів, включно з міжнародними відправленнями через Nova Poshta Global. Сервісна модель охоплює «від дверей до дверей», доставлення у відділення і поштомати, а також інтеграції для електронної комерції і глобальних відправок із супутніми митними процедурами. Публічні матеріали підкреслюють наявність мобільного застосунку, трекінгу, виклику кур'єра і карти інфраструктури, що формує стандарт очікувань для локального ринку.

На рисунку 1.4 зображено інтерфейс вебзастосунку Nova Post.

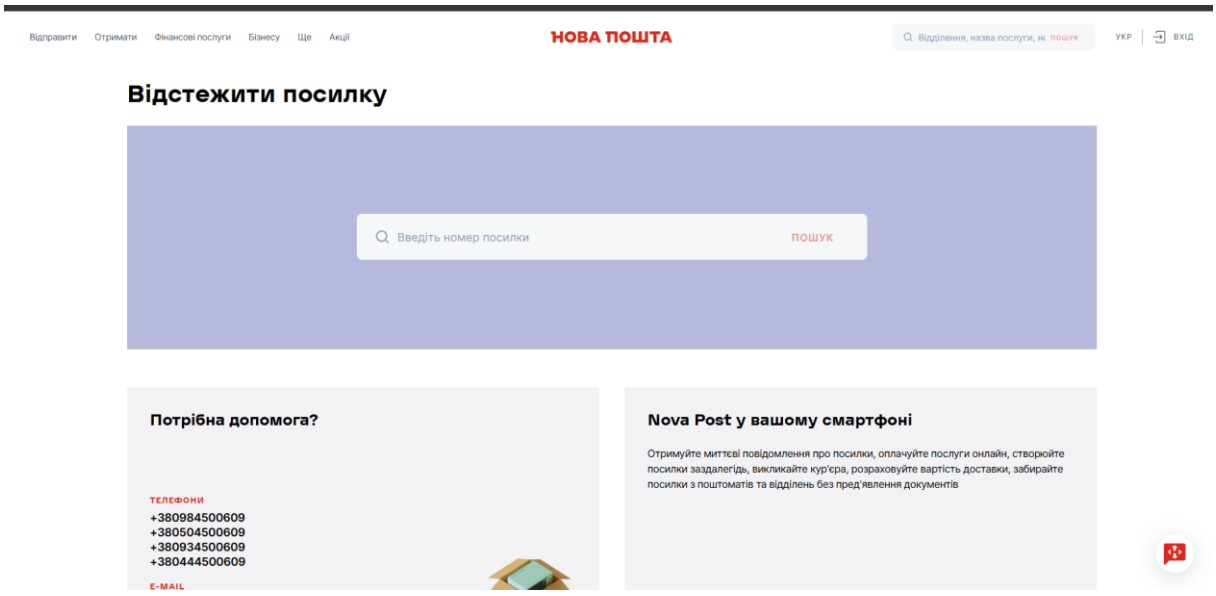

Рисунок 1.4 – Інтерфейс вебзастосунку Nova Post

Переваги підходу Nova Poshta виявляються у високому рівні стандартизації і відтворюваності процесів: широка мережа точок, передбачувані терміни в типових коридорах, сервісна підтримка бізнес-клієнтів і зрілий набір

API під потреби e-commerce. У межах міських агломерацій та магістральних напрямів класична мережа забезпечує стабільність і масштабованість. Проте, незважаючи на ці переваги, Nova Poshta не підтримує краудшипінгову модель, а тому є вразливою через необхідність централізованих складів та сортувальних терміналів.

Наочна демонстрація відмінностей аналогів описана в таблиці 1.1.

Таблиця 1.1 – Порівняння характеристик систем доставлення посилок

Критерій	TravelPost	Fleetli	GoGoBag	Nova Post	Власна розробка
Можливість використання системи з браузера	0	0	0	1	1
Можливість відображення доставок на інтерактивній мапі	0	0	1	0	1
Можливість пошуку замовлення поблизу маршруту користувача	0	1	0	0	1
Наявність статистики про виконані доставки	0	0	0	0	1
Наявність рейтингової системи користувачів	1	1	0	0	1
Загальна оцінка	1	2	1	1	5

Отже, за розглянутими критеріями, власна розробка є актуальною та має деякі переваги перед існуючими аналогами.

Проаналізувавши існуючі аналоги, було сформовано перелік функіональних вимог, які повинна мати розроблювана вебсистема організованого доставлення посилок, а саме наявність: системи аутентифікації

користувачів, фільтрації релевантних пропозицій за місцезнаходженням, система рейтингу користувачів, модуль статистики, зручний інтерфейс, візуалізація доставок на мапі.

1.3 Аналіз методів розв’язання задачі

Основним методом розв’язання поставленої задачі під час розробки вебсистеми організованого доставлення посилок є автоматизоване визначення географічного розташування користувацьких адрес, що забезпечує коректну роботу навігаційних і логістичних функцій системи.

Метод пошуку координат за текстовою адресою дає змогу перетворити довільно введену поштову адресу на точні географічні координати. Після введення адреси користувачем система звертається до зовнішнього геокодувального API, який повертає перелік можливих відповідників, ранжованих за релевантністю. З отриманого набору система обирає оптимальний результат, після чого числові координати трансформуються у внутрішнє геопросторове представлення. Це дозволяє системі проводити подальші розрахунки – пошук найближчих об’єктів, побудову маршрутів або визначення радіуса відхилення. У випадку невдалої геокодувальної спроби система повертає індикацію помилки, що забезпечує можливість повторного уточнення адреси без зупинки загального робочого процесу.

Другим ключовим методом є метод пошуку замовлень у радіусі допустимого відхилення від маршруту, який дозволяє оптимізувати логістику та підвищити ефективність процесу доставлення посилок.

Користувач визначає початкову та кінцеву точки маршруту, а також максимальну відстань, на яку він готовий відхилитися. Система геокодує введені адреси, після чого надсилає запит до зовнішнього маршрутизувального сервісу, який повертає детальний маршрут у вигляді множини опорних точок. Паралельно система отримує актуальні перевізницькі пропозиції зі сховища даних та обчислює мінімальну відстань між координатами кожної пропозиції та маршрутом користувача. Якщо відправна та кінцева точки пропозиції

потрапляють у зону допустимого відхилення, така пропозиція вважається сумісною з маршрутом. Це дає змогу автоматично формувати оптимальні підбірки послуг, що не потребують значних об’їздів, та підвищити ефективність використання транспортних ресурсів.

Важливою складовою системи є метод формування рейтингу користувачів, що забезпечує прозорість взаємодії між учасниками сервісу.

Після завершення замовлення користувач може оцінити іншого учасника, поставивши рейтинг за п’ятибальною шкалою та додавши текстовий коментар. Перед збереженням відгуку система виконує низку перевірок: підтверджує, що значення рейтингу належить допустимому діапазону, перевіряє, що користувач не оцінює самого себе, та гарантує, що автор відгуку справді був учасником відповідного замовлення. Також перевіряється, чи не існує вже відгуку від цього автора щодо цього ж замовлення, що запобігає дублюванню і штучному накручуванню рейтингу. Після успішної валідації відгук зберігається у системі, де може бути врахований у загальній оцінці користувача.

Завершальним елементом є метод формування статистики профілю користувача, який забезпечує узагальнення ключових показників його активності.

Система отримує дані про користувача, після чого обчислює середній рейтинг, кількість отриманих відгуків, а також формує розподіл оцінок за всіма можливими значеннями. Паралельно аналізуються дії користувача у ролі відправника та перевізника: кількість створених замовлень, кількість успішних доставок, скасованих замовлень, завершених та анульованих бронювань. На основі цих даних формується структурований набір характеристик, який використовується для побудови графічних елементів, діаграм і узагальнених статистичних блоків у профілі. Це дозволяє користувачу бачити власну активність у динаміці та оцінювати якість взаємодії з іншими учасниками сервісу.

У сукупності зазначені методи забезпечують створення цілісної вебсистеми, здатної автоматично обробляти адреси, маршрути, перевізницькі

пропозиції та дані взаємодії користувачів. Їхнє поєднання створює інтелектуальну платформу, що підтримує організоване доставлення посилок і підвищує ефективність, надійність і зручність роботи сервісу.

1.4 Постановка задач дослідження

Провівши аналіз методів розв’язання поставленої задачі, аналіз стану застосунків для організованого доставлення посилок, порівняння аналогів було визначено такі завдання дослідження:

– розробити модель, методи та алгоритми роботи вебсистеми організованого доставлення посилок;

– розробити функціонал для пошуку координат за текстовою адресою;

– розробити функціонал для пошуку замовлень в радіусі встановленого відхилення від маршруту;

– розробити функціонал для формування рейтингу користувачів системи;

– розробити формування статистики профілю користувача;

– провести тестування розробленої системи.

Технічне завдання на розробку наведено в додатку А.

1.5 Висновки

У першому розділі було проведено аналіз стану питання розробки вебсистеми організованого доставлення посилок, а також порівняльний аналіз аналогів: TravelPost, Fleetli, GoGoBag, Nova Post. Завдяки цьому визначено основні недоліки існуючих систем.

Було проаналізовано методи розв’язання задачі дослідження, що дозволило визначити функціональні вимоги та сформувати комплекс критеріїв, необхідних для створення ефективної вебсистеми.

На основі отриманих результатів виконано постановку задач дослідження та прийнято рішення розробити вебсистему організованого доставлення посилок, що забезпечить користувачеві зручний інструмент для керування доставками посилок.

2 РОЗРОБКА МЕТОДІВ, МОДЕЛІ ТА АЛГОРИТМІВ РОБОТИ СИСТЕМИ

2.1 Розробка архітектури системи

Архітектурна модель системи визначає спосіб перетворення вхідних подій користувача на узгоджені зміни стану даних і візуальне представлення результатів. З огляду на домен «організованого доставлення посилок» доцільним є прийняття класичної парадигми розшарування за відповідальністю, де обробка запитів, інкапсуляція бізнес-правил та персистентність даних залишаються незалежними площинами. Для опису структурних рішень у цьому розділі використовується модель MVC та серверний рендеринг представлення; технологічні засоби будуть обґрунтовані окремо, тому далі застосовується інструментально-агностичний виклад із фокусом на ролях шарів, потоках даних і взаємодії сервісів.

Архітектурний паттерн MVC (Model-View-Controller) [10] є одним з найпоширеніших та ефективних підходів до розробки веб-додатків. Він дозволяє відокремити логіку додатку від представлення та обробки даних, що сприяє підтримці коду, його розширенню та збереженню чистоти.

Візуальне представлення моделі MVC зображено на рисунку 2.1.

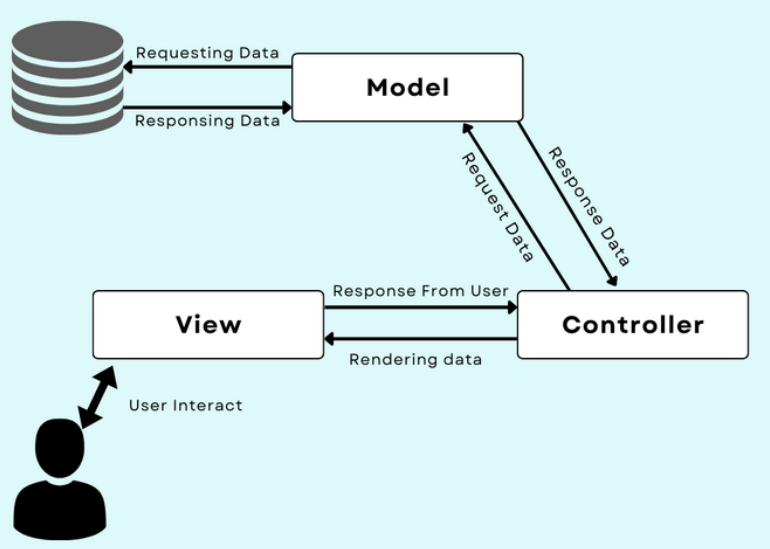

Рисунок 2.1 – Архітектура MVC

В основі обраної парадигми лежить чітке розмежування між представленням, логікою застосунку та моделлю даних. Модель уособлює стабільні інваріанти предметної області: користувачі, відправлення, пропозиції перевезення, бронювання та події життєвого циклу. Контролер виступає координатором, що приймає запит, виконує валідації, активує варіанти використання і формує модель для відображення. Подання відповідає за побудову кінцевого HTML-документа, спираючись на дані, що отримані після застосування бізнес-правил. Такий поділ створює передумови до керованої еволюції: зміни у правилах узгодження часових вікон або в алгоритмах просторового відбору не вимагають модифікацій у структурі HTML [11], а удосконалення інтерфейсу не впливають на персистентність.

Рендеринг представлення на стороні сервера обрано як базовий механізм побудови клієнтських сторінок. Ця стратегія забезпечує детермінований перший відгук із повністю сформованим HTML, що позитивно впливає на час до відображення, індексацію пошуковими системами та доступність; водночас зменшується обсяг клієнтської логіки, яка могла б дублювати перевірки або бізнес-обмеження. Серверний рендеринг спрощує контроль за безпекою відображуваних даних, адже екранування, локалізація й додержання протоколів підтвердження виконуються до того, як документ потрапляє в браузер. Додатково він знижує ризик розсинхронізації станів між клієнтом і сервером, оскільки джерело істини знаходиться на боці застосунку; динаміка взаємодій може доповнюватися прогресивним покращенням – картографічними віджетами, інтерактивними підказками чи асинхронними оновленнями – але фундаментальний життєвий цикл запитів і відповідей залишається серверно-центрованим.

Конструкція загального потоку запиту формується як низка послідовних перетворень. Вхідний HTTP-запит до контролера перевіряється на валідність та права доступу; далі активується варіант використання, який взаємодіє зі службами просторового пошуку й модулем узгодження часових вікон, а також з підсистемою персистентності для транзакційного збереження змін. На виході

формується модель для представлення з явно окресленими даними та повідомленнями, після чого шар подання будує HTML-відповідь. Для інтенсивних шляхів, зокрема просторової фільтрації пропозицій за координатами відправлення і призначення, застосовується попередня оптимізація запитів за рахунок індексів і скорочених проєкцій; результати надходять до подання у впорядкованому вигляді разом із метаданими, необхідними для візуалізації на карті.

У результаті реалізація шаблону MVC у вебсистемі організованого доставлення посилок сприяла формуванню якісної архітектури, забезпечила стабільну роботу системи та зручність її подальшого обслуговування, що відповідає основним вимогам, які висуваються до сучасного програмного забезпечення.

2.2 Розробка методу пошуку координат за текстовою адресою

Метод пошуку координат за текстовою адресою призначений для автоматизованого перетворення введеного користувачем текстового опису місця (наприклад, міста, вулиці або повної поштової адреси) у географічні координати, придатні для подальшого використання в системі. Для цього застосунок звертається до зовнішнього геокодувального сервісу, який на основі заданої адреси повертає впорядкований набір можливих відповідників. Із отриманих результатів обирається найбільш релевантний варіант, для якого відомі значення географічної широти та довготи.

Подальшим кроком є перетворення пари числових координат на внутрішнє геопросторове представлення у вигляді точки. Така точка використовується як універсальний формат зберігання та обробки просторових даних у системі, зокрема під час виконання пошуку об’єктів у заданому радіусі, побудови маршрутів або формування просторових вибірок у сховищі даних. У випадку, коли зовнішній сервіс не повертає жодного придатного результату або під час обробки відповіді виникає помилка, метод повертає індикацію відсутності координат для заданої адреси. Це дає змогу вищим рівням логіки

коректно реагувати на такі ситуації (наприклад, повідомляти користувача або пропонувати уточнити адресу), не порушуючи загальний процес роботи системи.

Покроковий опис роботи методу пошуку координат за текстовою адресою можна подати у такій послідовності кроків:

1. Користувач вводить текстову адресу (наприклад, адресу відправника чи одержувача) у відповідному полі інтерфейсу веб-застосунку.

2. Застосунок передає введену адресу у вигляді рядка до спеціалізованого сервісного методу геокодування.

3. Метод виконує початкову перевірку коректності вхідних даних: якщо рядок адреси порожній або містить лише пробіли, подальша обробка не виконується, формується індикація відсутності результату, а ситуація може бути зафіксована засобами моніторингу.

4. Текстова адреса попередньо готується до передачі у запиті до зовнішнього сервісу (наприклад, шляхом належного кодування символів), щоб гарантувати коректне трактування її в мережевому протоколі.

5. На основі базової адреси зовнішнього геокодувального сервісу та підготовленого текстового опису місця формується повний запит, який містить усю необхідну інформацію для пошуку координат.

6. Компонент мережевої взаємодії виконує запит до зовнішнього геокодувального сервісу, очікуючи відповідь у вигляді впорядкованого набору можливих відповідників введеної адреси.

7. Після отримання відповіді перевіряється її успішність: у разі помилки під час виклику сервісу або отримання некоректного стану виконання подія фіксується, а метод повертає індикацію відсутності результату.

8. Якщо зовнішній сервіс успішно обробив запит, аналізується вміст відповіді: у випадку, коли набір результатів порожній або не містить придатних даних, вважається, що відповідних координат для заданої адреси не знайдено, і формується порожній результат.

9. Із непорожнього набору результатів обирається найрелевантніший збіг

(наприклад, перший у впорядкованому списку), з якого зчитуються значення географічної широти та довготи.

10. Отримані координати передаються до підсистеми роботи з геометричними даними, де на їх основі створюється внутрішнє геопросторове представлення точки у глобальній системі координат, що використовується в межах застосунку.

11. Сформований геопросторовий об’єкт повертається як результат роботи методу й надалі може бути збережений у сховищі даних або використаний іншими підсистемами для виконання просторових розрахунків, пошуку об’єктів у радіусі та побудови маршрутів.

Блок-схема методу пошуку координат за текстовою адресою зображена на рисунку 2.2.

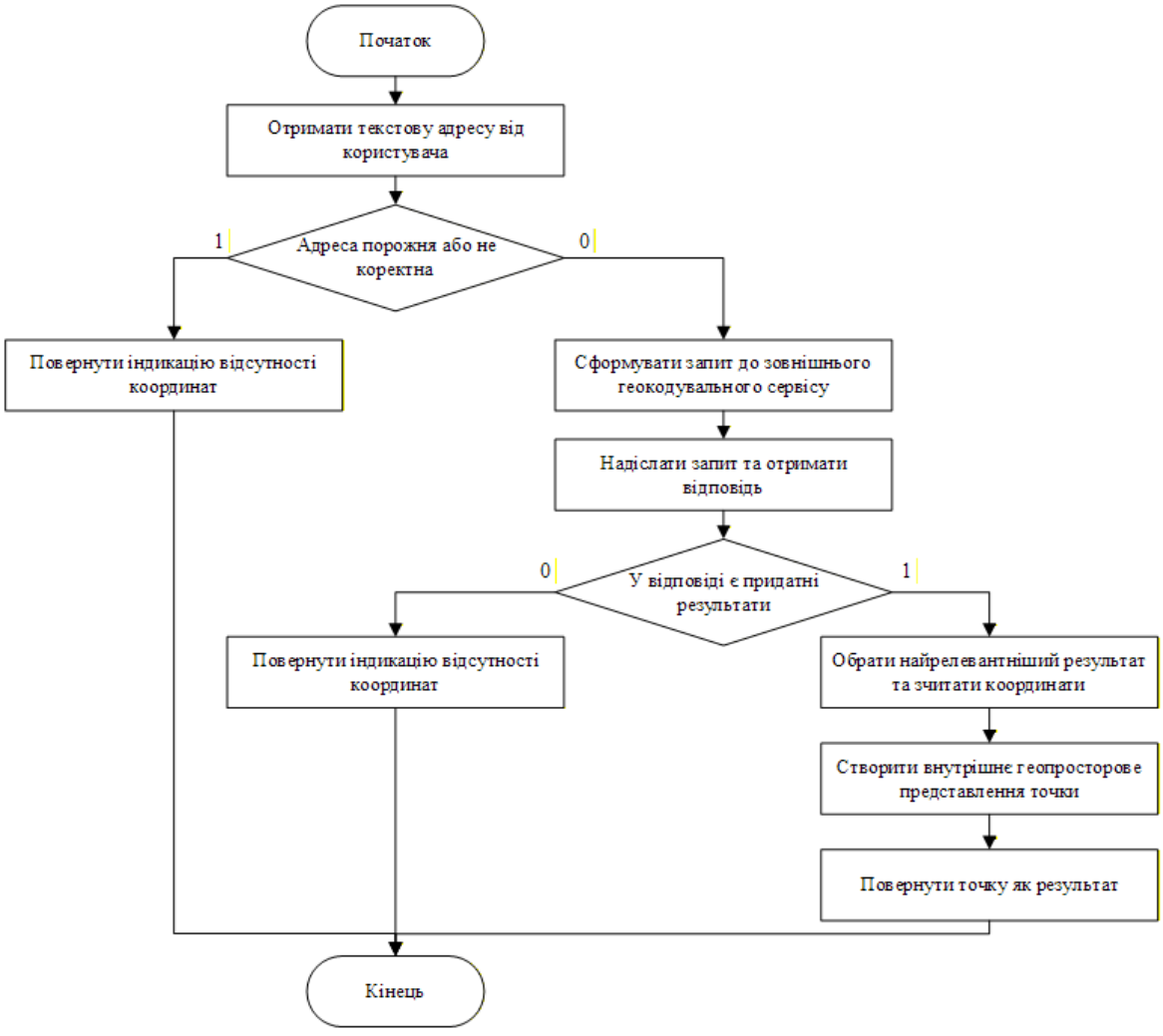

Рисунок 2.2 – Блок-схема методу пошуку координат за текстовою адресою

Діаграма послідовностей для розроблюваного методу зображена на рисунку 2.3.

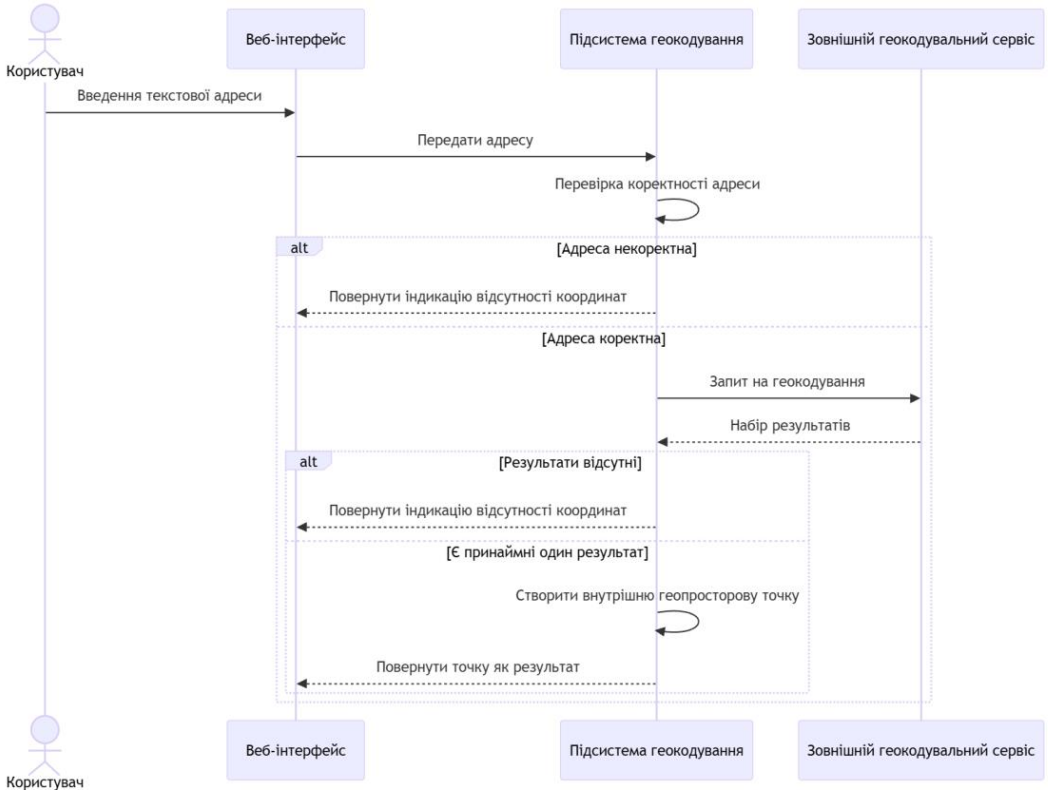

Рисунок 2.3 – Діаграма послідовностей методу пошуку координат за текстовою адресою

Повідомлення взаємодій об’єктів з діаграми послідовностей описані в таблиці 2.1.

Таблиця 2.1 – Повідомлення, якими обмінюються об’єкти під час пошуку координат за текстовою адресою

№	Відправник	Одержувач	Опис повідомлення
1	Користувач	Веб-інтерфейс	Введення текстової адреси у полі вводу (наприклад, адреси відправника чи одержувача).
2	Веб-інтерфейс	Підсистема геокодування	Передача введеної текстової адреси для подальшої обробки та пошуку координат.

Продовження таблиці 2.1

№	Відправник	Одержувач	Опис повідомлення
3	Підсистема геокодування	Підсистема геокодування	Первинна перевірка коректності адреси (порожнє значення, наявність лише пробільних символів тощо).
4	Підсистема геокодування	Веб-інтерфейс	У разі некоректної адреси – повернення індикації відсутності координат для заданого текстового опису місця.
5	Підсистема геокодування	Зовнішній геокодувальний сервіс	У разі коректної адреси – формування та надсилання запиту на геокодування для отримання можливих відповідників.
6	Зовнішній геокодувальний сервіс	Підсистема геокодування	Повернення набору результатів геокодування (може бути порожнім або містити один чи кілька варіантів).
7	Підсистема геокодування	Веб-інтерфейс	Якщо придатних результатів немає – повернення індикації відсутності координат для заданої адреси.
8	Підсистема геокодування	Підсистема геокодування	Якщо є хоча б один результат – вибір найбільш релевантного варіанта та формування внутрішнього геопросторового об’єкта точки.
9	Підсистема геокодування	Веб-інтерфейс	Повернення сформованої геопросторової точки як результату, придатного для подальших просторових операцій у системі.

2.3 Розробка методу пошуку замовлень в радіусі встановленого відхилення від маршруту

Метод пошуку замовлень в радіусі встановленого відхилення від маршруту призначений для відбору таких перевізницьких пропозицій, початкова та кінцева точки яких розташовані поблизу маршруту, заданого користувачем. Користувач задає текстові адреси початку та завершення свого шляху, а також максимально допустиме відхилення в кілометрах. На основі цих текстових описів місць система, за допомогою підсистеми геокодування, отримує їхні географічні координати, після чого звертається до зовнішньої служби для побудови автомобільного маршруту між цими точками. У відповідь система отримує геометричне представлення маршруту у вигляді послідовності опорних точок, що дискретно наближують фактичну лінію руху.

Паралельно з цим із внутрішнього сховища даних завантажується множина актуальних перевізницьких пропозицій, для кожної з яких зберігаються координати пункту відправлення та пункту призначення. Для кожної пропозиції обчислюється просторове наближення мінімальної відстані між її початковою та кінцевою точками і множиною точок маршруту, використовуючи відповідну метрику на поверхні Землі.

Пропозиція вважається такою, що пролягає «біля маршруту», якщо і точка відправлення, і точка прибуття розташовані не далі за заданий радіус відхилення від деяких точок побудованого маршруту. У результаті метод формує та повертає підмножину перевізницьких пропозицій, які потенційно можуть бути використані для перевезення посилок уздовж обраного користувачем маршруту без суттєвих додаткових об’їздів.

Алгоритм роботи методу пошуку замовлень в радіусі встановленого відхилення від маршруту можна подати у такій послідовності кроків:

1. Користувач у веб-інтерфейсі задає текстові адреси початкової та кінцевої точки свого маршруту, а також значення допустимого радіуса відхилення від маршруту в кілометрах.

2. Заданий набір параметрів передається на серверну частину системи, де

запускається метод пошуку перевізницьких пропозицій поблизу маршруту.

3. На початку роботи методу перевіряється коректність радіуса: якщо значення є неприпустимим (наприклад, меншим або рівним нулю), формування результату припиняється з повідомленням про помилку.

4. Для побудови маршруту між заданими адресами система за допомогою підсистеми геокодування отримує координати початкової та кінцевої точок і звертається до зовнішньої маршрутизувальної служби, яка повертає геометрію маршруту у вигляді послідовності опорних точок.

5. Паралельно, із внутрішнього сховища даних завантажується множина актуальних перевізницьких пропозицій, для кожної з яких відомі координати пункту відправлення та пункту призначення, а також супровідні атрибути.

6. Для кожної пропозиції послідовно аналізуються точки побудованого маршруту: за допомогою геодезичної відстані (з урахуванням кривизни поверхні Землі) обчислюється відстань від кожної точки маршруту до точки відправлення та до точки прибуття пропозиції.

7. Якщо хоча б для однієї точки маршруту відстань до пункту відправлення не перевищує заданого радіуса, пропозиція позначається як така, що розташована поблизу маршруту в точці відправлення; аналогічно перевіряється близькість пункту призначення.

8. У випадку, коли і пункт відправлення, і пункт прибуття пропозиції виявляються розташованими в межах допустимого радіуса від деяких точок маршруту, така пропозиція включається до множини релевантних результатів, і подальший аналіз для неї може бути завершено достроково.

9. Після обробки всіх доступних пропозицій формується підмножина оферів пропозицій, що відповідають умові близькості до маршруту на всьому відрізку перевезення, і цей список повертається на рівень веб-інтерфейсу для відображення користувачеві на карті або у вигляді табличного переліку доступних варіантів доставки.

Блок-схема методу пошуку замовлень в радіусі встановленого відхилення від маршруту зображена на рисунку 2.4.

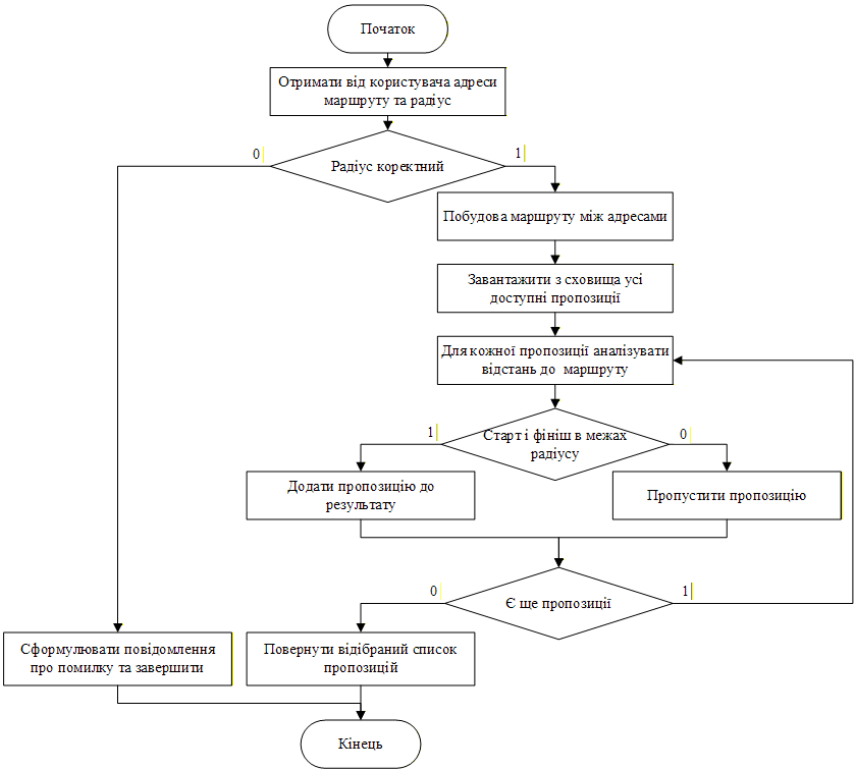

Рисунок 2.4 – Блок-схема методу пошуку замовлень в радіусі встановленого відхилення від маршруту

Діаграму послідовностей для методу зображено на рисунку 2.5.

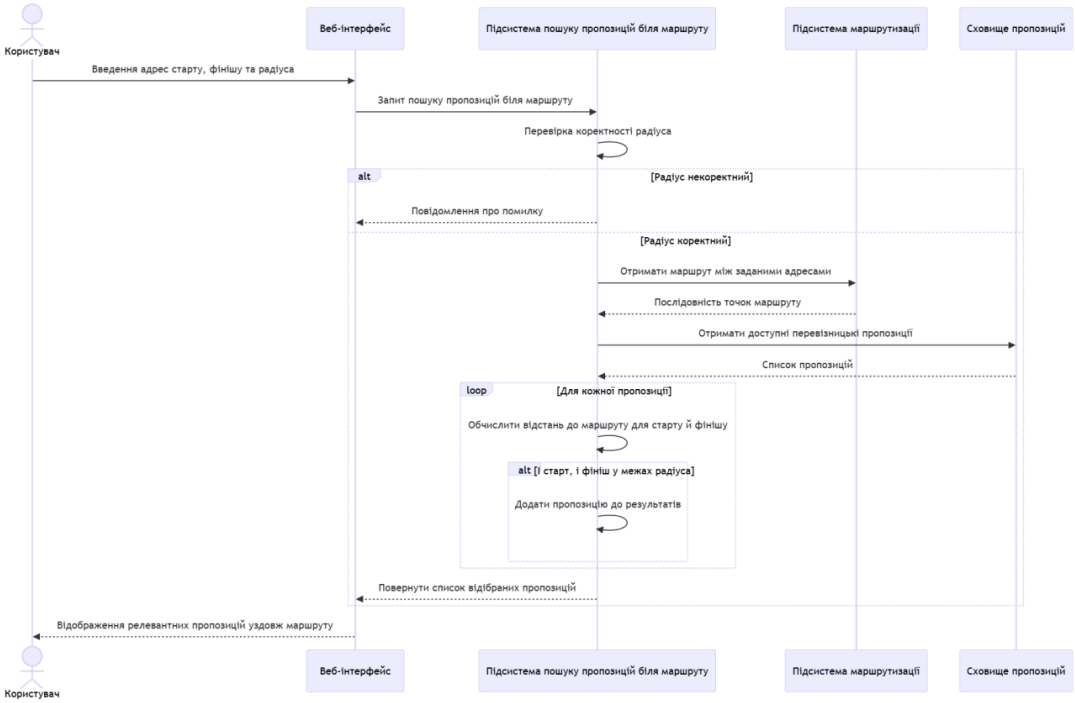

Рисунок 2.5 – Діаграма послідовностей методу пошуку замовлень в радіусі встановленого відхилення від маршруту

Повідомлення взаємодій об’єктів з діаграми послідовностей описані в таблиці 2.2.

Таблиця 2.2 – Повідомлення, якими обмінюються об’єкти під час пошуку замовлень біля маршруту

№	Відправник	Одержувач	Опис повідомлення
1	Користувач	Веб-інтерфейс	Введення адрес початку, кінця маршруту та значення радіуса відхилення.
2	Веб-інтерфейс	Підсистема пошуку	Передача запиту на пошук перевізницьких пропозицій поблизу заданого маршруту.
3	Підсистема пошуку	Підсистема пошуку	Перевірка коректності значення радіуса (діапазон, ненульове значення тощо).
4	Підсистема пошуку	Веб-інтерфейс	У разі некоректного радіуса – повернення повідомлення про помилку та відмова від подальшої обробки.
5	Підсистема пошуку	Підсистема маршрутизації	У разі коректного радіуса – запит на побудову маршруту між заданими адресами.
6	Підсистема маршрутизації	Підсистема пошуку	Повернення геометрії маршруту у вигляді послідовності опорних точок.
7	Підсистема пошуку	Сховище пропозицій	Запит на отримання множини доступних перевізницьких пропозицій.
8	Сховище пропозицій	Підсистема пошуку	Повернення списку пропозицій з координатами точок відправлення та прибуття.

Продовження таблиці 2.2

№	Відправник	Одержувач	Опис повідомлення
9	Підсистема пошуку	Підсистема пошуку	Послідовний аналіз кожної пропозиції: обчислення відстані від її старту й фінішу до точок маршруту.
10	Підсистема пошуку	Веб-інтерфейс	Повернення сформованого списку пропозицій, що лежать у межах заданого радіуса від маршруту.
11	Веб-інтерфейс	Користувач	Відображення відібраних пропозицій уздовж маршруту (на карті або в табличному вигляді).

2.4 Розробка методу формування рейтингу користувачів системи

Метод формування рейтингу користувачів системи призначений для створення, перевірки та збереження відгуку одного користувача про іншого на основі виконаного або завершеного замовлення. Його основною метою є забезпечення прозорої та справедливої оцінки взаємодій усередині сервісу, підвищення довіри між учасниками та покращення якості наданих послуг.

У процесі оцінювання користувач, який залишає відгук, обирає одержувача оцінки (наприклад, перевізника чи відправника), задає значення рейтингу за п’ятибальною шкалою та, за потреби, додає текстовий коментар. Після підтвердження форми ці дані передаються на серверну частину системи, де відбувається їх подальша обробка.

На сервері система ідентифікує особу автора відгуку, визначає користувача, якого оцінюють, а також, за наявності, встановлює відповідне замовлення, у межах якого відбулась взаємодія між сторонами. Подальша логіка роботи методу включає низку послідовних перевірок, спрямованих на забезпечення коректності, обґрунтованості та унікальності оцінки.

Перевіряється належність значення рейтингу допустимому діапазону, забороняється виставлення оцінки самому собі, а у випадку прив’язки до

конкретного замовлення контролюється, що автор відгуку справді є його учасником.

Додатково здійснюється перевірка на відсутність дубліката, тобто система не дозволяє створити більше одного відгуку від того самого користувача для одного й того самого замовлення.

Лише після успішного проходження всіх контролів формується запис відгуку з посиланнями на автора, одержувача, пов’язане замовлення, числову оцінку, коментар і мітку часу, який зберігається у сховищі даних.

Таким чином, метод забезпечує одноразове та верифіковане виставлення рейтингу для кожної завершеної взаємодії між користувачами системи.

Алгоритм роботи методу формування рейтингу користувачів системи можна подати у такій послідовності кроків:

1. Користувач на сторінці історії замовлень обирає, кого саме він хоче оцінити, задає числову оцінку в діапазоні від 1 до 5, за потреби додає текстовий коментар та підтверджує відправлення форми.

2. Веб-інтерфейс передає введені дані на серверну частину системи, де ідентифікується автор відгуку, користувач, якого оцінюють, а також, за наявності, пов’язане замовлення.

3. На початковому етапі обробки виконується перевірка коректності значення рейтингу: якщо воно виходить за межі допустимого діапазону, формування відгуку припиняється з повідомленням про помилку.

4. Далі система завантажує з внутрішнього сховища даних облікові записи автора відгуку та користувача, якого оцінюють; у разі відсутності будь-якого з них створення відгуку неможливе.

5. Виконується контроль на самооцінювання: якщо автор і одержувач відгуку збігаються, система блокує спробу виставити рейтинг самому собі.

6. Якщо відгук прив’язується до конкретного замовлення, система перевіряє наявність відповідного запису, а також те, що автор відгуку є учасником цього замовлення (наприклад, виступає перевізником або відправником посилки).

7. Додатково здійснюється перевірка на відсутність дубліката: система не допускає створення більше ніж одного відгуку від одного й того самого користувача для одного замовлення.

8. Після успішного проходження всіх перевірок формується новий запис відгуку, який містить посилання на автора, одержувача, за потреби – на замовлення, а також числову оцінку, коментар і мітку часу створення.

9. Сформований відгук зберігається у сховищі даних, після чого користувач повертається на сторінку історії замовлень, де результати оцінювання можуть бути враховані під час відображення інформації про рейтинги користувачів системи.

Блок-схему для розроблюваного методу формування рейтингу користувачів системи зображено на рисунку 2.6.

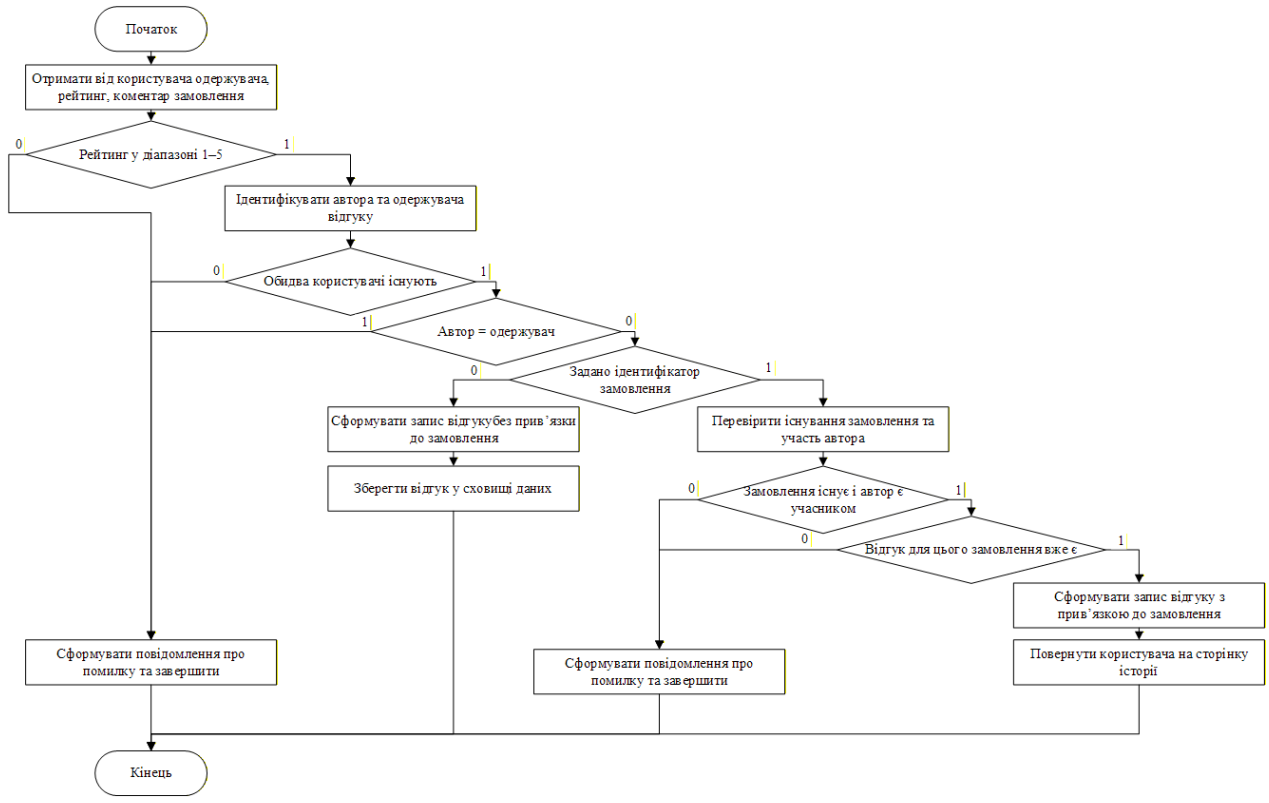

Рисунок 2.6 – Блок-схема методу формування рейтингу користувачів системи

Діаграму послідовностей для методу формування рейтингу користувачів системи зображено на рисунку 2.7.

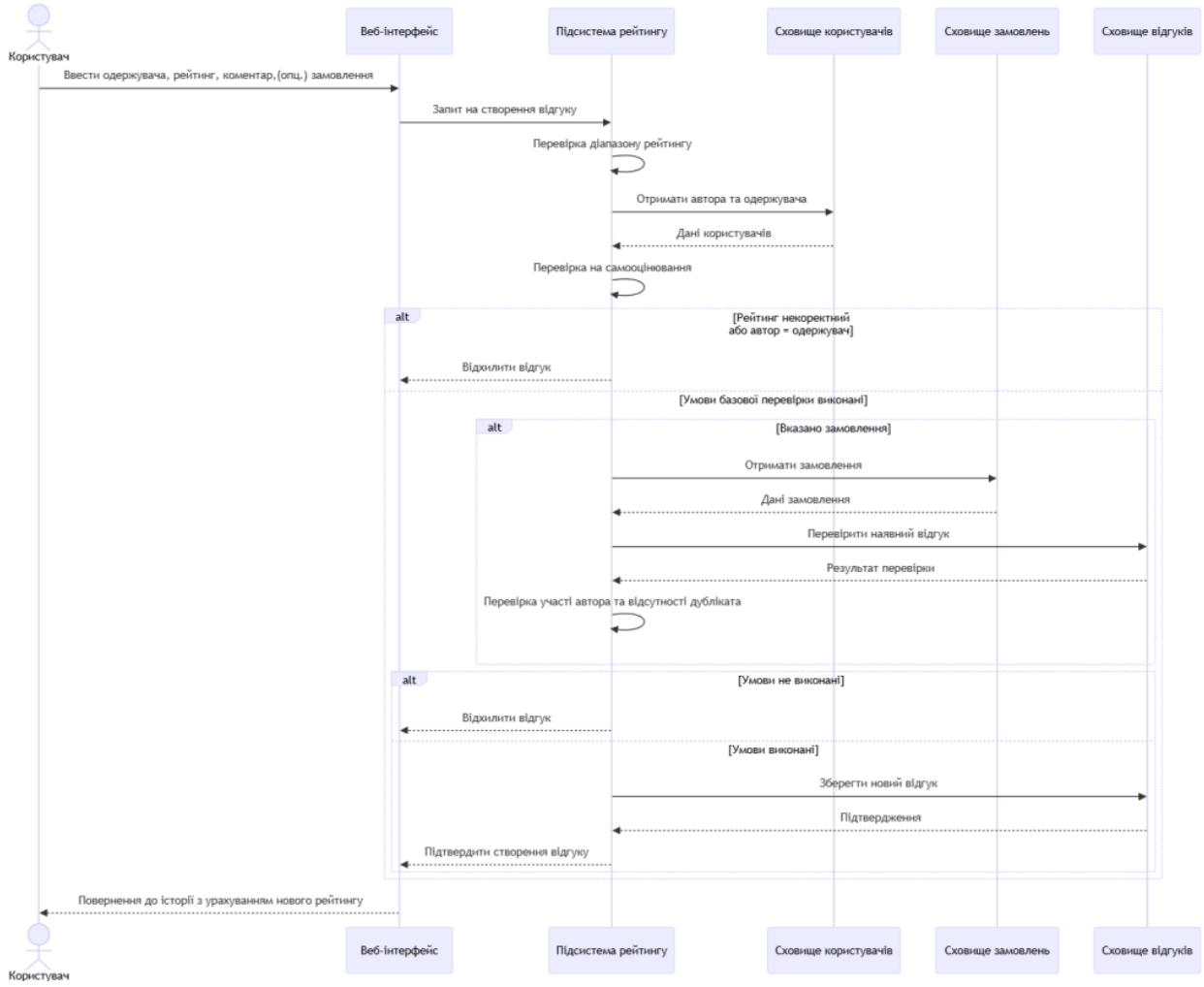

Рисунок 2.7 – Діаграма послідовностей методу формування рейтингу користувачів системи

Повідомлення взаємодій об’єктів з діаграми послідовностей описані в таблиці 2.3.

Таблиця 2.3 – Повідомлення, якими обмінюються об’єкти під час формування рейтингу користувачів системи

№	Відправник	Одержувач	Опис повідомлення
1	Користувач	Веб-інтерфейс	Введення одержувача оцінки, числового рейтингу, коментаря та, за потреби, іденти-фікатора замовлення.
2	Веб-інтерфейс	Підсистема рейтингу	Передача запиту на створення відгуку з усіма введеними параметрами.

Продовження таблиці 2.3

№	Відправник	Одержувач	Опис повідомлення
3	Підсистема рейтингу	Підсистема рейтингу	Перевірка коректності значення рейтингу (належність допустимому діапазону).
4	Підсистема рейтингу	Сховище користувачів	Запит на отримання даних автора відгуку та користувача, якого оцінюють.
5	Сховище користувачів	Підсистема рейтингу	Повернення даних обох користувачів або індикації їх відсутності.
6	Підсистема рейтингу	Підсистема рейтингу	Перевірка на самооцінювання та базові умови можливості створення відгуку.
7	Підсистема рейтингу	Сховище замовлень	У разі вказаного замовлення – отримання даних про відповідне бронювання.
8	Сховище замовлень	Підсистема рейтингу	Повернення інформації про замовлення для подальших перевірок.
9	Підсистема рейтингу	Сховище відгуків	Перевірка, чи існує вже відгук для даного поєднання автор–замовлення (контроль відсутності дубліката).
10	Сховище відгуків	Підсистема рейтингу	Повернення результату перевірки на наявність попереднього відгуку.
11	Підсистема рейтингу	Сховище відгуків	За умови виконання всіх перевірок – збереження нового відгуку у сховищі даних.
12	Підсистема рейтингу	Веб-інтерфейс	Повернення результату обробки: підтвердження створення відгуку або повідомлення про відмову.
13	Веб-інтерфейс	Користувач	Повернення на сторінку історії замовлень із урахуванням результатів формування рейтингу.

2.5 Розробка методу формування статистики профілю користувача

Метод формування статистики профілю користувача призначений для побудови узагальнених показників рейтингу та активності, які відображаються на сторінці особистого кабінету. Після переходу до профілю система ідентифікує поточного користувача, отримує його облікові дані зі сховища та ініціює обчислення ключових характеристик: середнього рейтингу, загальної кількості отриманих відгуків і розподілу оцінок за шкалою від 1 до 5. Це дає змогу сформувати цілісне уявлення про репутацію користувача в системі та використовувати ці дані для візуалізації у вигляді діаграм.

Паралельно формується статистика за операціями, у яких користувач брав участь як відправник і як перевізник. Підраховується загальна кількість створених відправлень, кількість успішно доставлених і скасованих посилок, а також кількість бронювань, завершених і анульованих у ролі перевізника. Усі отримані показники об’єднуються в єдиний набір атрибутів, який передається до шаблону профілю та використовується для відображення зведеної статистики та побудови графічних елементів інтерфейсу.

Алгоритм роботи методу формування статистики профілю користувача можна подати у такій послідовності кроків:

1. Користувач у веб-інтерфейсі переходить на сторінку профілю, що ініціює запит до серверної частини системи.

2. Система ідентифікує поточного користувача за даними аутентифікації, завантажує його обліковий запис зі сховища даних і готує базову інформацію для відображення у формі редагування профілю.

3. Далі запускається допоміжний метод формування статистики, який отримує ідентифікатор користувача та використовує його як ключ у всіх подальших обчисленнях.

4. Через підсистему керування рейтингами обчислюється середній рейтинг користувача, загальна кількість отриманих відгуків і кількість оцінок кожного значення за шкалою 1–5, що дозволяє сформувати розподіл рейтингу.

5. Паралельно за допомогою підсистеми роботи з відправленнями

визначається загальна кількість створених посилок, кількість успішно доставлених і скасованих відправлень, у яких користувач виступав відправником.

6. Аналогічно, через підсистему роботи з доставками обчислюється загальна кількість бронювань, а також кількість завершених і скасованих доставок, у яких користувач виконував роль перевізника.

7. Усі отримані показники рейтингу та активності додаються до моделі у вигляді окремих атрибутів і передаються до шаблону профілю.

8. На основі цих атрибутів система формує сторінку профілю, де статистика користувача може бути відображена як у текстовому вигляді, так і у формі графічних елементів (діаграм).

Блок-схему для розроблюваного методу зображено на рисунку 2.8.

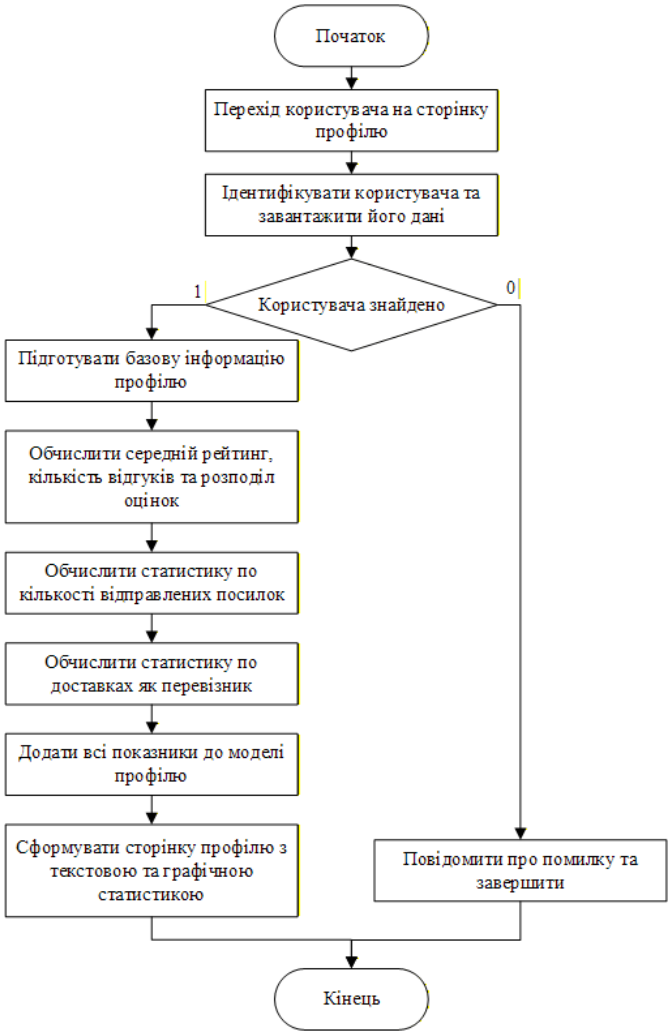

Рисунок 2.8 – Блок-схема методу формування статистики профілю користувача

Діаграму послідовностей для розроблюваного методу формування статистики профілю користувача зображено на рисунку 2.9.

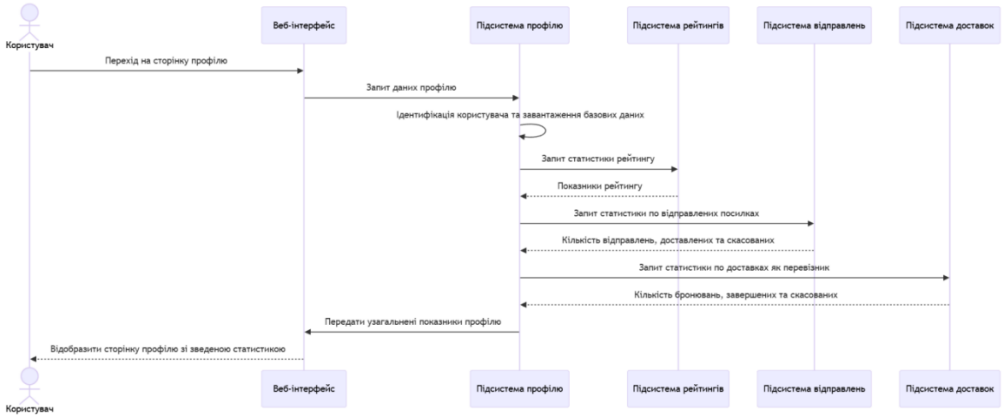

Рисунок 2.9 – Діаграма послідовностей методу формування статистики профілю користувача

Повідомлення взаємодій об’єктів з діаграми послідовностей описані в таблиці 2.4.

Таблиця 2.4 – Повідомлення, якими обмінюються об’єкти під час формування статистики профілю користувача

№	Відправник	Одержувач	Опис повідомлення
1	Користувач	Веб-інтерфейс	Перехід на сторінку профілю, ініціація запиту на завантаження даних профілю.
2	Веб-інтерфейс	Підсистема профілю	Передача запиту на отримання даних профілю поточного користувача.
3	Підсистема профілю	Підсистема профілю	Ідентифікація користувача за даними аутентифікації та завантаження базової інформації профілю.
4	Підсистема профілю	Підсистема рейтингів	Запит статистики рейтингу: середній рейтинг, кількість відгуків та розподіл оцінок 1–5.
5	Підсистема рейтингів	Підсистема профілю	Повернення показників рейтингу користувача.

Продовження таблиці 2.4

№	Відправник	Одержувач	Опис повідомлення
6	Підсистема профілю	Підсистема відправлень	Запит статистики щодо відправлених посилок (усього, доставлені, скасовані).
7	Підсистема відправлень	Підсистема профілю	Повернення агрегованих показників по відправленнях.
8	Підсистема профілю	Підсистема доставок	Запит статистики щодо доставок, у яких користувач виступає перевізником.
9	Підсистема доставок	Підсистема профілю	Повернення кількості бронювань, завершених та скасованих доставок.
10	Підсистема профілю	Веб-інтерфейс	Передача зведених показників рейтингу та активності для відображення в профілі користувача.
11	Веб-інтерфейс	Користувач	Відображення сторінки профілю зі зведеною статистикою та, за потреби, графічними елементами.

2.6 Розробка моделі та загального алгоритму роботи системи

На рисунку 2.10 зображено модель розроблюваної системи, яка створена у вигляді UML діаграми прецедентів.

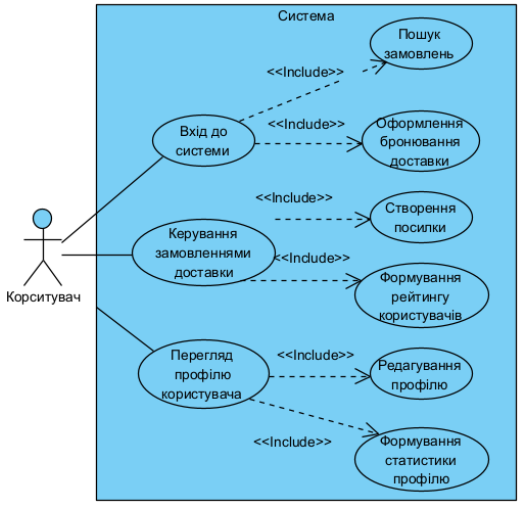

Рисунок 2.10 – Діаграма прецедентів моделі системи

Діаграма прецедентів використання системи організованої доставки посилок включає такі прецеденти: «Вхід до системи», «Керування замовленнями доставки», «Пошук замовлень», «Оформлення бронювання доставки», «Створення посилки», «Формування рейтингу користувачів», «Перегляд профілю користувача», «Редагування профілю», «Формування статистики профілю».

Базові прецеденти «Керування замовленнями доставки» та «Перегляд профілю користувача» включають відповідні допоміжні сценарії, що деталізують роботу з замовленнями, рейтингами й даними профілю.

Потік подій для прецеденту «Керування замовленнями доставки».

Актор – користувач.

Мета – створити посилку, підібрати відповідну пропозицію доставки, оформити бронювання та, за потреби, зафіксувати рейтинг іншого учасника угоди.

Передумови – користувач виконав вхід до системи; обліковий запис активний.

Результат – у системі створені або оновлені записи про посилки, бронювання та відгуки, що відображають актуальний стан замовлень доставки.

Основний потік:

1. Користувач, після успішного входу до системи, переходить до розділу «Керування замовленнями доставки».

2. Система відображає перелік наявних посилок та пов’язаних із ними бронювань користувача.

3. Для створення нового відправлення запускається включений прецедент «Створення посилки»: користувач задає адреси відправлення й доставки, вагу та опис посилки, а система перевіряє коректність даних і зберігає новий запис.

4. Для пошуку можливостей доставки користувач ініціює включений прецедент «Пошук замовлень», задаючи критерії (напрямок маршруту, дату, допустиме відхилення чи інші фільтри). Система підбирає релевантні перевізницькі пропозиції та відображає їх у вигляді списку.

5. Обравши потрібну пропозицію, користувач запускає прецедент «Оформлення бронювання доставки»: система пов’язує вибрану посилку з обраним перевізником, фіксує погоджену ціну та умови, створює запис бронювання з відповідним статусом.

6. Після завершення фактичної доставки система оновлює статуси посилки та бронювання, і користувач може скористатися прецедентом «Формування рейтингу користувачів» – обрати замовлення, вказати контрагента, задати числову оцінку та, за бажанням, текстовий коментар.

7. Система виконує перевірку можливості оцінювання (один відгук для пари «користувач–замовлення», належність користувача до учасників угоди) та зберігає новий відгук у базі даних.

8. Оновлена інформація про посилки, бронювання й рейтинги стає доступною для подальшого перегляду та аналізу в інших прецедентах системи.

Альтернативні потоки:

Е-1: некоректні дані посилки. Якщо обов’язкові параметри посилки (адреси, вага тощо) заповнені некоректно або відсутні, система відхиляє створення посилки, відображає повідомлення про помилку та пропонує користувачу виправити дані.

Е-2: відсутні підходящі пропозиції доставки. Якщо за заданими критеріями пошуку система не знаходить жодного варіанту доставки, користувач отримує відповідне повідомлення та може змінити параметри пошуку або відкласти бронювання.

Е-3: неможливість оформити бронювання. Якщо обрана пропозиція стала недоступною (наприклад, уже заброньована іншим користувачем або перевізник змінив умови), система не створює бронювання і повідомляє про причину відмови.

Е-4: відгук уже існує. Якщо користувач намагається залишити повторний відгук для того самого замовлення, система блокує операцію та інформує, що рейтинг за цим бронюванням уже було виставлено.

Потік подій для прецеденту «Перегляд профілю користувача».

Актор – користувач.

Мета – переглянути та, за потреби, оновити особисті дані, а також ознайомитися зі зведеною статистикою власної активності й рейтингу в системі.

Передумови – користувач успішно виконав вхід до системи.

Результат – користувач отримує актуальну інформацію про свій профіль, може змінити контактні дані та бачить узагальнені показники рейтингу й участі в доставках.

Основний потік:

1. Користувач переходить до розділу «Перегляд профілю користувача» через навігаційне меню інтерфейсу.

2. Система завантажує з бази даних основні дані профілю (ім’я, прізвище, електронну адресу, номер телефону та інші реквізити) і відображає їх у вигляді форми.

3. Для зміни персональних даних ініціюється включений прецедент «Редагування профілю»: користувач оновлює потрібні поля, після чого система перевіряє коректність введеної інформації та зберігає зміни.

4. Паралельно або за запитом користувача система виконує прецедент «Формування статистики профілю» – агрегує дані про залишені й отримані відгуки, створені посилки, виконані й скасовані доставки, обчислює середній рейтинг і формує показники для подальшої візуалізації.

5. Система відображає на сторінці профілю як оновлені особисті дані, так і зведену статистику у вигляді числових показників та, за потреби, графічних елементів (діаграм).

6. Користувач аналізує отриману інформацію, за потреби коригує дані профілю або повертається до інших розділів системи.

Альтернативні потоки:

Е-1: некоректні дані профілю. Якщо при редагуванні профілю вказано некоректні або неповні дані (наприклад, неправильний формат номера телефону чи обов’язкові поля залишено порожніми), система не зберігає зміни та відображає користувачу повідомлення з переліком помилок.

E-2: тимчасова недоступність статистики. У разі недоступності окремих підсистем (наприклад, сервісу зберігання відгуків чи історії бронювань) система може відобразити базові дані профілю без статистики, супроводивши це інформаційним повідомленням про часткове обмеження функціональності.

Для візуалізації загального алгоритму роботи системи було створено діаграму станів загального алгоритму роботи системи, яка демонструє усю можливу взаємодію користувача із системою.

Діаграма наведена на рисунку 2.11.

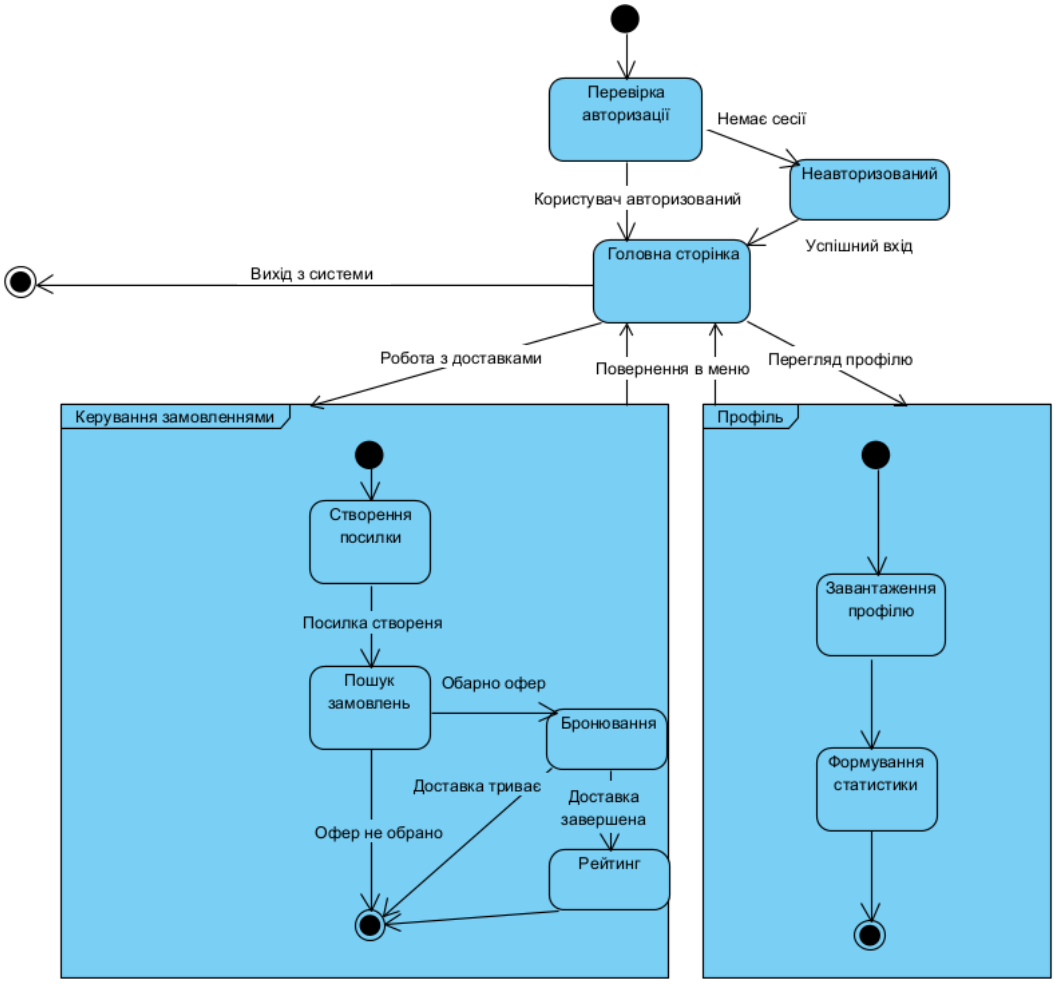

Рисунок 2.11 – Діаграма станів загального алгоритму роботи системи

2.7 Розробка структури таблиць бази даних

Розробка структури таблиць бази даних є ключовим етапом проєктування системи, оскільки саме від неї залежить цілісність, узгодженість та доступність

даних під час роботи веб-застосунку. У межах даної системи необхідно зберігати інформацію про користувачів, їхні ролі, посилки, бронювання перевезень, рейтинги та відгуки, а також координати об’єктів для просторових операцій. Непродумана або надмірно спрощена схема може призвести до дублювання даних, складності в обробці запитів, помилок при оновленні інформації та обмежень у розвитку функціональності.

Тому на цьому етапі виконується логічне структурування предметної області у вигляді взаємопов’язаних таблиць із чітко визначеними ключами, зовнішніми зв’язками та довідниковими сутностями для ролей і статусів. Це дозволяє забезпечити нормалізацію даних, підтримувати однозначність інтерпретації кожного запису та створити надійну основу для реалізації алгоритмів пошуку, маршрутизації, бронювання та формування рейтингу користувачів системи.

Схеми таблиць бази даних зображено на рисунку 2.12.

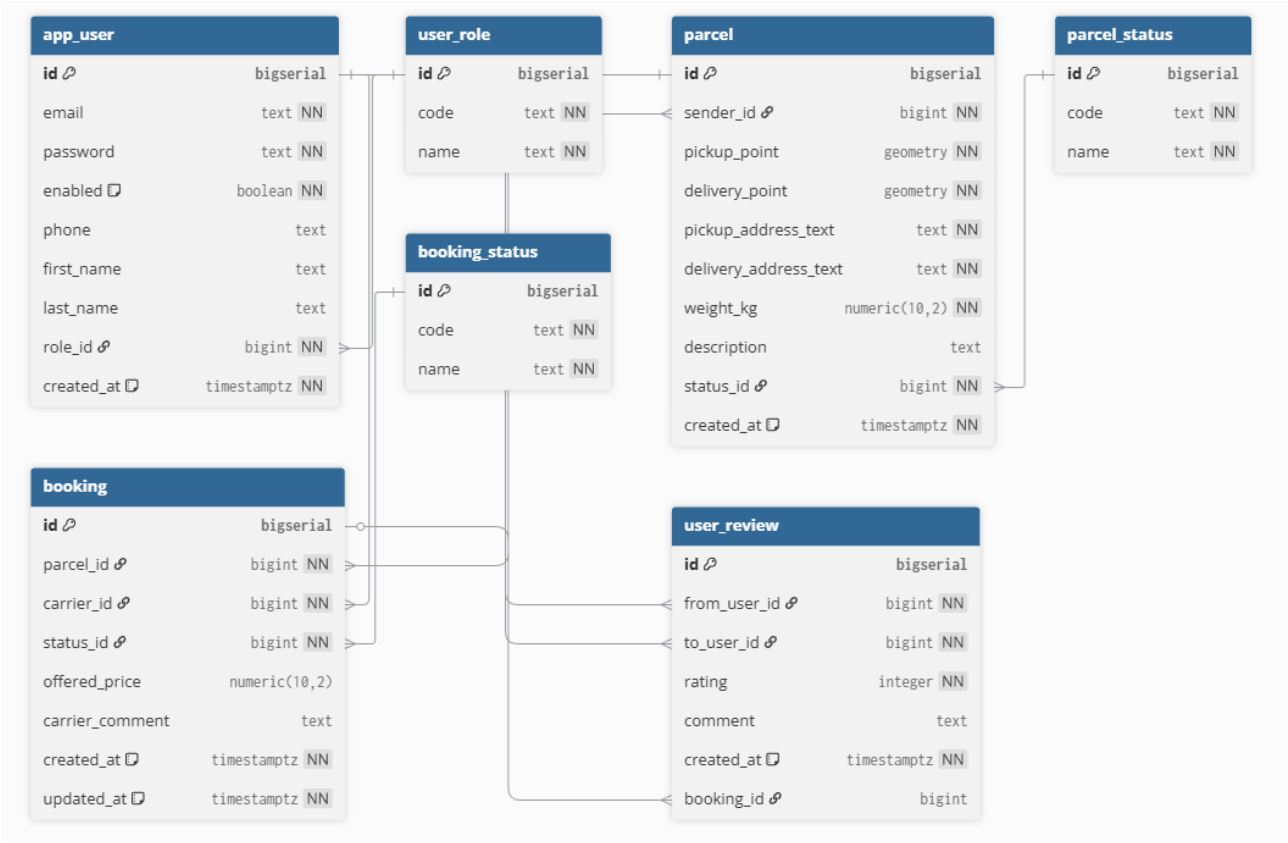

Рисунок 2.12 – Загальна схема таблиць та їх зв’язків

Таблиця app_user містить інформацію про зареєстрованих користувачів системи.

Вона містить такі поля:

– id – унікальний ідентифікатор користувача;

– email – електронна адреса, що використовується для входу в систему та ідентифікації користувача;

– password – хешований пароль користувача;

– enabled – ознака активності облікового запису (дозволений/заблокований доступ);

– phone – контактний номер телефону користувача;

– first_name – ім’я користувача;

– last_name – прізвище користувача;

– role_id – зовнішній ключ на таблицю ролей, що визначає права та статус користувача в системі;

– created_at – дата й час створення облікового запису.

Таблиця user_role містить довідникову інформацію про ролі користувачів системи.

Вона містить такі поля:

– id – унікальний ідентифікатор ролі;

– code – символьний код ролі (наприклад, ADMIN, USER, CARRIER), який використовується в логіці застосунку;

– name – людиночитна назва ролі для відображення в інтерфейсі та звітах.

Таблиця parcel містить інформацію про посилки, які користувачі створюють для відправлення.

Вона містить такі поля:

– id – унікальний ідентифікатор посилки;

– sender_id – зовнішній ключ на таблицю користувачів, що вказує на відправника посилки;

– pickup_point – геометрична точка місця забору посилки в географічній системі координат;

– delivery_point – геометрична точка місця доставки посилки;

– pickup_address_text – текстове представлення адреси забору посилки;

– delivery_address_text – текстове представлення адреси доставки посилки;

– weight_kg – вага посилки в кілограмах;

– description – текстовий опис вмісту або особливостей посилки;

– status_id – зовнішній ключ на таблицю статусів посилок, що характеризує поточний стан відправлення;

– created_at – дата й час створення запису про посилку.

Таблиця parcel_status містить довідникову інформацію про можливі статуси посилок.

Вона містить такі поля:

– id – унікальний ідентифікатор статусу посилки;

– code – машинно-орієнтований код статусу (наприклад, NEW, DELIVERED, CANCELED);

– name – описова назва статусу для відображення користувачу.

Таблиця booking містить інформацію про бронювання перевезення посилок між відправником і перевізником.

Вона містить такі поля:

– id – унікальний ідентифікатор бронювання;

– parcel_id – зовнішній ключ на таблицю посилок, що вказує, яку саме посилку бронюють для перевезення;

– carrier_id – зовнішній ключ на таблицю користувачів, що визначає перевізника;

– status_id – зовнішній ключ на таблицю статусів бронювання, який описує поточний стан угоди;

– offered_price – запропонована ціна перевезення (вартість послуги);

– carrier_comment – коментар перевізника щодо умов або деталей перевезення;

– created_at – дата й час створення запису про бронювання;

– updated_at – дата й час останнього оновлення стану або параметрів бронювання.

Таблиця booking_status містить довідникову інформацію про можливі статуси бронювань.

Вона містить такі поля:

– id – унікальний ідентифікатор статусу бронювання;

– code – символьний код статусу (наприклад, PENDING, COMPLETED, CANCELED);

– name – описова назва статусу для відображення в інтерфейсі та звітності.

Таблиця user_review містить інформацію про відгуки та рейтинги, які користувачі залишають один про одного. Вона містить такі поля:

– id – унікальний ідентифікатор відгуку;

– from_user_id – зовнішній ключ на таблицю користувачів, що позначає автора відгуку;

– to_user_id – зовнішній ключ на таблицю користувачів, якого оцінюють;

– rating – числове значення рейтингу за фіксованою шкалою (наприклад, від 1 до 5);

– comment – текстовий коментар, який деталізує оцінку або описує досвід взаємодії;

– created_at – дата й час створення відгуку;

– booking_id – зовнішній ключ на таблицю бронювань, що (за наявності) вказує замовлення, у межах якого сформовано відгук.

Створені таблиці відіграють роль фундаменту системи, забезпечуючи шар бізнес-логіки, який дозволяє оптимально працювати з даними. На їх основі формується цілісна інфраструктура, яка гарантує узгодженість прикладної логіки, надійну роботу програми та забезпечує передумови для подальшого розширення функціональних можливостей.

2.8 Висновки

У другому розділі магістерської кваліфікаційної роботи було розроблено архітектуру системи та прийнято рішення застосувати архітектурний шаблон MVC, який забезпечує модульну побудову системи, її масштабованість і спрощує подальший супровід та розвиток.

Було розроблено й описано метод пошуку координат за текстовою адресою, метод пошуку замовлень в радіусі встановленого відхилення від маршруту, метод формування рейтингу користувачів системи, а також метод формування статистики профілю користувача, що надає весь необхідний функціонал для керування посилками користувачеві

Розроблено модель системи, її діаграму варіантів використання, блок-схему загального алгоритму роботи системи та структури таблиць бази даних. У результаті було сформовано міцний програмний базис, який слугує основою всієї функціональності системи та гарантує її правильне й стабільне функціонування.

3 РОЗРОБКА ПРОГРАМНИХ ЗАСОБІВ СИСТЕМИ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації вебсистеми

Необхідно обрати мову програмування для створюваної вебсистеми. Так як за архітектурою було обрано клієнт-серверну, потрібно обрати рішення як для розробки серверної частини, так і для клієнтської.

Спочатку розглянемо можливі варіанти серверних мов програмування, серед них:

Java [12] – це об'єктно-орієнтована мова програмування з відкритим вихідним кодом, розроблена Oracle. Хоча вона була створена для програмування різних типів застосунків, Java широко використовується для розробки корпоративних веб-застосунків.

Мова забезпечує портативність коду – написане один раз можна запускати на різних системах без змін. Об’єктно-орієнтована модель дає чіткі абстракції та повторне використання. Висока продуктивність у поєднанні з доброю масштабованістю дозволяє впевнено тримати великі навантаження. Розвинена екосистема бібліотек і фреймворків пришвидшує розробку, а вбудована підтримка багатопоточності спрощує паралельне виконання задач.

Попри сильні сторони, Java має й недоліки: високий поріг входу та відчутну складність екосистеми; підвищені вимоги до ресурсів – як оперативної пам’яті, так і процесора; громіздкі процеси збирання й розгортання, що додають інженерних накладних витрат; порівняно повільний запуск застосунків, особливо під час «холодного» старту, але це нівелюється продуктивністю після запуску; а також обмежені можливості у функційному підході порівняно з мовами, де він є центральним.

C# [13] – це об'єктноорієнтована й багатопарадигмальна мова для створення сайтів, програм та ігор. Попри універсальне призначення й здатність розв’язувати широкий спектр задач, її особливо часто використовують у веб-розробці для створення веб-застосунків на платформі .NET.

Мова вирізняється суворою статичною типізацією, що підвищує надійність і передбачуваність коду, а зріла й інтуїтивна об’єктно-орієнтована модель спрощує проєктування архітектури. Вбудовані можливості функційного підходу разом із LINQ дають змогу лаконічно й декларативно працювати з даними.

Розгалужена екосистема бібліотек, інструментів і фреймворків пришвидшує розробку та спрощує підтримку. До того ж тісна інтеграція з Microsoft Visual Studio – одним із найпотужніших середовищ розробки, забезпечує зручне налагодження, рефакторинг і профілювання.

Попри сильні сторони, є й обмеження: кросплатформеність історично була неідеальною (хоча ситуація помітно покращилась із .NET Core/.NET), окремі інструменти екосистеми Microsoft можуть коштувати недешево, тісна прив’язка до платформи .NET інколи ускладнює початкове налаштування та розгортання, а на частині ринків C# поступається за популярністю Java чи JavaScript, що впливає на доступність фахівців і готових рішень.

PHP [14] – це популярна скриптова мова програмування на стороні сервера з відкритим кодом, розроблена спеціально для вебу. Її широко використовують для створення динамічних сайтів і веб-застосунків, а тісна інтеграція дозволяє без зайвих зусиль вбудовувати PHP у HTML-розмітку та поєднувати його з CSS і JavaScript.

Кросплатформені можливості роблять розгортання передбачуваним у різних середовищах, а величезна спільнота підкріплює це готовими бібліотеками та документацією. Простий синтаксис допомагає швидко переходити від ідеї до робочого коду, при цьому сучасні версії мови й двигунів дають гідну продуктивність. Підтримка популярних баз даних забезпечує гнучкість у проєктуванні сховищ.

У довгостроковій перспективі проявляються недоліки: динамічна типізація підвищує ймовірність латентних дефектів, а часткова реалізація функціональних підходів інколи знижує виразність і елегантність архітектурних конструкцій. Супровід, масштабування та еволюція великих систем

ускладнюються, тоді як рівень захищеності значною мірою визначається дисципліною команди й послідовним дотриманням практик безпечного кодування.

Go [15] – компільована, статично типізована мова програмування з відкритим вихідним кодом. Go використовують для створення серверів, веб-застосунків та інших програм з великим навантаженням, де важлива висока продуктивність, ефективність і простота в обслуговуванні.

Мову Go вирізняє поєднання короткого циклу компіляції та виконання з економним використанням пам’яті й апаратних ресурсів. Засоби конкурентності інтегровано на рівні мови, що спрощує конструювання паралельних обчислювальних процесів і знижує накладні витрати на синхронізацію.

Статична типізація підвищує надійність програмних артефактів і покращує їх читабельність, а розгалужена стандартна бібліотека разом з активною спільнотою забезпечують інструментальну підтримку для більшості типових завдань розробки.

Екосистема інструментів і бібліотек залишається відносно молодою, що звужує вибір готових компонентів для спеціалізованих завдань і підвищує витрати на інтеграцію. Підтримку узагальнених типів запроваджено лише у версії 1.18; відповідні підходи та найкращі практики ще консолідуються, що може ускладнювати проєктування узагальнених структур і алгоритмів. Класичне об’єктно-орієнтоване моделювання реалізоване опосередковано (переважно через інтерфейси та вбудовування), тому перенесення усталених ООП-патернів вимагає додаткової адаптації.

Додатковим фактором є обмежені засоби функціонального програмування, які зменшують виразність під час побудови складних композицій перетворень даних.

В таблиці 3.1 представлено порівняння мов програмування за різними критеріями, що дозволить обрати мову програмування, яка найкраще підійде для розробки вебсистеми.

Таблиця 3.1 – Порівняння мов програмування.

Критерій/Мова	Java	C#	PHP	Go
Функціональність	5	4	3	3
Підтримка ООП	5	5	2	3
Активна спільнота	5	4	4	4
Розширюваність	5	4	3	3
Простота використання	3	4	4	4
Швидкодія	5	4	5	3

З таблиці видно, що Java є лідером в даному порівнянні, тому саме її буде обрано в якості серверної мови програмування для розроблюваної вебсистеми.

Сучасна розробка вебсистем на Java давно вийшла за межі ручної роботи з сервлетами. Технологічний стек еволюціонував від низькорівневих API керування запитами та відповідями до високорівневих фреймворків, які інкапсулюють рутинні аспекти життєвого циклу застосунку, надають готові механізми інверсії керування, конфігурації, безпеки, доступу до даних, валідації, спостережуваності та тестування. Використання фреймворку перетворює створення серверної частини з набору технік «ручного складання» на системну інженерію з передбачуваною структурою, зрілою екосистемою бібліотек і зрозумілими практиками промислової експлуатації. Найпопулярнішими серед таких засобів для Java сьогодні є Spring Boot, Quarkus та Micronaut.

З огляду на те, що цільовий продукт цієї роботи – вебсистема з класичною веб-орієнтованою моделлю взаємодії, у якій поєднуються MVC-підхід, серверний рендеринг HTML-шаблонів і стандартні інтеграції з базою даних, зовнішніми сервісами та компонентами безпеки, постає завдання зваженого вибору фреймворку. Кожне із згаданих рішень має сильні та слабкі сторони, а тому для коректного рішення слід проаналізувати їх архітектурну модель,

ступінь зрілості екосистеми, відповідність обраній парадигмі представлення, інтеграційні можливості, вимоги до експлуатації у контейнеризованому середовищі, а також вплив на повний життєвий цикл продукту – від швидкості розробки до супроводу у виробництві, тож варто розглянути кожен з них.

Spring Boot [16] – найпоширенішим фреймворком для створення серверних застосунків на базі платформи Spring; його основна ідея – різко зменшити «тертя» початкової конфігурації за допомогою автоналаштувань, стандартизованих «стартерів» залежностей, однорідної моделі конфігурації та глибокої інтеграції з ключовими модулями екосистеми Spring.

Типова вебсистема в межах Boot спирається на Spring MVC для класичного контролер-орієнтованого підходу, підтримує серверний рендеринг шаблонів (зокрема, через Thymeleaf), має прозору інтеграцію з Spring Data для доступу до реляційних і нереляційних сховищ, з Spring Security для автентифікації та авторизації, з Micrometer і Spring Boot Actuator для спостережуваності та експлуатаційної телеметрії. Фреймворк зберігає сумісність із величезним масивом бібліотек Java-світу та пропонує різні стилі програмування – від імперативного MVC до реактивної моделі WebFlux у випадку систем з високою конкуренцією запитів.

Переваги Spring Boot полягають у зрілості та масштабі екосистеми, що безпосередньо впливає на продуктивність команди і зниження інтеграційних ризиків. Розробник отримує усталені патерни, всеохопну документацію і практики, перевірені мільйонами розгортань; це означає передбачуваність поведінки у виробництві, багатство прикладів і відповідей на типові питання, а також простоту підбору фахівців із релевантними навичками.

Автоконфігурації забезпечують швидкий старт без ручного «зшивання» компонентів, тоді як Actuator разом з Micrometer роблять експлуатацію та моніторинг невід’ємною частиною платформи, усуваючи бар’єри між розробкою і SRE-процесами. Сумісність із Thymeleaf у межах Spring MVC дає безшовний шлях до класичного серверного рендерингу HTML і побудови MVC у чистому вигляді, що важливо для проєктів, орієнтованих на стабільну

серверну генерацію представлення без важких клієнтських SPA. Додатково слід відзначити підтримку AOT-підготовки та компіляції в нативні образи за допомогою GraalVM у Spring Boot 3-лінійці, що розширює сценарії застосування у мікросервісному та безсерверному середовищі, коли стрімкий старт і низький профіль пам’яті стають критичними.

Недоліки Spring Boot зазвичай пов’язані з вартістю «зручності за замовчуванням». Автоконфігурації, покликані прибрати рутину, іноді призводять до завеликого обсягу підключених компонентів і помітнішого споживання ресурсів у порівнянні з більш «аскетичними» фреймворками, якщо не проводити цілеспрямоване «схуднення» залежностей. Старт часу виконання у класичному JVM-режимі часто повільніший за спеціалізовані платформи, оптимізовані під миттєвий запуск, а побудова нативних образів, хоча й підтримується, вимагає додаткової уваги до сумісності бібліотек із обмеженнями нативного світу. Нарешті, глибина і різноманітність екосистеми іноді ускладнюють навчання: для послідовного застосування реактивної моделі, складних шаблонів безпеки або тонких аспектів транзакційності потрібна дисципліна й досвід.

Quarkus [17] – позиціонується як «Supersonic Subatomic Java», підкреслюючи орієнтацію на дуже швидкий старт, низьке споживання пам’яті та щільну інтеграцію з контейнеризованими середовищами і Kubernetes.

Архітектурно фреймворк робить ставку на інтенсивну обробку на етапі збірки, мінімізуючи роботу під час старту застосунку та зменшуючи обсяг відбитка в пам’яті. У веб-частині часто використовуються стек RESTEasy (класичний або реактивний), SmallRye як реалізація специфікацій MicroProfile, Vert.x як реактор подієвої моделі, а також зв’язка з Hibernate ORM і зручним шаром Panache для доступу до даних. Значна увага приділяється розробницькому досвіду: «живий» режим розробки забезпечує швидку зворотну реакцію, перезавантажуючи частини застосунку практично миттєво при зміні коду.

Переваги Quarkus стають особливо відчутними у сценаріях, де

критичними є щільність розміщення сервісів, висока еластичність у контейнерному оточенні, а також експлуатація у безсерверних платформах: короткий холодний старт, невеликі ліміти пам’яті, масштабування за подією. Орієнтація на GraalVM і нативні образи – не доповнення, а базовий сценарій; значна частина бібліотек у складі Quarkus адаптована до обмежень нативного світу. Інтеграція з Kubernetes і OpenShift, конфігурація, секрети, профілі середовищ та спостережуваність через сумісні інструменти MicroProfile створюють відчуття «хмарного фреймворку за замовчуванням».

Недоліки Quarkus проявляються там, де очікується класичне MVC із серверним рендерингом у стилі Spring MVC + Thymeleaf. Хоча Quarkus має власний шаблонізатор Qute і існують шляхи інтеграції з поширеними видами представлення, модель не є настільки природною для традиційного MVC, як у Spring. Екосистема поза межами «рідних» розширень зростає, але за спектром прикладів, зрілістю документації сторонніх інтеграцій і насиченістю відповідей вона поступається Spring-світу, що може збільшувати витрати на інтеграції у гетерогенних середовищах.

Додатковою особливістю є шар Panache, який спрощує роботу з ORM, але водночас додає специфічність, що вимагатиме огляду практик при міграції або стандартизації з іншими проєктами. Для команд, не знайомих із реактивною парадигмою, перехід до Vert.x-орієнтованих підходів може вимагати додаткового навчання.

Micronaut [18] – сформувався як фреймворк із радикальною ставкою на попереднє, компіляційне формування метаданих і залежностей: механізми інверсії керування, AOP-перехоплення, валідації та конфігурації виконуються без рефлексії, що мінімізує накладні витрати під час старту та у рантаймі. У підсумку застосунки мають дуже швидкий запуск, невеликий профіль пам’яті й високу передбачуваність поведінки з погляду продуктивності. У складі платформи є Micronaut Data для доступу до даних із компіляційним генеруванням репозиторіїв, Micronaut Security для інтеграції механізмів безпеки, а також цілісний підхід до конфігурації, профілів і середовищ.

Підтримка GraalVM і нативних образів є органічною, адже відсутність рефлексії значно спрощує шлях до native-збірок.

Переваги Micronaut особливо привабливі для мікросервісів і безсерверних функцій, де важливі швидкі холодні старти та строгий контроль використання ресурсів. Компіляційна модель зменшує клас проблем, пов’язаних із проксі-генерацією, рефлексією та неявними автоконфігураціями, полегшуючи прогнозування характеристик виконання. Структура фреймворку є стриманою і виразною: це допомагає збудувати компактні сервіси з мінімумом залежностей і прозорим шляхом у продакшн.

Недоліки Micronaut стосуються передусім ширини екосистеми та «критичної маси» напрацювань. Порівняно зі Spring і навіть із Quarkus, масив прикладів, сторонніх інтеграцій, готових «рецептів» і зрілість інституційних практик є скромнішими. Хоча існує підтримка відображення представлень, зокрема через модулі шаблонізаторів, класичний MVC із Thymeleaf не має того рівня інтегрованості, який пропонує Spring Boot у зв’язці зі Spring MVC. Для складних бізнес-сценаріїв або специфічних інтеграцій частіше доведеться розв’язувати завдання на нижчому рівні абстракції або вдаватися до вузьких бібліотек. Навчальна крива є помірною, але потреба мислити «компіляційними» артефактами та специфікою анотацій-процесорів може вимагати перебудови звичних підходів.

Проаналізувавши архітектурні моделі, екосистему, зрілість інструментів і відповідність вимогам цільової вебсистеми, було вирішено обрати Spring Boot як основний серверний фреймворк. Таке рішення зумовлене сукупністю факторів: природною підтримкою MVC-парадигми з Thymeleaf і повною сумісністю з класичним серверним рендерингом; глибиною інтеграцій із безпекою, доступом до даних і спостережуваністю; масштабом і зрілістю екосистеми, що мінімізує інтеграційні ризики та підвищує продуктивність команди; наявністю перевірених практик розгортання і супроводу у контейнеризованих і хмарних середовищах.

Таким чином, Spring Boot забезпечує потрібний баланс між

технологічною адекватністю, інженерною передбачуваністю та операційною надійністю, що робить його оптимальним вибором для реалізації серверної частини розроблюваної вебсистеми.

Враховуючи обраний перелік технологій для розробки вебсистеми – Java разом з Spring Boot для серверної частини, патерн MVC з серверним рендерингом клієнтської частини, було прийняте рішення використовувати шаблонізатор Thymeleaf разом з класичними HTML, CSS та JS файлами.

Thymeleaf [19] – це серверний рушій шаблонів для Java, орієнтований на формування HTML-представлення за принципом «natural templating». У цій парадигмі вихідні шаблони зберігають валідність і зрозумілість як звичайні HTML-документи, придатні до перегляду в браузері без попередньої обробки. Логіка відмалювання виражається семантичними атрибутами, що не порушують структуру розмітки, а отже, полегшують взаємодію між розробниками й дизайнерами, зберігають доступність та забезпечують чисте розділення відповідальностей між контролерами, моделлю даних і шаром представлення.

Архітектурно Thymeleaf побудований навколо системи «діалектів» і процесорів атрибутів, де стандартний діалект надає базовий набір конструкцій для підстановки значень, умовного відображення, ітерацій над колекціями, роботи з фрагментами, повідомленнями локалізації та формуванням посилань. Розширюваність досягається через підключення власних діалектів, що дозволяє вводити доменно-специфічні атрибути та перетворення без зміни вихідної моделі виконання.

Підсистема розв’язувачів шаблонів працює з різними джерелами – файловою системою, classpath-ресурсами чи зовнішніми сховищами; поверх неї реалізовано кешування попередньо розібраних шаблонів, що знижує накладні витрати в продакшн-середовищах і забезпечує передбачувану продуктивність. Механізм режимів шаблонів підтримує HTML, XML, текстові й скриптові контексти, що дозволяє використовувати єдині вирази й атрибути для різних типів вбудованого контенту, включно з JavaScript і CSS.

Інтеграція зі Spring-екосистемою здійснюється природним чином через

спеціалізований Spring-діалект і компоненти представлення. У контексті Spring MVC шаблони сприймаються як звичайні подання, а дані моделі прив’язуються до форм за допомогою стандартних атрибутів. Підтримуються механізми валідації з відображенням повідомлень про помилки на рівні полів, використовується єдиний сервіс перетворення типів, а також ресурси локалізації, визначені у конфігурації застосунку.

Окремий набір розширень забезпечує гармонійну взаємодію з підсистемами безпеки, включно з умовним показом елементів залежно від ролей і стану автентифікації. Таке поєднання спрощує реалізацію повного циклу подання форм: від ініціалізації значень і CSRF-захисту до відображення результатів перевірок і повернення користувача до виправлення даних.

В якості системи керування базами даних обрано PostgreSQL.

PostgreSQL [20] – це сучасна об’єктно-реляційна система керування базами даних, яка вирізняється високою стабільністю роботи, широкими функціональними можливостями та гнучкістю налаштувань. Її історія розвитку бере початок ще у 90-х роках, а перша офіційна версія вийшла у 1996 році. Відтоді PostgreSQL постійно вдосконалюється завдяки активній міжнародній спільноті та відкритому вихідному коду.

На відміну від багатьох комерційних СУБД, PostgreSQL є повністю безкоштовним рішенням з ліцензією open source, що дає змогу вільно використовувати її в навчальних, комерційних та великих промислових проєктах. Завдяки відкритості коду розробники можуть адаптувати систему під власні потреби, створювати розширення або оптимізувати роботу для специфічних сценаріїв.

3.2 Розробка модулів системи

Програмна реалізація вебсистеми складається з чотирьох основних модулів, кожен з яких реалізовує свій метод в системі:

– метод пошуку координат за текстовою адресою;

– метод пошуку замовлень в радіусі встановленого відхилення від

маршруту;

– метод формування рейтингу користувачів системи;

– метод формування статистики профілю користувача.

Для наглядної демонстрації структури програмної реалізації цих методів та їх модулів, було створено та описано діаграми класів для кожного з них.

Діаграма класів модуля пошуку координат за текстовою адресою зображена на рисунку 3.1.

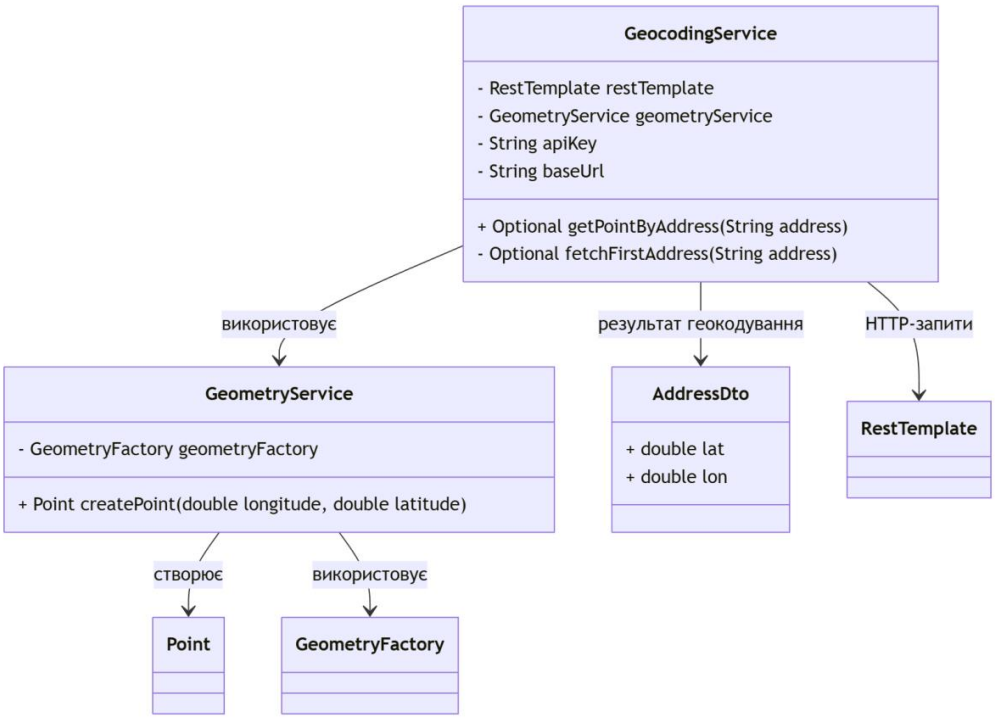

Рисунок 3.1 – Діаграма класів модуля пошуку координат за текстовою адресою

Клас GeocodingService відповідає за виклик зовнішнього геокодувального веб-сервісу та перетворення отриманих координат у геометричну точку. Він містить такі поля:

– restTemplate: RestTemplate – HTTP-клієнт, який використовується для надсилання GET-запитів до сервісу геокодування та отримання відповіді у форматі JSON;

– geometryService: GeometryService – сервіс роботи з геометрією, якому делегується створення об’єкта Point на основі отриманих координат;

– apiKey: String – API-ключ, що використовується для авторизації запитів до зовнішнього геокодувального сервісу;

– baseUrl: String – базова URL-адреса веб-сервісу геокодування, до якої додаються параметри запиту.

Для реалізації методу пошуку координат клас використовує такі функції:

– getPointByAddress(address: String): Optional<Point> – публічний метод, який приймає текстову адресу, перевіряє коректність вхідних даних, викликає допоміжний метод отримання результатів геокодування, а в разі успіху перетворює широту та довготу в об’єкт Point за допомогою GeometryService і повертає його у вигляді Optional. Якщо координати не знайдені або сталася помилка під час виклику зовнішнього сервісу, метод повертає порожнє значення;

– fetchFirstAddress(address: String): Optional<AddressDto> – приватний метод, що формує URI запиту з використанням baseUrl, apiKey та параметра q, виконує HTTP GET-запит через restTemplate, перевіряє код стану відповіді, перетворює тіло відповіді на масив AddressDto[] та повертає перший елемент масиву як найрелевантніший результат. У разі неуспішного статусу, відсутності результатів або винятку під час виклику зовнішнього сервісу цей метод повертає порожнє значення.

Клас GeometryService виконує перетворення числових координат у геометричні об’єкти бібліотеки JTS у системі координат WGS84 (SRID 4326).

Він містить таке поле:

– geometryFactory: GeometryFactory – фабрика геометрій, ініціалізована з використанням моделі точності (PrecisionModel) та простору координат з ідентифікатором 4326, яка відповідає за створення об’єктів типу Point та інших геометричних примітивів.

Для роботи з координатами клас використовує таку функцію:

– createPoint(longitude: double, latitude: double): Point – створює об’єкт Coordinate з переданою довготою та широтою (x = довгота, y = широта), на його основі формує об’єкт Point за допомогою geometryFactory та повертає його для

подальшого збереження або використання в просторових обчисленнях.

Клас AddressDto є передавальним об’єктом даних (DTO), який використовується для десеріалізації JSON-відповіді геокодувального сервісу у внутрішнє представлення.

Він містить такі поля:

– lat: double – значення широти точки, отриманої в результаті геокодування текстової адреси;

– lon: double – значення довготи відповідної точки.

Об’єкти AddressDto створюються автоматично під час розбору відповіді у класі GeocodingService і передаються в GeometryService для формування геометричної точки Point.

Діаграма класів модуля пошуку замовлень в радіусі встановленого відхилення від маршруту зображена на рисунку 3.2.

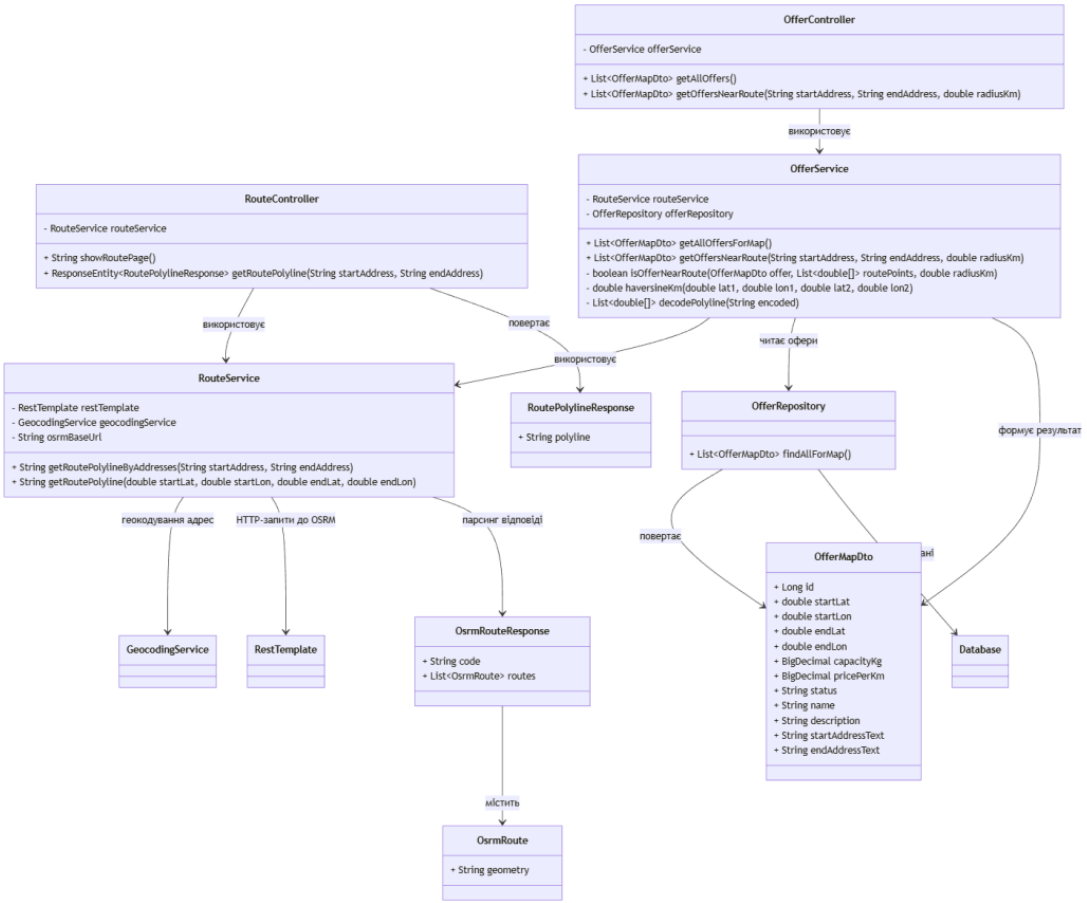

Рисунок 3.2 – Діаграма класів модуля пошуку замовлень в радіусі встановленого відхилення від маршруту

Клас RouteController виконує роль веб-контролера, який обробляє HTTP-запити, пов’язані з побудовою маршруту.

Він містить таке поле:

– routeService: RouteService – сервісний компонент, якому контролер делегує логіку побудови маршруту та отримання polyline між заданими адресами.

Для обробки запитів контролер використовує такі функції:

– showRoutePage(): String – повертає назву Thymeleaf-шаблону сторінки з картою, на якій користувач може задати параметри маршруту;

– getRoutePolyline(startAddress: String, endAddress: String): ResponseEntity<RoutePolylineResponse> – приймає текстові адреси початку та завершення маршруту, передає їх до RouteService, отримує закодовану polyline та повертає її клієнту у вигляді об’єкта RoutePolylineResponse у форматі JSON.

Клас OfferController є REST-контролером, що відповідає за надання даних про перевізницькі пропозиції (офери) для відображення на карті та для пошуку біля маршруту.

Він містить таке поле:

– offerService: OfferService – сервіс бізнес-логіки, який здійснює завантаження, фільтрацію та підготовку оферів до повернення клієнту.

У складі класу використовуються такі функції:

– getAllOffers(): List<OfferMapDto> – повертає повний список доступних оферів для відображення на карті (наприклад, при первинному завантаженні сторінки);

– getOffersNearRoute(startAddress: String, endAddress: String, radiusKm: double): List<OfferMapDto> – приймає параметри маршруту та радіус допустимого відхилення, викликає сервісний метод пошуку оферів біля маршруту та повертає лише відібрані пропозиції, що задовольняють умову близькості до маршруту.

Клас RouteService інкапсулює логіку побудови маршруту між двома точками та взаємодії з зовнішнім маршрутизатором OSRM.

Він містить такі поля:

– restTemplate: RestTemplate – HTTP-клієнт, який використовується для надсилання REST-запитів до сервісу маршрутизації OSRM та отримання відповіді у форматі JSON;

– geocodingService: GeocodingService – сервіс геокодування, який перетворює текстові адреси у координати точок (широта і довгота);

– osrmBaseUrl: String – базова URL-адреса веб-сервісу OSRM, до якої додаються параметри маршруту та опції в запиті.

Для побудови маршруту клас використовує такі функції:

– getRoutePolylineByAddresses(startAddress: String, endAddress: String): String – виконує геокодування початкової та кінцевої адреси за допомогою GeocodingService, отримує їх координати та делегує побудову маршруту методу взаємодії з OSRM, повертаючи закодовану polyline;

– getRoutePolyline(startLat: double, startLon: double, endLat: double, endLon: double): String – на основі координат початкової та кінцевої точки формує HTTP-запит до OSRM API, викликає зовнішній сервіс, перевіряє коректність відповіді та повертає рядок із закодованою геометрією маршруту.

Клас OfferService реалізує бізнес-логіку роботи з перевізницькими пропозиціями та їх фільтрації відносно побудованого маршруту.

Він містить такі поля:

– routeService: RouteService – сервіс, який відповідає за побудову маршруту між заданими адресами та отримання polyline;

– offerRepository: OfferRepository – репозиторій доступу до бази даних, через який завантажуються всі активні офери з координатами точок відправлення та прибуття.

Для реалізації функціональності пошуку оферів використовуються такі функції:

– getAllOffersForMap(): List<OfferMapDto> – завантажує з бази даних повний список оферів у вигляді об’єктів OfferMapDto, придатних для відображення на карті або подальшої обробки;

– getOffersNearRoute(startAddress: String, endAddress: String, radiusKm: double): List<OfferMapDto> – перевіряє коректність заданого радіуса, отримує polyline маршруту за допомогою RouteService, декодує її у список точок, завантажує всі офери з бази та для кожного з них визначає, чи знаходяться його точки відправлення й прибуття в радіусі radiusKm від маршруту, формуючи підсумковий список оферів, що задовольняють умову близькості;

– isOfferNearRoute(offer: OfferMapDto, routePoints: List<double[]>, radiusKm: double): boolean – приватний допоміжний метод, який для заданого офера проходить по всіх точках маршруту, обчислює відстань до точки відправлення та точки прибуття, встановлює логічні прапорці близькості та повертає «істина», якщо обидві точки офера виявилися не далі за заданий радіус від маршруту;

– haversineKm(lat1: double, lon1: double, lat2: double, lon2: double): double – приватний метод, що реалізує обчислення відстані між двома географічними точками за формулою гаверсинуса, повертаючи результат у кілометрах з урахуванням сферичної моделі Землі;

– decodePolyline(encoded: String): List<double[]> – приватний метод декодування закодованого polyline (формат Google/OSRM) у впорядкований список точок маршруту, де кожен елемент містить пару значень широти та довготи у десятковому форматі.

Клас OfferRepository відображає шар доступу до бази даних для сутності оферів.

Він містить таку функцію:

– findAllForMap(): List<OfferMapDto> – виконує запит до бази даних, завантажує перелік доступних перевізницьких пропозицій та проєціює їх у вигляді об’єктів OfferMapDto, що містять необхідні для відображення на карті та фільтрації поля.

Клас OfferMapDto є передавальним об’єктом даних, який використовується для відображення оферів на карті та в інтерфейсі користувача, а також для участі в алгоритмах фільтрації.

Він містить такі поля:

– id: Long – унікальний ідентифікатор офера;

– startLat: double – широта точки відправлення для маршруту поточного перевізника;

– startLon: double – довгота точки відправлення для маршруту поточного перевізника;

– endLat: double – широта точки прибуття (кінцевого пункту) маршруту;

– endLon: double – довгота точки прибуття;

– capacityKg: BigDecimal – максимально доступна вантажопідйомність або об’єм посилок у кілограмах, який може бути перевезений у рамках даного офера;

– pricePerKm: BigDecimal – вартість перевезення за один кілометр маршруту;

– status: String – поточний статус офера (наприклад, активний, чернетка тощо), який використовується для відбору доступних пропозицій;

– name: String – коротка назва офера, що відображається користувачеві;

– description: String – текстовий опис деталей маршруту та умов перевезення;

– startAddressText: String – текстове представлення адреси відправлення, зручне для читання користувачем;

– endAddressText: String – текстове представлення адреси прибуття.

Клас RoutePolylineResponse є простим DTO, призначеним для повернення клієнту інформації про маршрут у вигляді закодованої polyline.

Він містить таке поле:

– polyline: String – рядок із закодованою геометрією маршруту, який клієнтська частина може декодувати та відобразити на карті.

Клас OsrmRouteResponse відображає структуру відповіді сервісу OSRM, що використовується при побудові маршруту.

Він містить такі поля:

– code: String – код статусу відповіді від OSRM (наприклад, «Ok» у разі

успішного знаходження маршруту), який використовується для перевірки коректності результату;

– routes: List<OsrmRoute> – список знайдених маршрутів, де кожен елемент містить геометрію окремого маршруту та може бути проаналізований для вибору найрелевантнішого варіанта.

Клас OsrmRoute є допоміжним об’єктом даних, який представляє окремий маршрут у відповіді OSRM.

Він містить таке поле:

– geometry: String – закодована у форматі polyline геометрія маршруту, побудованого між заданими початковою та кінцевою точками, яка в подальшому декодується у послідовність координат.

Клас GeocodingService на діаграмі виступає як зовнішній по відношенню до даного методу сервіс, що забезпечує перетворення текстових адрес у координати точок.

У цьому підпункті він фігурує як залежність для RouteService, а детальний опис його полів і функцій наведений у окремому підпункті, присвяченому методу геокодування.

Клас RestTemplate відображає стандартний клієнт Spring для виконання HTTP-запитів; у межах діаграми він позначає технічну залежність RouteService від засобу взаємодії з зовнішнім OSRM API без деталізації внутрішньої структури.

Сутність Database на діаграмі умовно представляє систему керування базою даних, у якій фізично зберігаються офери.

Взаємодія з нею здійснюється через OfferRepository, тому на рівні класів вона показана як абстракція без окремих полів та методів.

Діаграма класів модуля формування рейтингу користувачів системи зображена на рисунку 3.3.

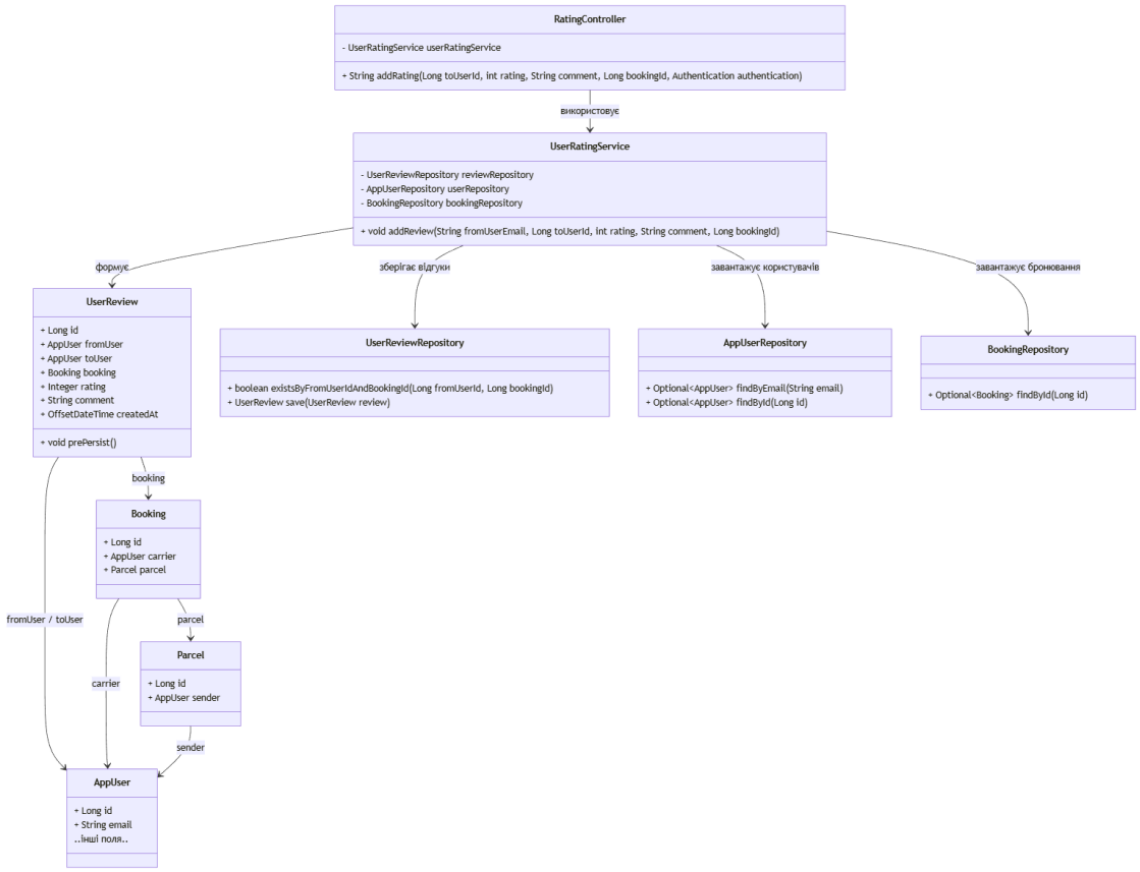

Рисунок 3.3 – Діаграма класів модуля формування рейтингу користувачів системи

Клас RatingController виконує роль веб-контролера, який приймає HTTP-запити на створення відгуків та делегує всю бізнес-логіку сервісному шару. Він містить таке поле:

– userRatingService: UserRatingService – сервіс, відповідальний за перевірку коректності вхідних даних, пошук користувачів і бронювання, а також безпосереднє створення об’єкта відгуку та його збереження в базі даних.

Для обробки запитів контролер використовує таку функцію:

– addRating(toUserId: Long, rating: int, comment: String, bookingId: Long, authentication: Authentication): String – приймає параметри форми з веб-інтерфейсу (toUserId, rating, comment, bookingId), отримує email поточного автентифікованого користувача з об’єкта Authentication, викликає сервісний метод addReview(...) і після успішного створення відгуку виконує перенаправлення користувача на сторінку історії ("redirect:/history").

Клас UserRatingService інкапсулює бізнес-правила формування рейтингу та перевірки права користувача залишити відгук.

Він містить такі поля:

– reviewRepository: UserReviewRepository – репозиторій для роботи з сутністю UserReview, який використовується для перевірки наявності вже існуючих відгуків та збереження нових записів;

– userRepository: AppUserRepository – репозиторій користувачів, через який завантажуються облікові записи автора відгуку і користувача, якого оцінюють;

– bookingRepository: BookingRepository – репозиторій бронювань, який використовується для завантаження запису Booking та перевірки участі користувача в конкретному замовленні.

Для реалізації формування рейтингу використовується така функція:

– addReview(fromUserEmail: String, toUserId: Long, rating: int, comment: String, bookingId: Long): void – виконує комплексний алгоритм створення відгуку

Клас UserReview є сутністю JPA, що представляє окремий відгук користувача в системі.

Він містить такі поля:

– id: Long – унікальний ідентифікатор відгуку в таблиці user_review;

– fromUser: AppUser – посилання на користувача, який залишив відгук (автор оцінки);

– toUser: AppUser – посилання на користувача, якого оцінюють;

– booking: Booking – (опційно) посилання на конкретне бронювання, у рамках якого було надано відгук, що дозволяє прив’язати рейтинг до певної виконаної доставки;

– rating: Integer – числова оцінка користувача за п’ятибальною шкалою (значення від 1 до 5);

– comment: String – текстовий коментар до оцінки, у якому користувач може деталізувати свій досвід співпраці;

– createdAt: OffsetDateTime – мітка часу створення відгуку, яка зберігається в часовому поясі UTC і не підлягає оновленню після запису.

Також клас містить службову функцію життєвого циклу:

– prePersist(): void – метод, відмічений анотацією @PrePersist, який автоматично викликається перед збереженням сутності в базу даних і встановлює значення createdAt поточним моментом часу, якщо поле ще не було заповнене.

Клас UserReviewRepository є інтерфейсом доступу до бази даних для сутності UserReview і розширює стандартний JpaRepository.

У контексті формування рейтингу використовуються такі функції:

– existsByFromUserIdAndBookingId(fromUserId: Long, bookingId: Long): boolean – перевіряє, чи існує вже відгук від заданого користувача по конкретному бронюванню; використовується для заборони дублюючих відгуків і гарантії принципу «одне замовлення – один відгук від одного користувача»;

– save(review: UserReview): UserReview – зберігає переданий екземпляр UserReview у базі даних, створюючи новий запис у таблиці user_review або оновлюючи існуючий (у даному випадку використовується для первинного створення нового відгуку).

Клас AppUserRepository є репозиторієм для роботи з сутністю користувача AppUser.

У рамках даного методу формування рейтингу використовуються такі функції:

– findByEmail(email: String): Optional<AppUser> – виконує пошук користувача в базі даних за його електронною адресою; використовується для завантаження автора відгуку на основі інформації з контексту аутентифікації;

– findById(id: Long): Optional<AppUser> – знаходить користувача за його унікальним ідентифікатором; використовується для отримання об’єкта користувача, якого оцінюють (toUser).

Клас BookingRepository забезпечує доступ до сутності Booking, що представляє бронювання доставки.

У даному методі формування рейтингу використовується така функція:

– findById(id: Long): Optional<Booking> – виконує пошук бронювання за його ідентифікатором bookingId; застосовується для прив’язки відгуку до конкретного замовлення та подальшої перевірки участі користувача в цьому бронюванні.

Клас AppUser представляє обліковий запис користувача системи. У контексті формування рейтингу важливими є такі поля:

– id: Long – унікальний ідентифікатор користувача в базі даних, який використовується для встановлення зв’язків із відгуками (fromUser, toUser) та бронюваннями;

– email: String – електронна адреса користувача, яка використовується для аутентифікації та для пошуку автора відгуку на основі даних безпеки. Інші поля (ПІБ, ролі, контактні дані тощо) у межах даного методу не деталізуються.

Клас Booking є сутністю, що описує конкретне бронювання доставки посилки між відправником і перевізником.

Для методу формування рейтингу мають значення такі поля:

– id: Long – унікальний ідентифікатор бронювання, який використовується для прив’язки відгуку до конкретного замовлення;

– carrier: AppUser – посилання на користувача-перевізника, який виконує доставку й може бути оцінений за результатами виконаного замовлення;

– parcel: Parcel – посилання на сутність посилки, що містить інформацію про відправника (та, за потреби, інші параметри відправлення).

Клас Parcel представляє відправлення (посилку), що бере участь у процесі бронювання.

У даному методі важливе таке поле:

– id: Long – унікальний ідентифікатор посилки;

– sender: AppUser – посилання на користувача-відправника посилки, який також може залишати або отримувати відгуки в процесі взаємодії з перевізником.

Таким чином, сукупність описаних класів реалізує повний цикл формування рейтингу: від прийому даних із веб-інтерфейсу, через валідацію прав користувача та унікальності відгуку, до створення та збереження сутності UserReview, що використовується в системі як основа для подальшого аналізу репутації користувачів.

Діаграма класів модуля формування рейтингу користувачів системи зображена на рисунку 3.4.

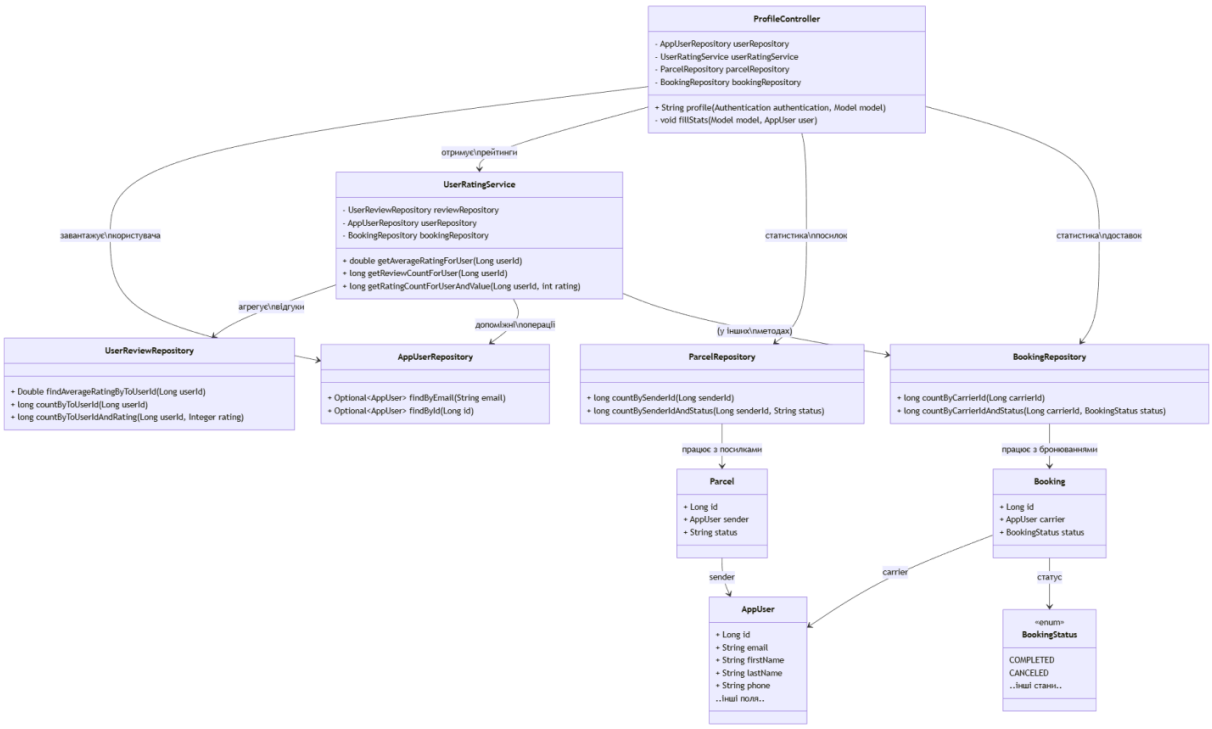

Рисунок 3.4 – Діаграма класів модуля формування рейтингу користувачів системи

Клас ProfileController відповідає за формування даних для відображення сторінки профілю користувача, включно зі статистикою його активності та рейтингу.

Він містить такі поля:

– userRepository: AppUserRepository – репозиторій користувачів, який використовується для завантаження поточного користувача за його email із контексту аутентифікації;

– userRatingService: UserRatingService – сервіс, відповідальний за обчислення усіх показників рейтингу (середнє значення, кількість відгуків, розподіл оцінок);

– parcelRepository: ParcelRepository – репозиторій посилок, який забезпечує доступ до статистики відправлених користувачем посилок;

– bookingRepository: BookingRepository – репозиторій бронювань, що використовується для отримання статистики доставок, у яких користувач виступав перевізником.

Контролер використовує такі функції:

– profile(authentication: Authentication, model: Model): String – метод-обробник запиту GET /profile, який завантажує поточного користувача за email, формує об’єкт форми для редагування профілю, додає користувача та форму до моделі, викликає fillStats(...) для заповнення статистичних атрибутів і повертає назву шаблону "profile";

– fillStats(model: Model, user: AppUser): void – приватний допоміжний метод, який на основі ідентифікатора користувача послідовно отримує статистику рейтингу, посилок і доставок через відповідні сервіси та репозиторії, після чого додає всі обчислені значення до моделі для подальшого відображення в профілі.

Клас UserRatingService реалізує сервісну логіку роботи з рейтингами користувачів. У контексті формування статистики використовуються його агрегуючі методи.

Він містить такі поля (для цього методу важливий лише доступ до відгуків):

– reviewRepository: UserReviewRepository – репозиторій відгуків, через який виконуються запити щодо середнього рейтингу, кількості відгуків та розподілу оцінок.

У рамках формування статистики профілю застосовуються такі функції:

– getAverageRatingForUser(userId: Long): double – обчислює середній рейтинг користувача за всіма відгуками, у яких він виступає одержувачем

(toUser); у разі відсутності відгуків повертає значення за замовчуванням і округлює результат до одного знаку після коми;

– getReviewCountForUser(userId: Long): long – повертає загальну кількість відгуків, залишених для даного користувача;

– getRatingCountForUserAndValue(userId: Long, rating: int): long – повертає кількість відгуків із конкретним значенням рейтингу (1, 2, 3, 4 або 5), що дозволяє побудувати розподіл оцінок для діаграм профілю.

Клас UserReviewRepository є інтерфейсом доступу до сутності відгуку UserReview і розширює JpaRepository.

У контексті формування статистики використовуються такі методи:

– findAverageRatingByToUserId(userId: Long): Double – обчислює середнє значення поля rating для всіх відгуків, адресованих користувачу з ідентифікатором userId; застосовується в сервісі для розрахунку середнього рейтингу;

– countByToUserId(userId: Long): long – повертає загальну кількість відгуків, у яких поле toUser.id відповідає заданому користувачу;

– countByToUserIdAndRating(userId: Long, rating: Integer): long – повертає кількість відгуків із конкретним значенням рейтингу для даного користувача, що використовується для підрахунку кількості оцінок 1, 2, 3, 4 і 5.

Клас ParcelRepository забезпечує доступ до сутності Parcel, яка представляє посилку, відправлену користувачем.

Для формування статистики профілю використовуються такі функції:

– countBySenderId(senderId: Long): long – підраховує загальну кількість посилок, створених користувачем як відправником;

– countBySenderIdAndStatus(senderId: Long, status: String): long – повертає кількість посилок даного користувача у певному статусі (наприклад, «доставлено» чи «скасовано»), що дозволяє виділити успішні та скасовані відправлення.

Клас BookingRepository відповідає за доступ до сутності Booking, яка представляє бронювання доставки з участю перевізника.

У рамках статистики профілю використовуються такі методи:

– countByCarrierId(carrierId: Long): long – повертає загальну кількість бронювань, у яких користувач виступає перевізником;

– countByCarrierIdAndStatus(carrierId: Long, status: BookingStatus): long – підраховує кількість бронювань певного статусу (наприклад, COMPLETED або CANCELED) для конкретного перевізника, що дозволяє виділити завершені та скасовані доставки.

Клас AppUserRepository є репозиторієм для роботи з обліковими записами користувачів (AppUser).

У даному методі статистики задіяні такі функції:

– findByEmail(email: String): Optional<AppUser> – виконує пошук користувача за його електронною адресою; використовується в ProfileController для завантаження поточного користувача за даними аутентифікації;

– findById(id: Long): Optional<AppUser> – загальний метод пошуку користувача за ідентифікатором, який може використовуватися іншими сервісами (зокрема, у UserRatingService).

Клас AppUser представляє обліковий запис користувача в системі. У контексті формування статистики профілю особливо важливі такі поля:

– id: Long – унікальний ідентифікатор користувача, що використовується як ключ у всіх запитах до репозиторіїв статистики;

– email: String – електронна адреса, за якою здійснюється аутентифікація та завантаження профілю;

– firstName: String – ім’я користувача, що відображається у профілі та попередньо заповнюється у формі редагування;

– lastName: String – прізвище користувача;

– phone: String – номер телефону, який також передається до форми оновлення профілю.

Інші поля (ролі, паролі, додаткова інформація) в рамках даного методу не деталізуються.

Клас Parcel є сутністю, що описує окрему посилку в системі.

Для обчислення статистики по відправленнях використовуються такі поля:

– id: Long – унікальний ідентифікатор посилки;

– sender: AppUser – посилання на користувача-відправника, чий ідентифікатор використовується у методах countBySenderId(...);

– status: String – статус посилки (наприклад, доставлена, скасована тощо), що дозволяє виділяти різні групи відправлень у статистиці.

Клас Booking представляє бронювання доставки, у рамках якого перевізник здійснює перевезення посилки.
У контексті статистики профілю мають значення такі поля:

– id: Long – унікальний ідентифікатор бронювання;

– carrier: AppUser – користувач-перевізник, для якого рахується кількість доставок за допомогою countByCarrierId(...);

– status: BookingStatus – статус бронювання (наприклад, завершено або скасовано), що використовується для формування статистики завершених і скасованих доставок.

Перерахування BookingStatus є enum-типом, що описує можливі стани бронювання.

У межах формування статистики використовуються, зокрема, такі значення:

– COMPLETED – бронювання успішно завершене, доставка виконана;

– CANCELED – бронювання скасоване.

Таким чином, сукупність описаних класів реалізує метод формування статистики профілю користувача: від завантаження його облікового запису й обчислення рейтингових показників до підрахунку створених відправлень і виконаних доставок для подальшого відображення в інтерфейсі системи.

3.3 Розгортання системи

Розгортання вебсистеми планується у контейнеризованому середовищі. Технологія контейнеризації дозволяє розгортати програму дуже компактно [21]. Контейнер застосунку є самодостатньою одиницею обчислень із чітко

визначеними залежностями та конфігурацією через змінні середовища; контейнер сховища даних працює окремо та надає стабільну точку з'єднання для транзакцій. Мережевий канал між контейнерами ізольований у межах віртуальної мережі середовища виконання, що спрощує контроль доступу та діагностику; зберігання даних винесене у томи, відокремлені від життєвого циклу контейнерів. Така композиція дозволяє відтворювати середовища локально та у виробництві без розбіжностей у конфігурації, а також поєднує гнучкість оновлень застосунку з консервативністю керування станом даних.

Схема з сервісною топологією зображено на рисунку 3.5. Клієнтський агент ініціює HTTP-запити; вебсервіс у контейнері приймає їх, виконує валідації, координує варіанти використання, звертається до підсистеми персистентності та повертає сформовані сервером сторінки. Контейнер зі сховищем даних забезпечує постійну двосторонню взаємодію із застосунком у межах внутрішньої мережі.

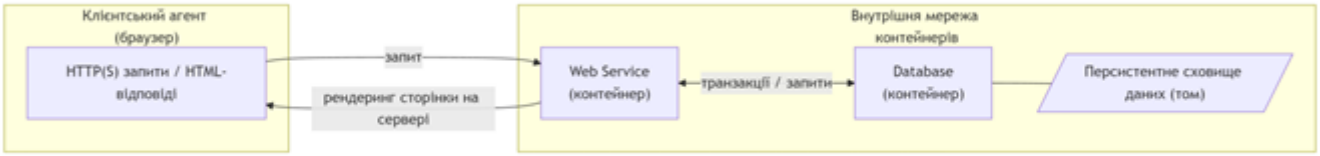

Рисунок 3.5 – Зображення сервісної топології

Таким чином було обґрунтовано вибір контейнеризованого середовища розгортання та описано сервісну топологію вебсистеми з розділенням застосунку й сховища даних, ізоляцією мережевих взаємодій та винесенням стану у томи, що дозволяє забезпечити відтворюваність конфігурації на різних середовищах, спростити адміністрування й масштабування системи та підвищити надійність зберігання й оброблення даних.

3.4 Розробка інтерфейсу системи

Для розробки інтерфейсу вебсистеми організованого доставлення

посилок обрано шаблонізатор Thymeleaf разом з HTML [11], CSS [22] та Javascript [23]. Це забезпечує тісну інтеграцію з серверною частиною на базі Spring Boot, дає змогу формувати динамічні HTML-сторінки безпосередньо на боці сервера, спрощує повторне використання шаблонів і підтримку єдиного стилю оформлення, а також дозволяє гнучко поєднувати статичні ресурси (CSS, JavaScript) з даними бізнес-логіки для побудови зручного та послідовного користувацького інтерфейсу.

Було розроблено інтерфейс сторінки логіну, структурна схема якого наведена на рисунку 3.6.

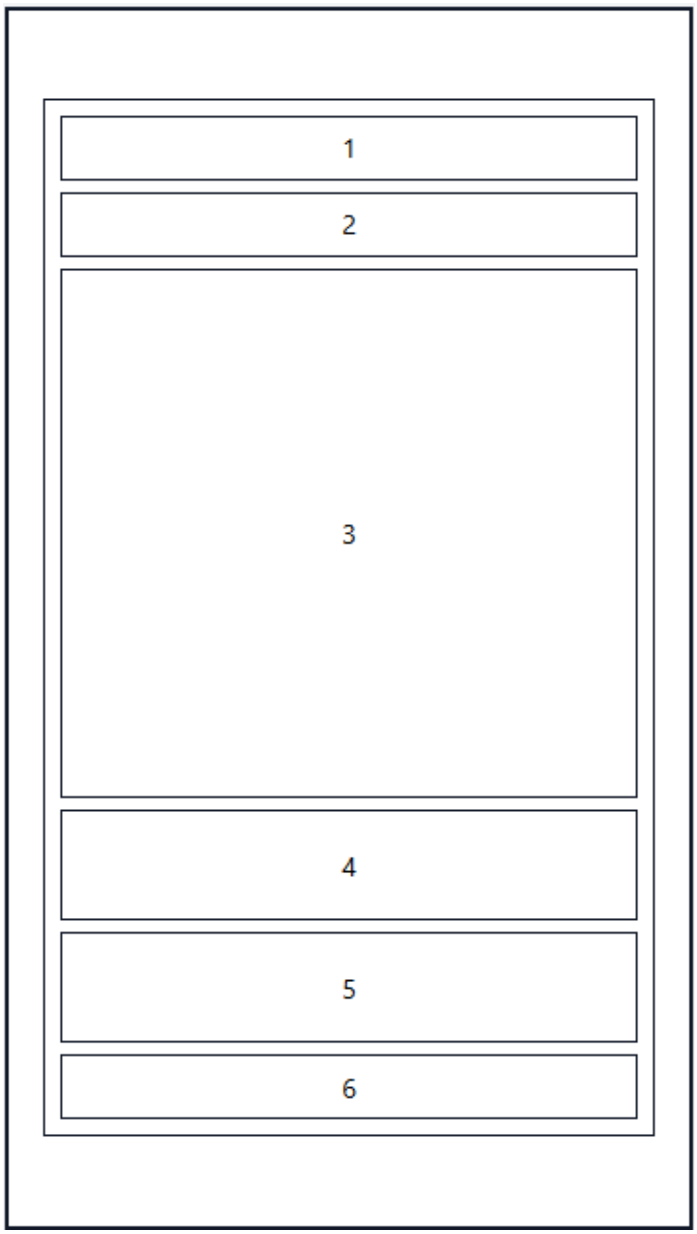

Рисунок 3.6 – Структурна схема сторінки логіну

Елементи сторінки:

1. Заголовок сторінки.
2. Пояснювальний підзаголовок.
3. Основна зона форми входу з полями для введення даних.
4. Кнопка входу «Увійти».
5. Блок службових повідомлень про помилки та успішні дії.
6. Посилання для переходу до реєстрації акаунта.

Інтерфейс сторінки логіну зображено на рисунку 3.7.

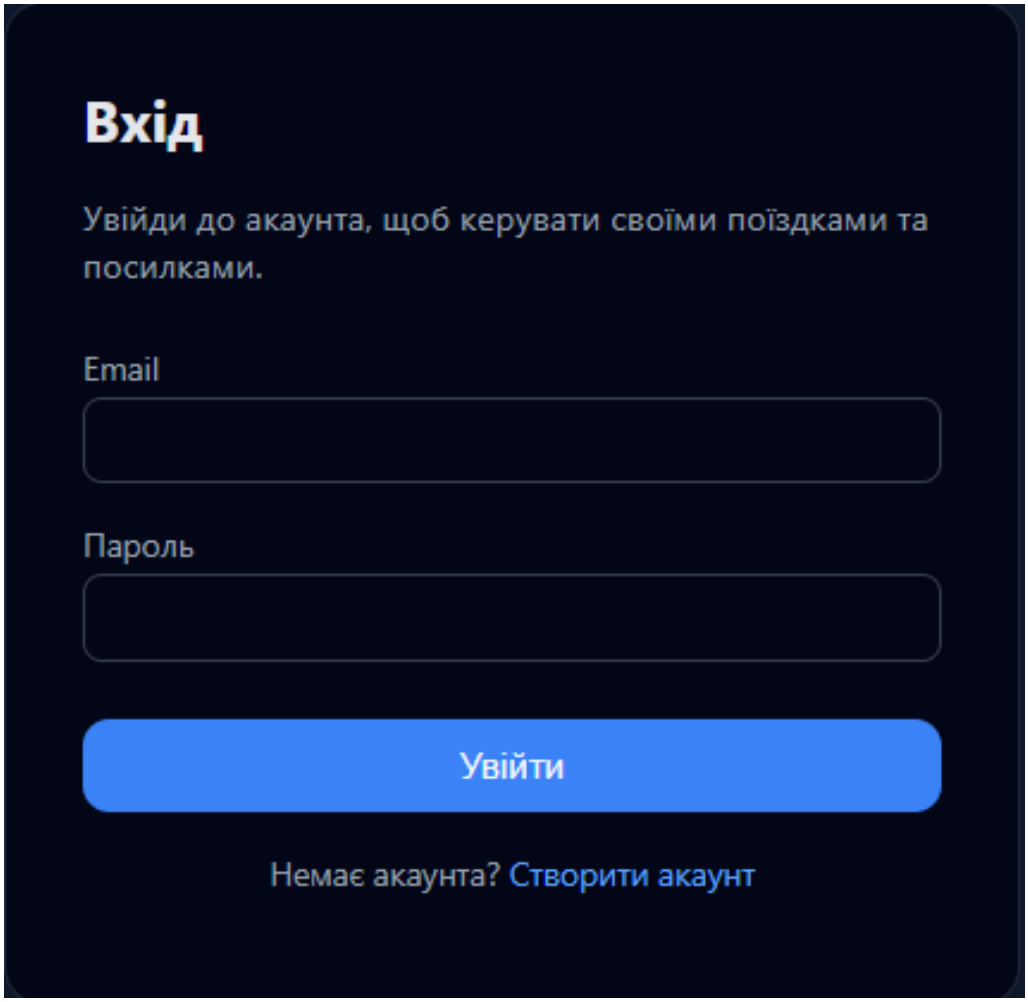

Рисунок 3.7 – Інтерфейс сторінки логіну

Було розроблено інтерфейс сторінки реєстрації, структурна схема якого наведена на рисунку 3.8.

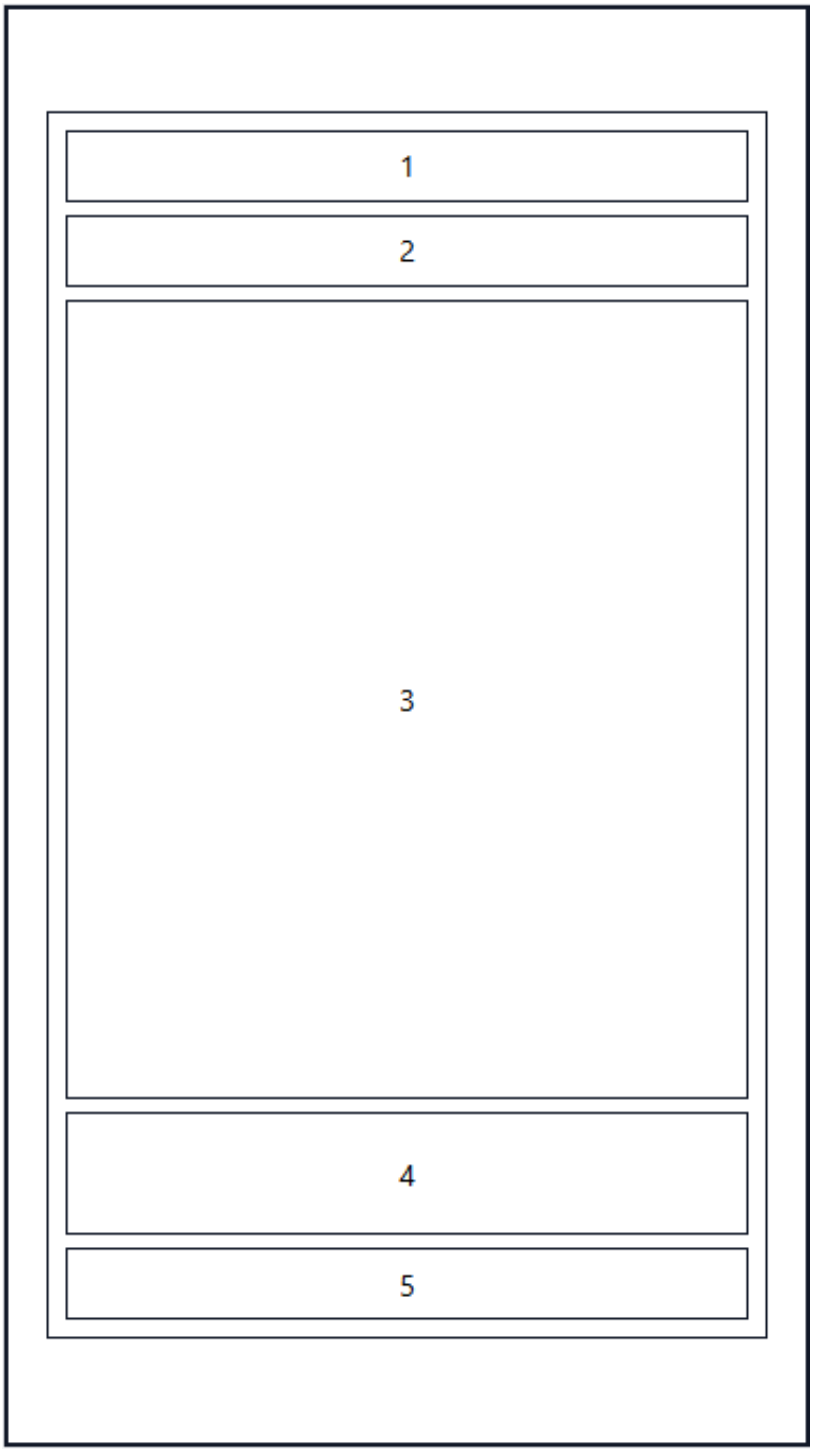

Рисунок 3.8 – Структурна схема сторінки реєстрації

Елементи сторінки:

1. Заголовок сторінки.
2. Пояснювальний підзаголовок.
3. Основна зона форми реєстрації з полями для введення особистих даних (ім’я, прізвище, email, телефон, пароль, підтвердження пароля).

4. Кнопка реєстрації «Створити акаунт».

5. Блок із текстом та посиланням для переходу на сторінку входу для користувачів, у яких вже є акаунт.

Інтерфейс сторінки реєстрації зображено на рисунку 3.9.

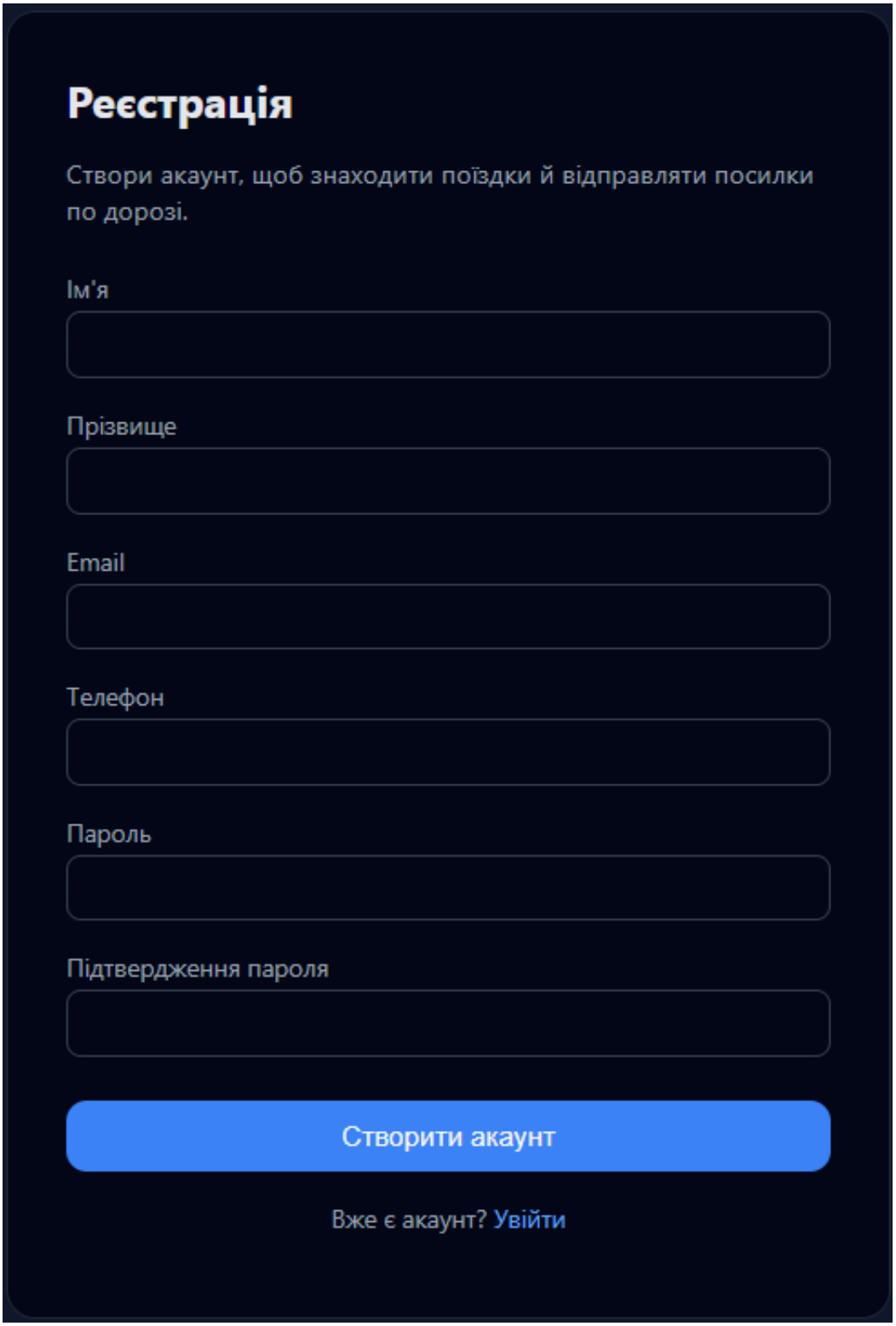

Рисунок 3.9 – Інтерфейс сторінки реєстрації

Було розроблено інтерфейс сторінки «Мої посилки», для перегляду даних про вже створені користувачем посилки, структурна схема якого наведена на рисунку 3.10.

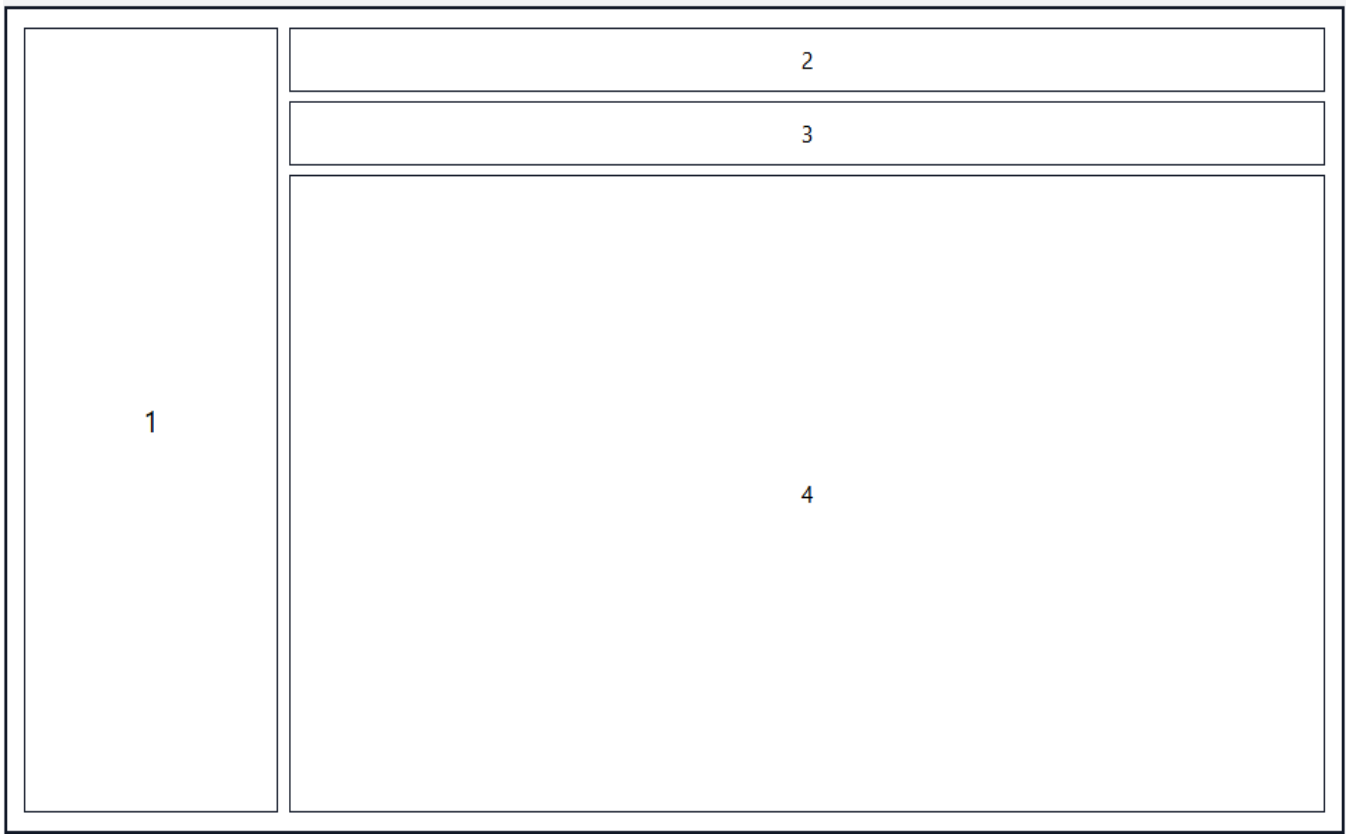

Рисунок 3.10 – Структурна схема сторінки «Мої посилки»

Елементи сторінки:

1. Блок навігації сторінкою (сайдбар) з посиланнями на основні розділи сервісу.

2. Заголовок сторінки з назвою розділу «Мої посилки».

3. Пояснювальний підзаголовок із коротким описом призначення списку посилок.

4. Основна зона контенту зі списком/таблицею створених посилок, станом порожнього списку та кнопками для зміни статусу посилки («Доставлено», «Скасувати»).

Інтерфейс сторінки «Мої посилки» зображено на рисунку 3.11.

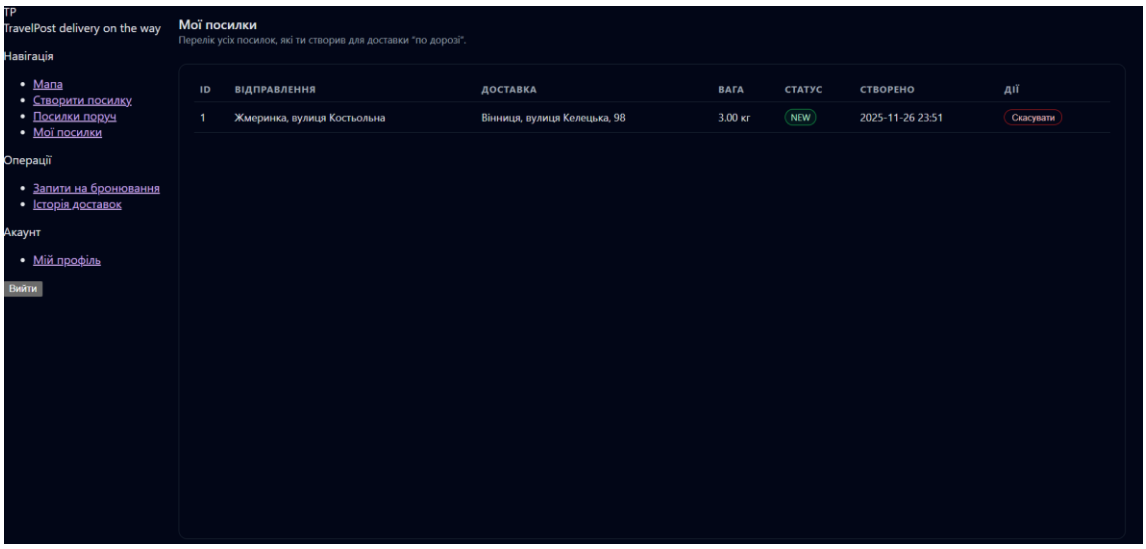

Рисунок 3.11 – Інтерфейс сторінки «Мої посилки»

Було розроблено інтерфейс сторінки пошуку посилок, структурна схема якого наведена на рисунку 3.12.

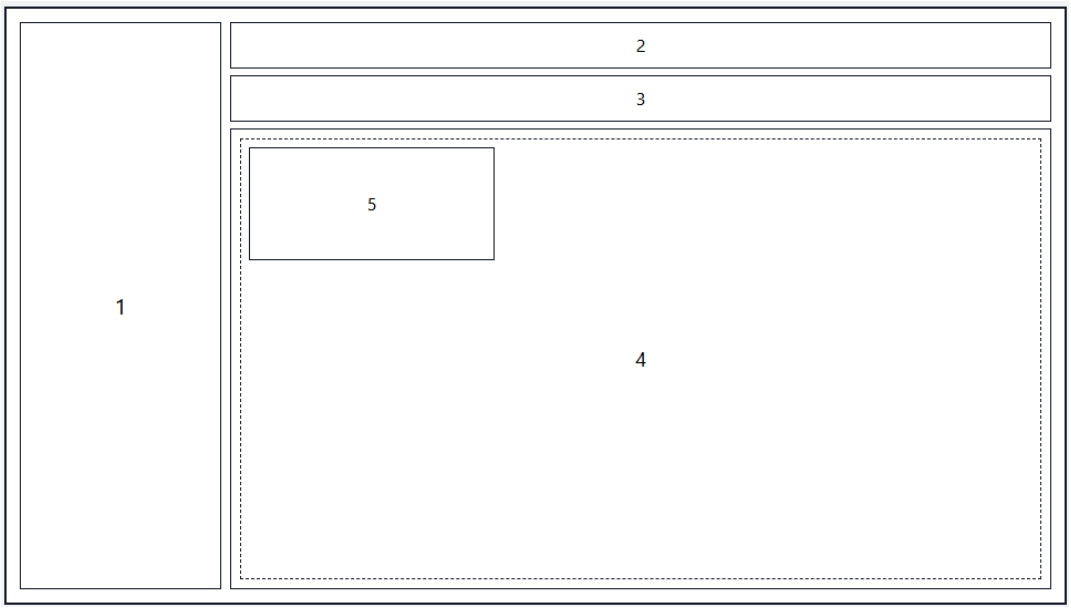

Рисунок 3.12 – Структурна схема сторінки пошуку посилок

Елементи сторінки:

1. Блок навігації сторінкою (сайдбар) з посиланнями на основні розділи сервісу.

2. Заголовок сторінки з назвою розділу «Посилки поруч».

3. Пояснювальний підзаголовок із коротким описом пошуку посилок у

радіусі від заданої адреси.

4. Основна зона карти з відображенням доступних посилок на мапі.

5. Панель керування пошуком посилок, розташована поверх карти, з полем для введення адреси, полем радіуса та кнопкою «Знайти посилки».

Інтерфейс сторінки пошуку посилок зображено на рисунку 3.13.

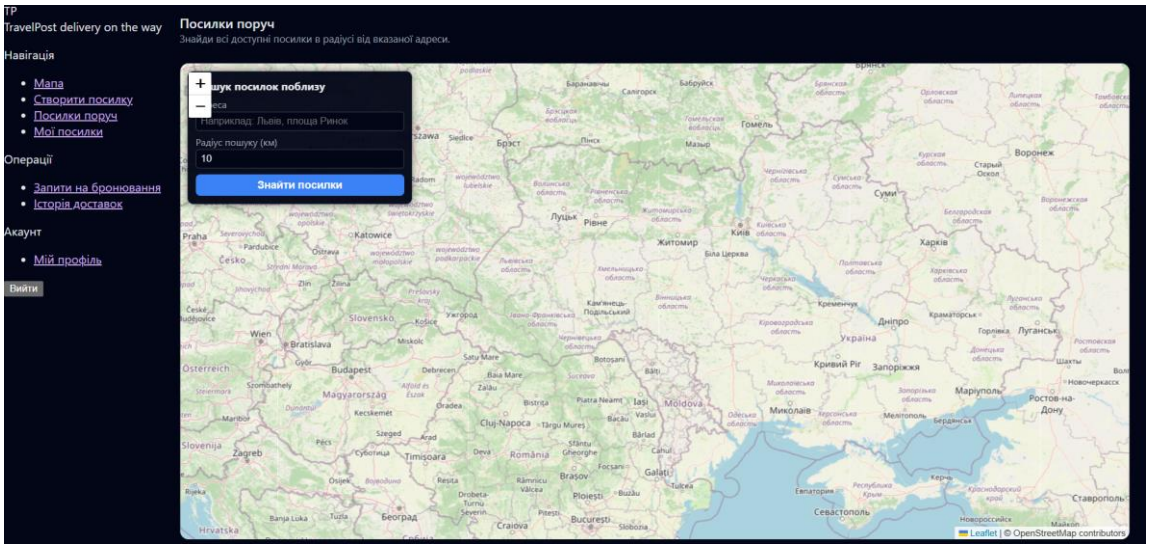

Рисунок 3.13 – Інтерфейс сторінки пошуку посилок

Було розроблено інтерфейс сторінки створення посилки, структурна схема якого наведена на рисунку 3.14.

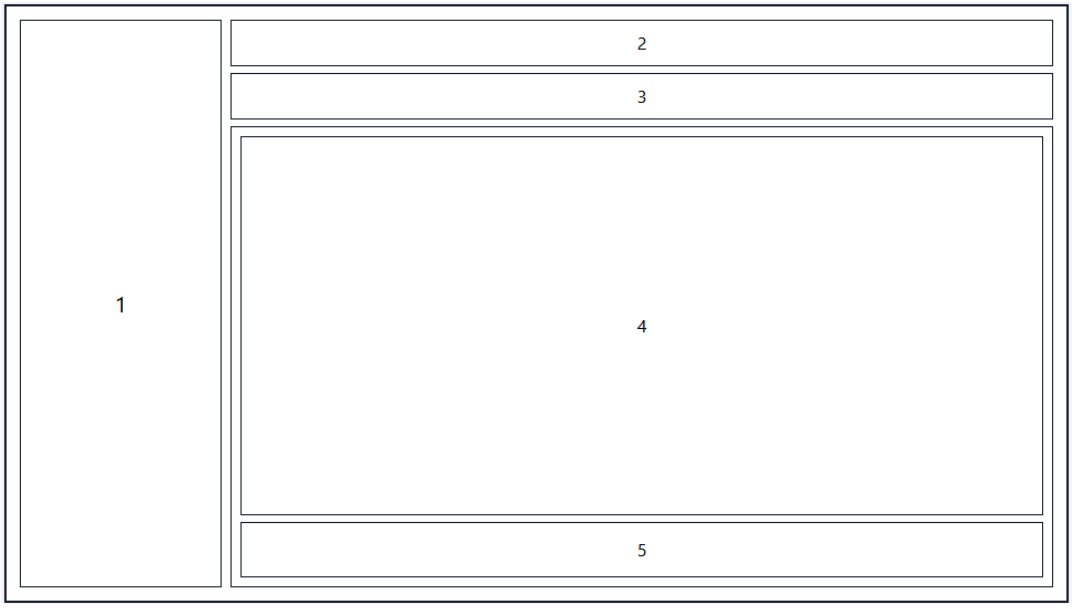

Рисунок 3.14 - Структурна схема сторінки створення посилки

Сторінка створення нової посилки містить кілька ключових інтерфейсних елементів.

Елементи сторінки:

1. Блок навігації сторінкою (сайдбар) з посиланнями на основні розділи сервісу.

2. Заголовок сторінки з назвою розділу «Створити посилку».

3. Пояснювальний підзаголовок із коротким описом призначення форми створення посилки.

4. Основна зона форми створення посилки з полями для введення адрес відправлення, доставки, ваги та опису.

5. Кнопка створення посилки «Створити посилку» , що відправляє запит на створення нової посилки.

Інтерфейс сторінки створення посилки зображено на рисунку 3.15.

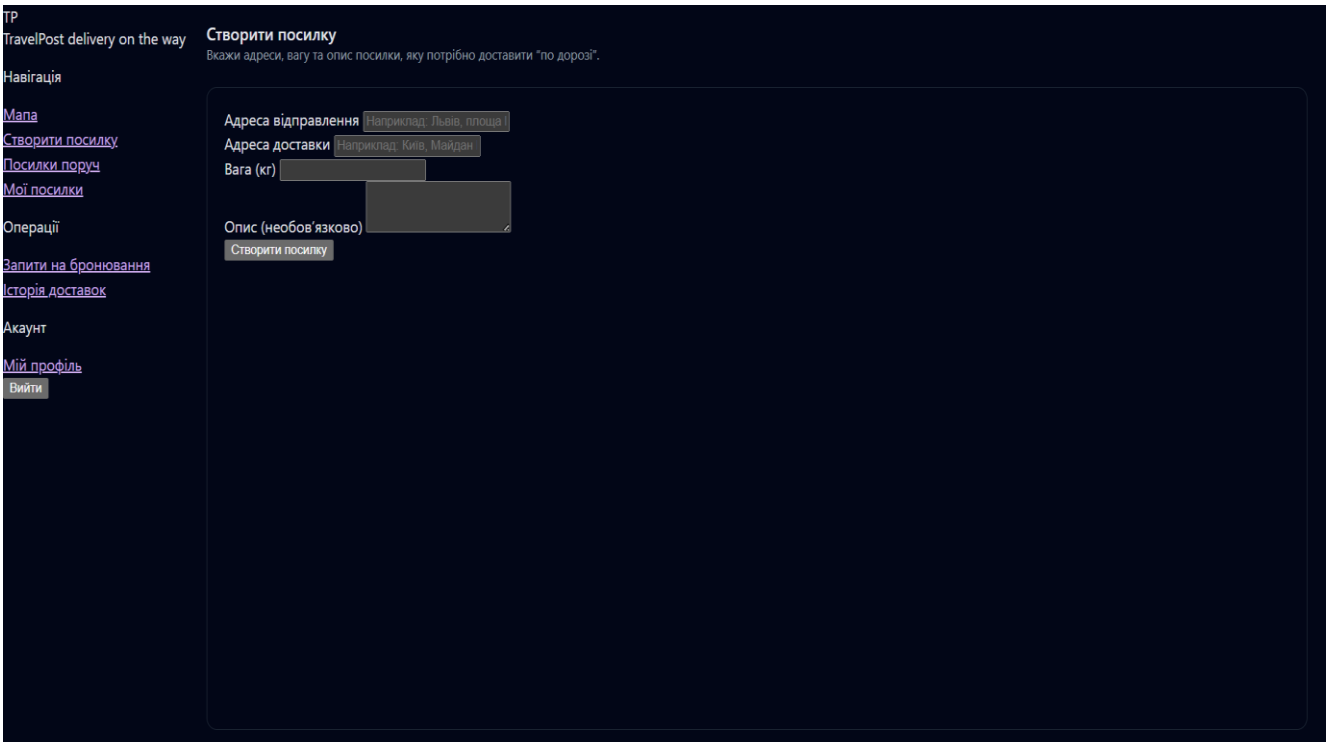

Рисунок 3.15 – Інтерфейс сторінки створення посилки

Було розроблено інтерфейс сторінки перегляду списку бронювань, структурна схема якого наведена на рисунку 3.16.

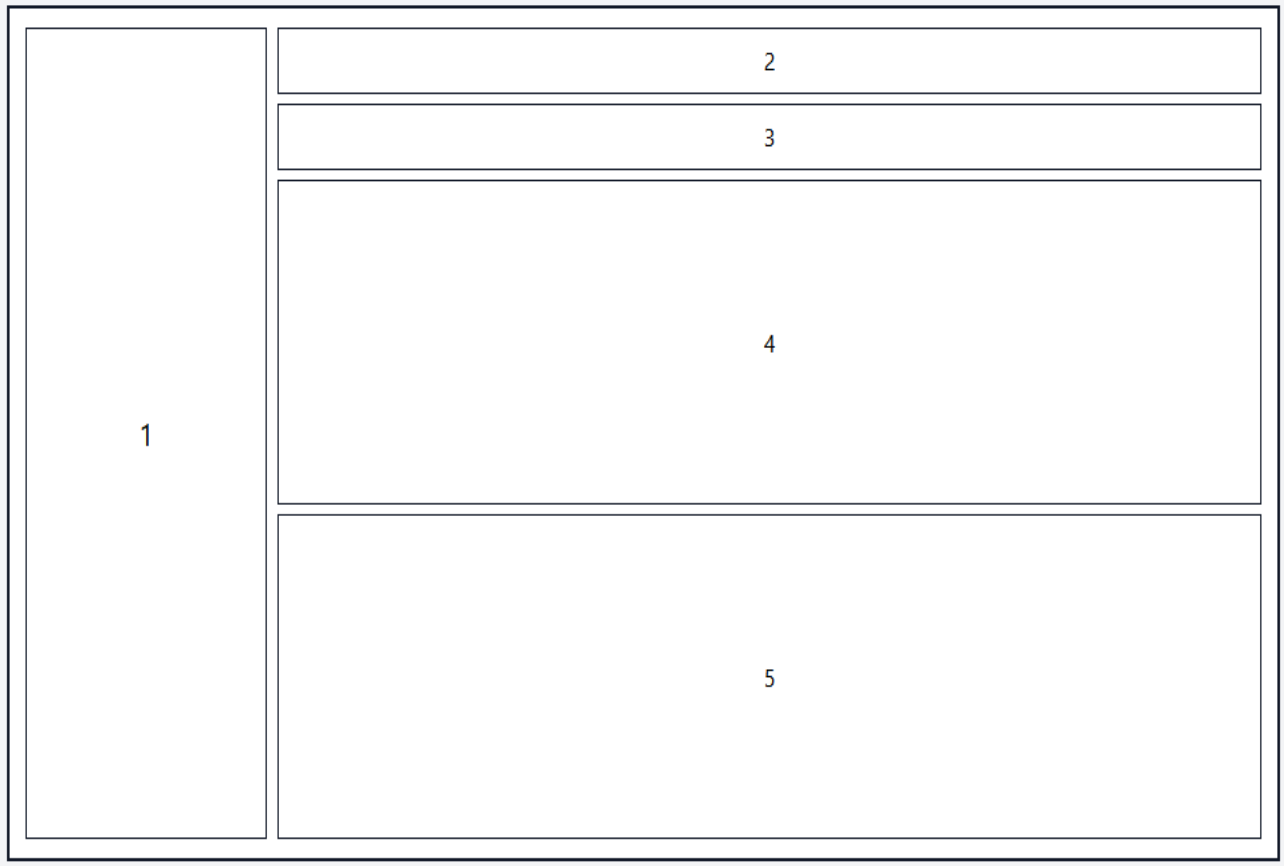

Рисунок 3.16 – Структурна схема сторінки перегляду списку бронювань

Сторінка перегляду запитів на бронювання містить кілька ключових інтерфейсних елементів.

Елементи сторінки:

1. Блок навігації сторінкою (сайдбар) з посиланнями на основні розділи сервісу.

2. Заголовок сторінки з назвою розділу «Запити на бронювання».

3. Пояснювальний підзаголовок із коротким описом відображених запитів.

4. Основна панель зі списком запитів до моїх посилок у вигляді таблиці або повідомлення про їх відсутність.

5. Додаткова панель зі списком моїх запитів як перевізника у вигляді таблиці або повідомлення про їх відсутність.

Інтерфейс сторінки перегляду списку бронювань зображено на рисунку 3.17.

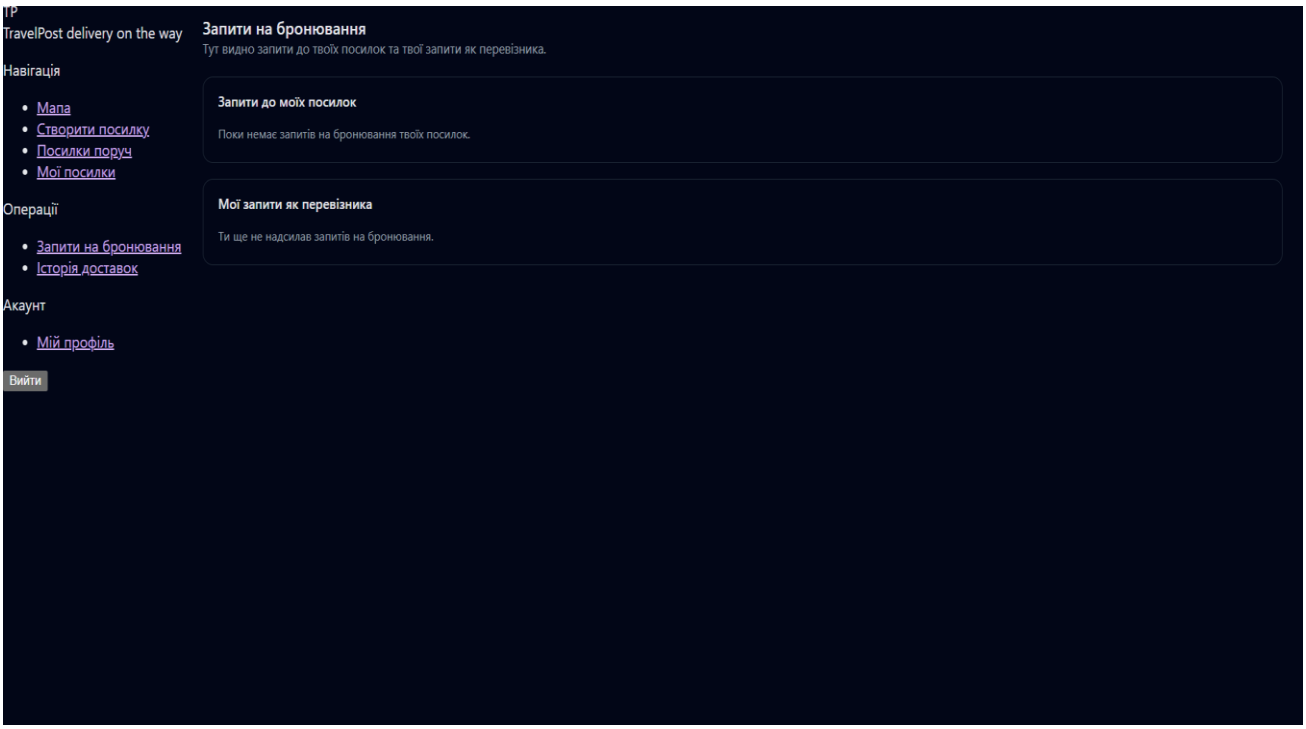

Рисунок 3.17 – Інтерфейс сторінки перегляду списку бронювань

Було розроблено інтерфейс сторінки перегляду історії посилок, структурна схема якого наведена на рисунку 3.18.

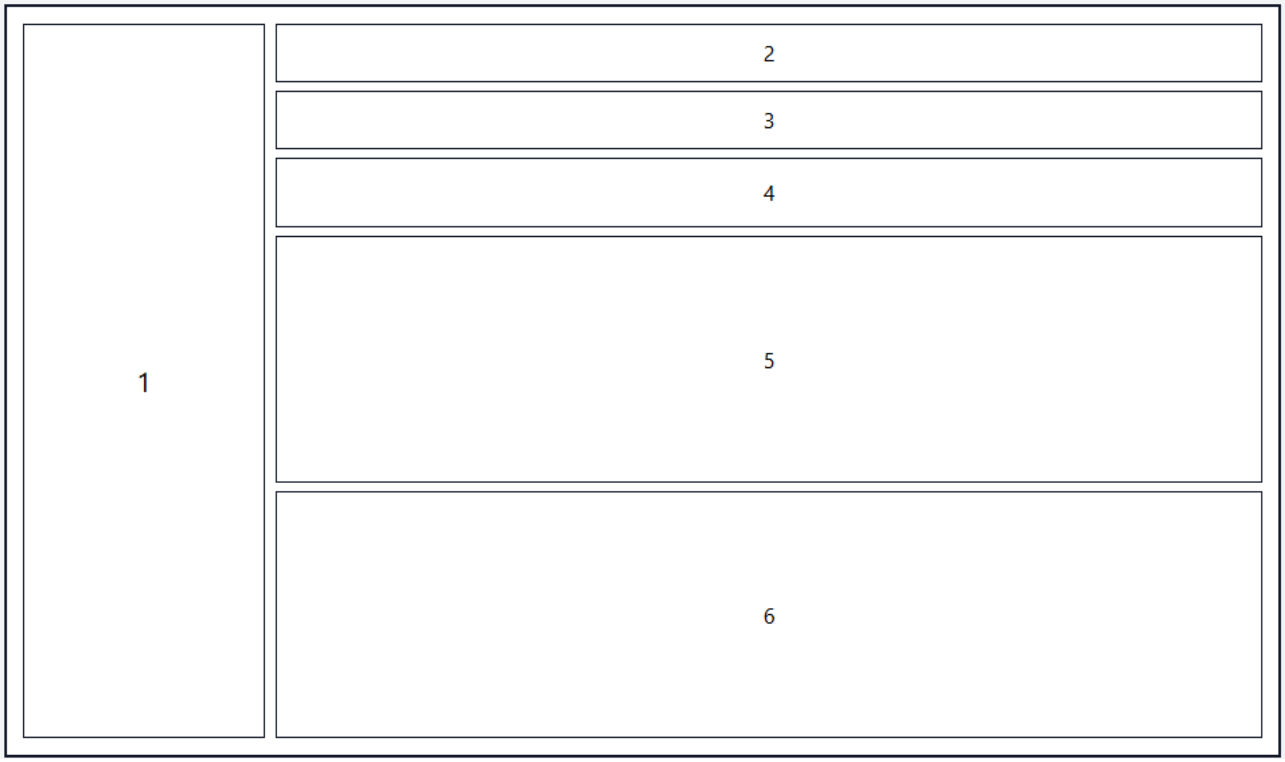

Рисунок 3.18 – Структурна схема сторінки перегляду історії посилок

Елементи сторінки:

1. Блок навігації сторінкою (сайдбар) з посиланнями на основні розділи сервісу.

2. Заголовок сторінки «Історія доставок».

3. Пояснювальний підзаголовок з описом відображення історії як відправника і як перевізника.

4. Панель фільтрації історії доставок за статусом (усі, лише доставлені, лише скасовані) з випадаючим списком і кнопкою «Фільтрувати».

5. Основна панель з історією посилок, які я відправив, у вигляді таблиці та формами оцінки перевізника.

6. Основна панель з історією доставок, які я виконував як перевізник, у вигляді таблиці та формами оцінки замовника.

Інтерфейс сторінки перегляду історії посилок зображено на рисунку 3.19.

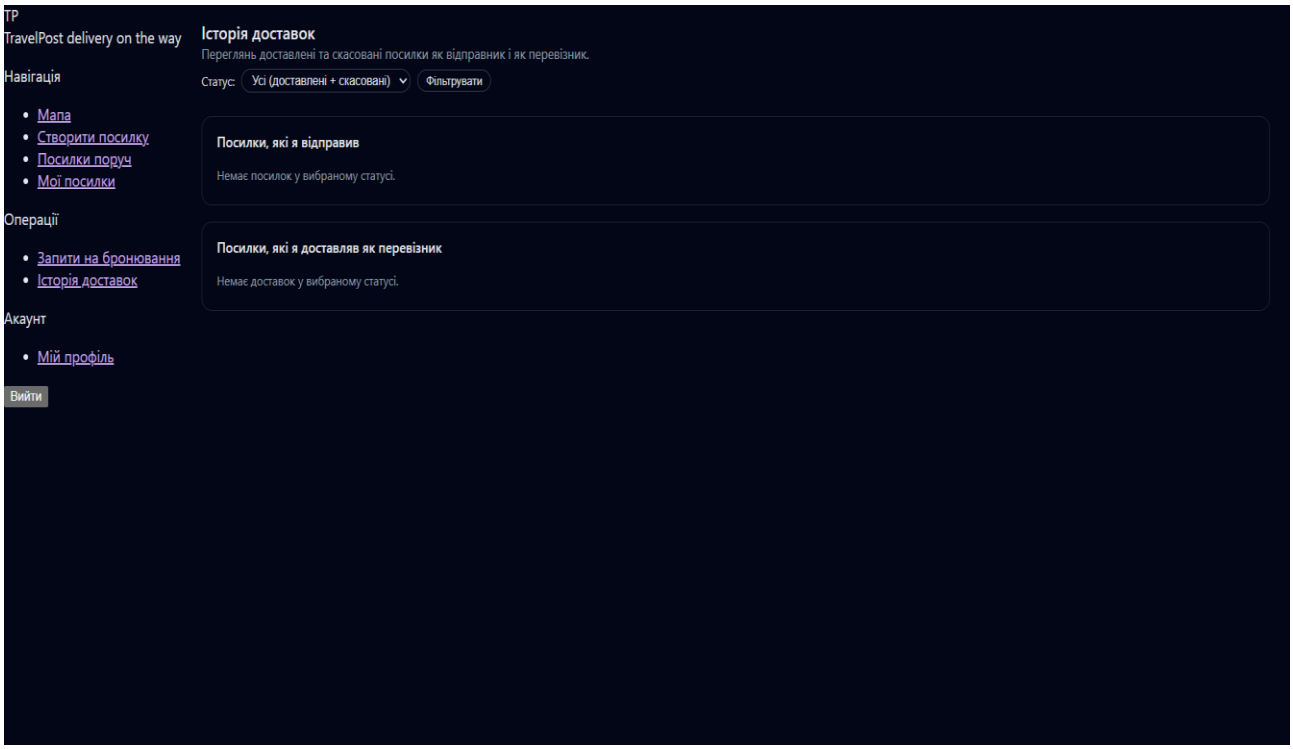

Рисунок 3.19 – Інтерфейс сторінки перегляду історії посилок

Було розроблено інтерфейс сторінки перегляду профілю та статистики користувача, структурна схема якого наведена на рисунку 3.20.

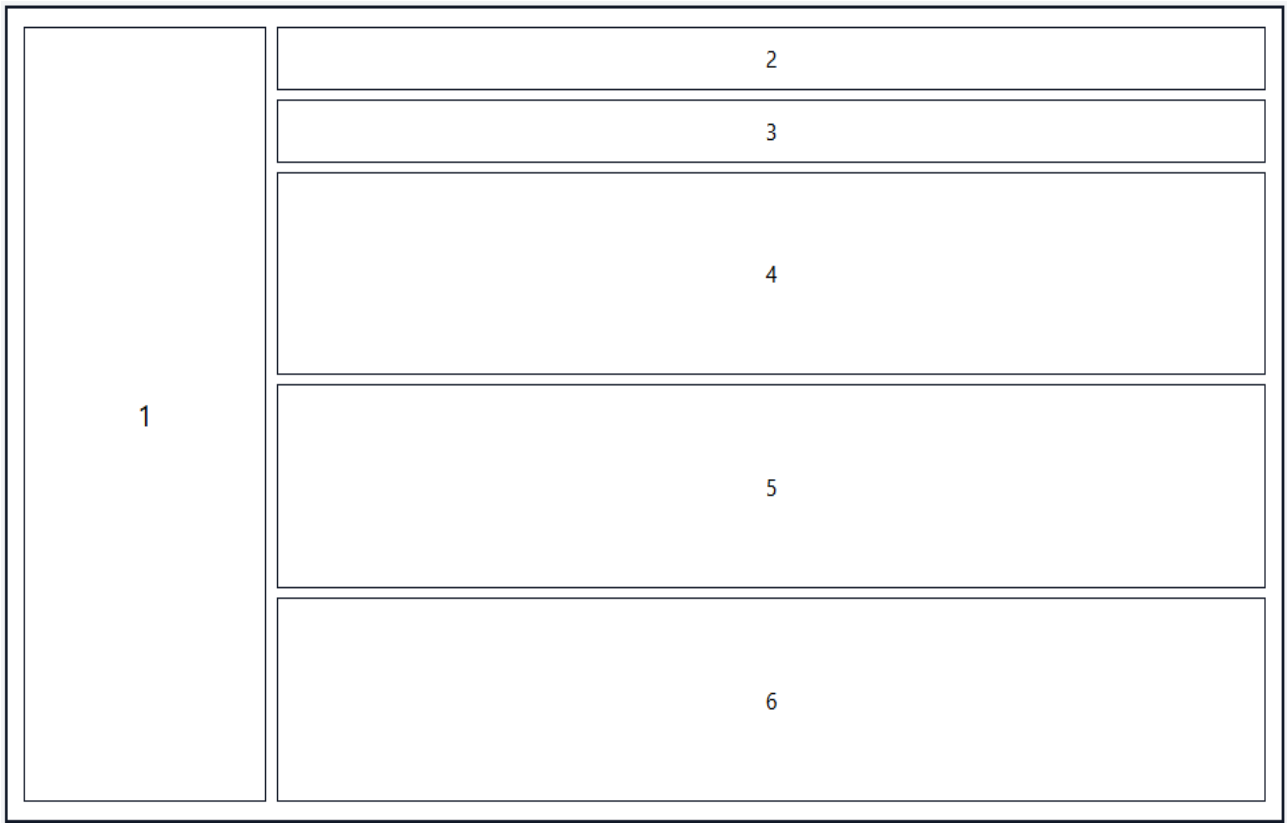

Рисунок 3.20 – Структурна схема сторінки перегляду профілю та статистики користувача

Елементи сторінки:

1. Блок навігації сторінкою (сайдбар) з посиланнями на основні розділи сервісу.

2. Заголовок сторінки «Мій профіль».

3. Пояснювальний підзаголовок про перегляд та оновлення особистих даних, репутації й активності.

4. Панель з особистими даними користувача та формою редагування полів (ім’я, прізвище, телефон).

5. Панель зі статистикою рейтингу та активності (загальний рейтинг, кількість посилок і доставок з розбивкою по статусах).

6. Панель з візуалізацією статистики у вигляді діаграм (розподіл оцінок, статуси посилок та доставок як перевізника).

Інтерфейс сторінки перегляду профілю та статистики користувача зображено на рисунку 3.21.

Рисунок 3.21 – Інтерфейс сторінки перегляду профілю та статистики користувача

3.5 Висновки

У третьому розділі було проведено порівняльний аналіз мов програмування та середовищ розробки, на основі якого обрано оптимальний технологічний стек: мову програмування Java, фреймворк Spring Boot, та шаблонізатор Thymelaf. Це сприяло досягненню високої продуктивності, підвищенню зручності розробки та надало широкі можливості для реалізації користувацького інтерфейсу.

Було розроблено та описано програмні модулі вебсистеми, що дозволило повністю обґрунтувати логіку побудови системи.

Описано можливий спосіб розгортання вебсистеми в контейнеризованому середовищі, що є уніфікованим сучасним стандартом розробки.

Розроблено структурні схеми, макети та повний графічний інтерфейс користувача. У результаті створено вебсистему організованого доставлення посилок «By-the-way», яка забезпечує зручність використання, інтуїтивну взаємодію та високу функціональність.

4 ТЕСТУВАННЯ СИСТЕМИ

4.1 Аналіз методів тестування програмного забезпечення

Тестування веб-застосунку є невід’ємною складовою процесу розробки програмного забезпечення і передбачає систематичну перевірку коректності, надійності, безпеки та продуктивності програмного продукту. Основна ідея тестування полягає в тому, щоб виявити помилки на якомога ранніх етапах життєвого циклу програмного забезпечення, до того як ці дефекти проявляться у реальній експлуатації й негативно вплинуть на кінцевих користувачів. Таким чином, тестування виконує запобіжну роль, мінімізуючи ризики збоїв, втрати даних чи некоректної роботи системи.

Узагальнено тестування можна розглядати як процес перевірки відповідності програмного забезпечення визначеним вимогам і очікуванням користувачів. Воно дає змогу підтвердити, що реалізований функціонал відповідає специфікації, а також оцінити низку важливих характеристик якості: функціональність, зручність використання, надійність, продуктивність, безпеку, сумісність з різними середовищами виконання тощо. Для веб-систем це особливо актуально, оскільки вони зазвичай мають розподілену архітектуру, працюють у мережі Інтернет та обслуговують потенційно велику кількість користувачів [24].

Одним із важливих напрямів перевірки є тестування інтерфейсу користувача. Воно дозволяє оцінити, наскільки інтерфейс є інтуїтивно зрозумілим, чи правильно обробляються введені дані, чи коректно відображається інформація на різних пристроях та у різних браузерах. Фактично, цей тип тестування наближений до реального сценарію використання системи і часто поєднує як функціональні, так і нефункціональні аспекти (зручність використання, час відгуку тощо).

Разом з тим, тестування лише на рівні клієнтської частини є недостатнім. Для веб-застосунків значну роль відіграє серверна логіка, яка відповідає за обробку бізнес-правил, взаємодію з базою даних, інтеграцію з зовнішніми

сервісами, авторизацію та автентифікацію користувачів. Тому доцільно застосовувати методи тестування, спрямовані саме на перевірку коректності роботи внутрішніх компонентів системи. До таких методів належать модульне та інтеграційне тестування, які в сукупності дозволяють поетапно підтверджувати правильність роботи системи – від окремих дрібних фрагментів логіки до їх спільної взаємодії.

Модульне (unit) тестування [25] орієнтоване на перевірку окремих, відносно невеликих одиниць програмного коду – методів, класів, сервісів або інших логічних блоків. Кожен такий блок розглядається ізольовано від решти системи, а зовнішні залежності (наприклад, доступ до бази даних, виклики мережевих сервісів, файлової системи) замінюються на імітації (mock-об’єкти, stubs, фіктивні реалізації). Це дозволяє зосередитися саме на логіці перевірюваного модуля і виключити вплив сторонніх чинників. Результатом модульного тестування є впевненість у тому, що базові обчислення, перевірки умов, обробка виняткових ситуацій та інші внутрішні операції виконуються відповідно до очікувань.

Інтеграційне тестування [26], на відміну від модульного, спрямоване не на перевірку ізольованих фрагментів, а на оцінку того, як окремі компоненти взаємодіють між собою після їх об’єднання. На цьому рівні можуть тестуватися, наприклад, зв’язки між веб-контролерами, сервісами та репозиторіями, коректність транзакційної логіки, робота з реальною (або максимально наближеною до реальної) базою даних, обмін повідомленнями між сервісами тощо. Основна мета інтеграційних тестів полягає в тому, щоб переконатися, що окремо працездатні модулі не вступають у конфлікт один з одним, правильно передають та обробляють дані, а також коректно поводяться у складніших, «ланцюгових» сценаріях.

Обидва ці методи – модульне та інтеграційне тестування – доповнюють один одного та забезпечують різні рівні контролю. Модульні тести зазвичай виконуються дуже швидко, їх легко запускати часто, наприклад при кожній зміні коду або при кожному збірці проєкту у системі безперервної інтеграції.

Інтеграційні тести, як правило, є «важчими» – їх виконання займає більше часу, потребує підготовки середовища (налаштування бази даних, контейнерів, зовнішніх сервісів), однак саме вони дозволяють виявити помилки, пов’язані з конфігурацією, несумісністю інтерфейсів, некоректною взаємодією модулів. Поступове покриття коду модульними та інтеграційними тестами дає можливість створити так званий набір регресійних тестів: після кожної зміни в системі ці тести можна запускати повторно, щоб переконатися, що новий функціонал не порушив роботу вже реалізованих частин.

Застосування описаних методів тестування в сукупності дає змогу підвищити рівень довіри до розроблюваного веб-застосунку, зменшити кількість критичних помилок у процесі експлуатації та спростити подальшу підтримку системи. Ретельно спроєктований набір тестів, що охоплює як користувацькі сценарії, так і внутрішню бізнес-логіку, сприяє підвищенню якості продукту, полегшує внесення змін та розширення функціоналу в майбутньому.

4.2 Модульне тестування серверної частини розроблюваної системи

Для написання модульних тестів в Spring-проектах зазвичай використовують такі бібліотеки як Mockito [27] та Junit [28]. Вони дозволяють підмінити всі зовнішні залежності тестованого класу на шаблони, які будуть повертати стандартні несправжні значення при виклику методів таких класів. Також цю поведінку можна конфігурувати та задати бажаний сценарій, щоб при виклику методу з певними параметрами, повертався завчасно підготовлений результат, або навіть викликалася помилка, щоб перевірити систему обробки помилок. У поєднанні з можливостями JUnit для організації тестових наборів, перевірки очікуваних результатів та винятків та інтеграції зі Spring Test Framework, такий підхід забезпечує систематичне і кероване тестування окремих компонентів застосунку.

Приклад створеного модульного тесту для сценарію створення бронювання зображений на рисунку 4.1.

```java
@Test  new *
void createBooking_success() {
    BigDecimal offeredPrice = new BigDecimal( val: "100.50");
    String comment = "Handle with care";

    //Підготовка
    when(parcelRepository.findById(PARCEL_ID)).thenReturn(Optional.of(parcel));
    when(bookingRepository.existsByParcelIdAndStatus(PARCEL_ID, BookingStatus.ACCEPTED)).thenReturn( t: false);
    when(userRepository.findByEmail(CARRIER_EMAIL)).thenReturn(Optional.of(carrier));
    when(bookingRepository.save(any(Booking.class)))
            .thenAnswer(invocation -> invocation.getArgument( i: 0));

    //Виконання
    Booking result = bookingService.createBooking(PARCEL_ID, CARRIER_EMAIL, offeredPrice, comment);

    ArgumentCaptor<Booking> bookingCaptor = ArgumentCaptor.forClass(Booking.class);
    verify(bookingRepository).save(bookingCaptor.capture());
    Booking saved = bookingCaptor.getValue();

    //Перевірка
    assertSame(parcel, saved.getParcel());
    assertSame(carrier, saved.getCarrier());
    assertEquals(BookingStatus.REQUESTED, saved.getStatus());
    assertEquals(offeredPrice, saved.getOfferedPrice());
    assertEquals(comment, saved.getCarrierComment());
    assertSame(saved, result);
}
```

Рисунок 4.1 – Модульний тест для сценарію успішного створення бронювання

З цього тесту дуже гарно видно загальний підхід до написання модульних тестів – воно відбувається в 3 дії:

1. Підготовка – задаємо бажану поведінку для зовнішніх залежностей;
2. Виконання – виклик методу, що тестується;
3. Перевірка – порівняння фактичного результату роботи методу з очікуваним.

Результат виконання модульних тестів для класу BookingService зображений на рисунку 4.2.

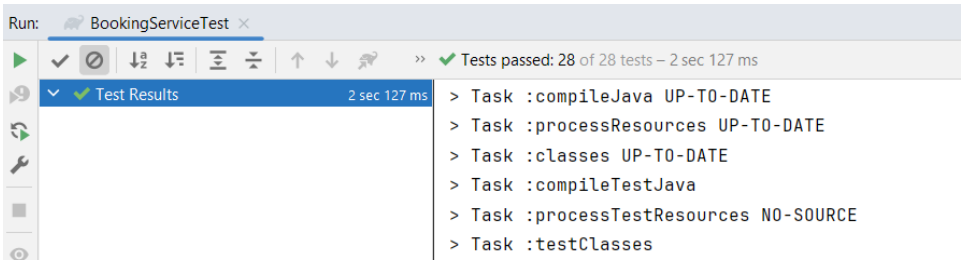

Рисунок 4.2 – Результат виконання модульних тестів

4.3 Інтеграційне тестування серверної частини розроблюваної системи

Важливим аспектом написання інтеграційних тестів є чітке формулювання того, наскільки глибоко має тестуватися взаємодія сервісів між

собою. Мається на увазі, що для найпростіших інтеграційних тестів можна розглядати взаємодію двох класів та підмінити всі інші зовнішні залежності цих класів на шаблони, як це робиться в модульному тестуванні.

Для більш просунутих тестів можна тестувати цілу функцію системи – всі взаємодії окремих класів та модулів будуть справжніми – надходження HTTP-запиту в контролер, опрацювання даних, звертання до сторонніх сервісів, робота з базою даних.

Головною відмінністю від справжнього запуску застосунку є те, що кожний тест має бути ізольований один від одного, щоб результати одного тесту не впливали на виконання іншого, такі тести називають тести чистого контексту.

А також, база даних та сторонні сервіси не є повністю справжніми. Для того, щоб не створювати на них додаткове навантаження та вберегти від помилок в результаті виконання тестів, використовують спеціальні окремі тестові версії, або ж їх очікувана поведінка конфігурується, як це робиться в модульному тестуванні.

Один з інтеграційних тестів розроблюваної системи зображений на рисунку 4.3.

```
68      @Test  new *
69      void getRoutePolyline_success_returnsEncodedPolylineFromOsrm() throws Exception {
70          String startAddress = "Kyiv, Khreschatyk 1";
71          String endAddress = "Kyiv, Maidan Nezalezhnosti";
72
73          // 1) Стаб для geocoding
74          wireMockServer.stubFor(get(urlPathEqualTo("/geocode"))
75                  .withQueryParam("api_key", equalTo("test-api-key"))
76                  .withQueryParam("q", equalTo(startAddress))
77                  .willReturn(okJson("""
78                          [{
79                            "lat": "50.450100",
80                            "lon": "30.523400"
81                          }]
82                          """)));
83
84          // 3) Стаб для OSRM — будь-який маршрут (патерн по шляху)
85          String expectedPolyline = "encoded_polyline_123";
86          wireMockServer.stubFor(get(urlPathMatching("/route/v1/driving/.*"))
87                  .willReturn(okJson("""
88                          {
89                            "code": "Ok",
90                            "routes": [
91                              { "geometry": "encoded_polyline_123" }
92                            ]
93                          }
94                          """)));
95
96          // Виклик ендпоінта
97          mockMvc.perform(get( uriTemplate: "/polyline")
98                          .param( name: "startAddress", startAddress)
99                          .param( name: "endAddress", endAddress))
100                 .andExpect(status().isOk())
101                 .andExpect(content().contentTypeCompatibleWith("application/json"))
102                 .andExpect(jsonPath( expression: "$.encodedPolyline").value(expectedPolyline));
103     }
```

Рисунок 4.3 – Приклад інтеграційного тесту

4.4 Тестування розроблюваної системи за допомогою інтерфейсу

Для тестування системи за допомогою інтерфейсу було складено такі тест-кейси:

1. Спроба входу в систему без даних для входу.
2. Спроба входу в систему з неправильним паролем.
3. Вхід в систему з правильними даними для входу.
4. Спроба побудови маршруту з неіснуючою адресою.
5. Побудова маршруту з існуючою адресою.
6. Створення посилки.
7. Знаходження посилки в заданому радіусі від маршруту.

Спроба входу в систему без даних для входу. Для цього відкриємо інтерфейс системи, на якому за замовчуванням одразу відкриється екран для логіну в систему. Екран для логіну в систему зображено на рисунку 4.4.

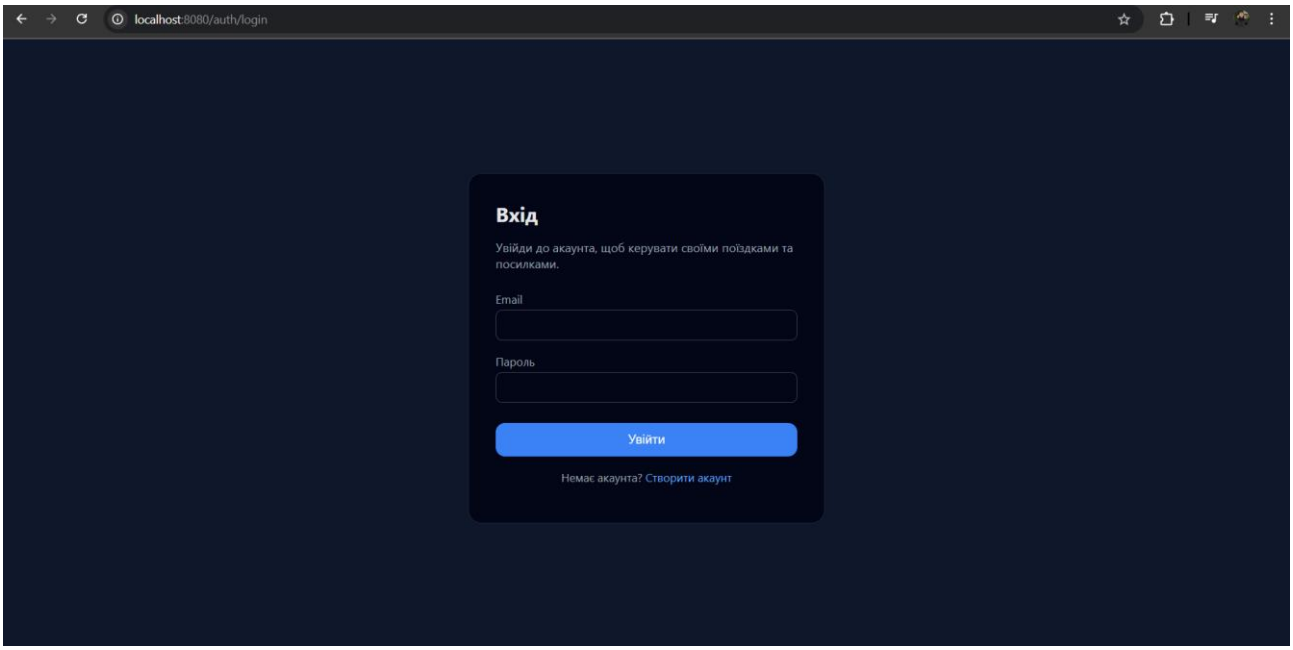

Рисунок 4.4 – Екран для логіну в систему

Після цього спробуємо натиснути на кнопку для входу, не вводячи жодних даних, щоб перевірити систему на стійкість до такого типу вразливостей. Результат виконання логіну без даних для входу зображено на рисунку 4.5.

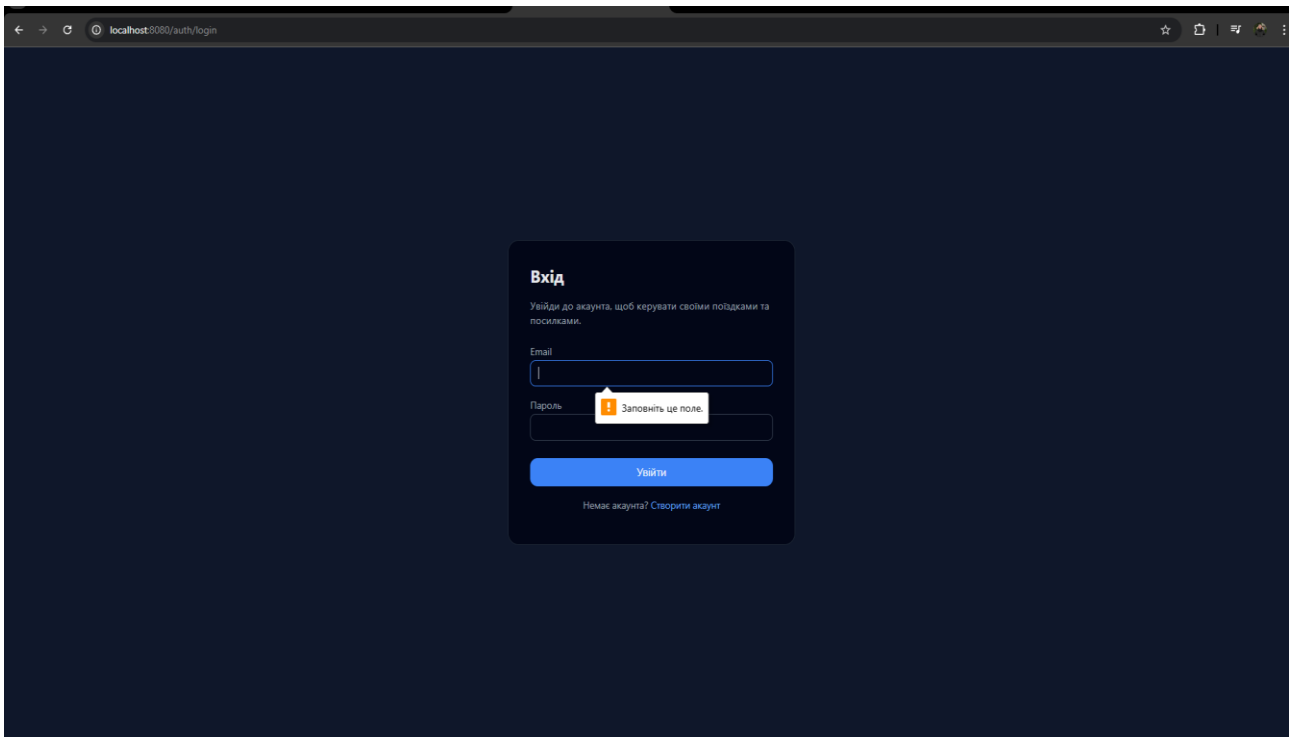

Рисунок 4.5 – Результат виконання логіну без даних для входу

При спробі входу без заповнення полів, виводить повідомлення про те, яке саме поле не було заповнене та є неможливим вхід в систему.

Спроба входу в систему з неправильним паролем. Для цього тесту потрібно завчасно створити користувача в системі, Це можна побачити з повідомлення про успішну реєстрацію, що зображене на рисунку 4.6.

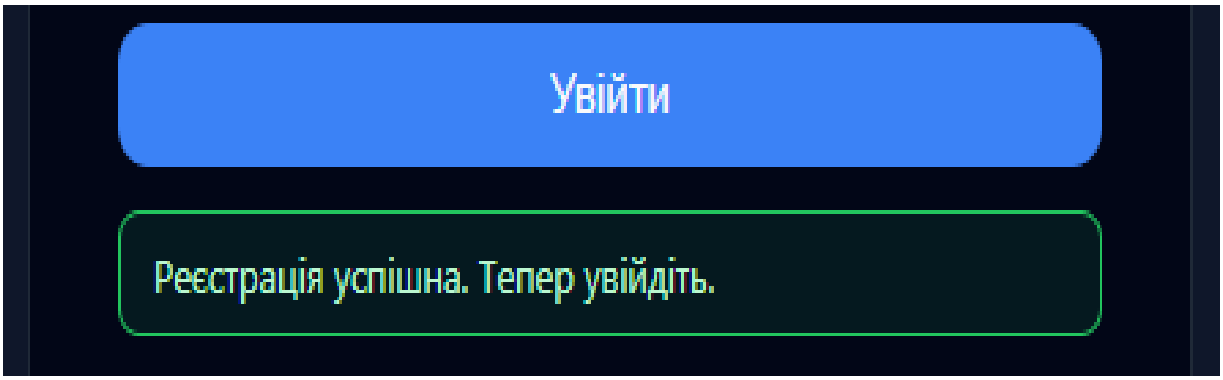

Рисунок 4.6 – Повідомлення про успішну реєстрацію

Тепер вписуємо правильну пошту, але довільний пароль. Результат виконання тесту зображено на рисунку 4.7.

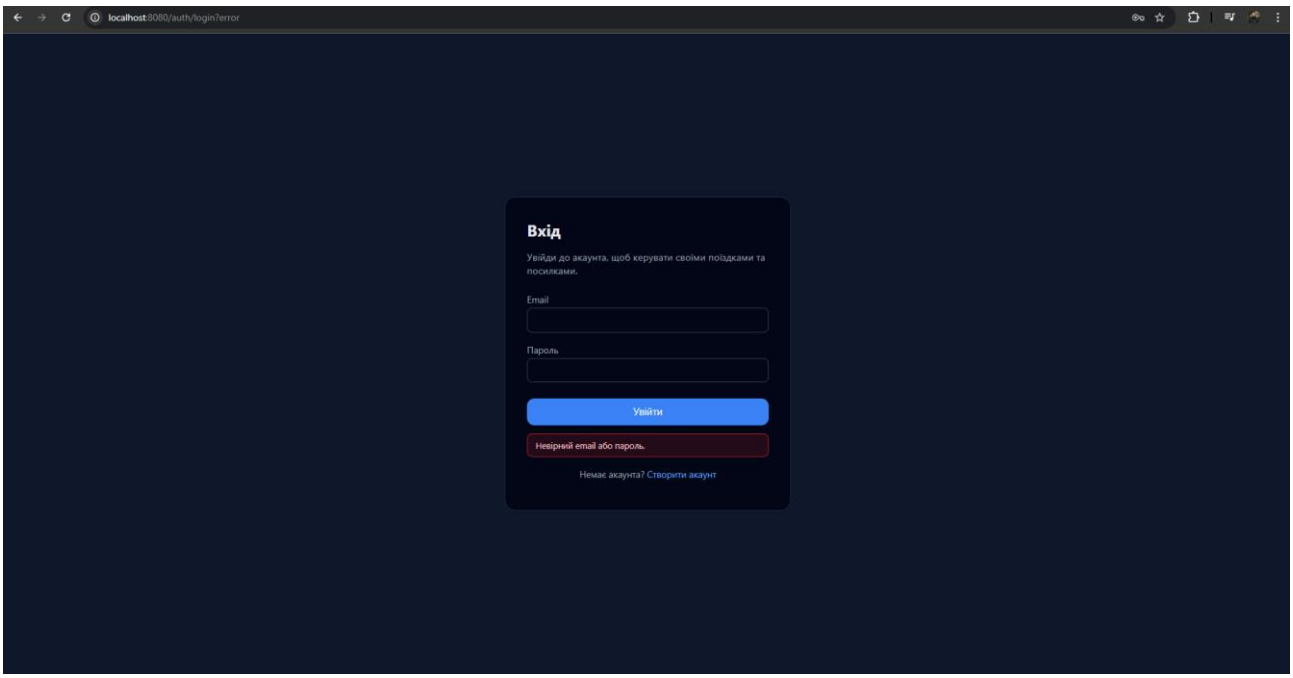

Рисунок 4.7 – Результат вводу неправильного паролю

Система виводить повідомлення про те, що дані для входу є неправильними, таким чином надає користувачу зрозумілий зворотній зв'язок.

Вхід в систему з правильними даними для входу. Для цього тесту потрібно попередньо зареєструвати користувача в системі та ввести правильні дані в форму для логіну. Результат виконання зображений на рисунку 4.8.

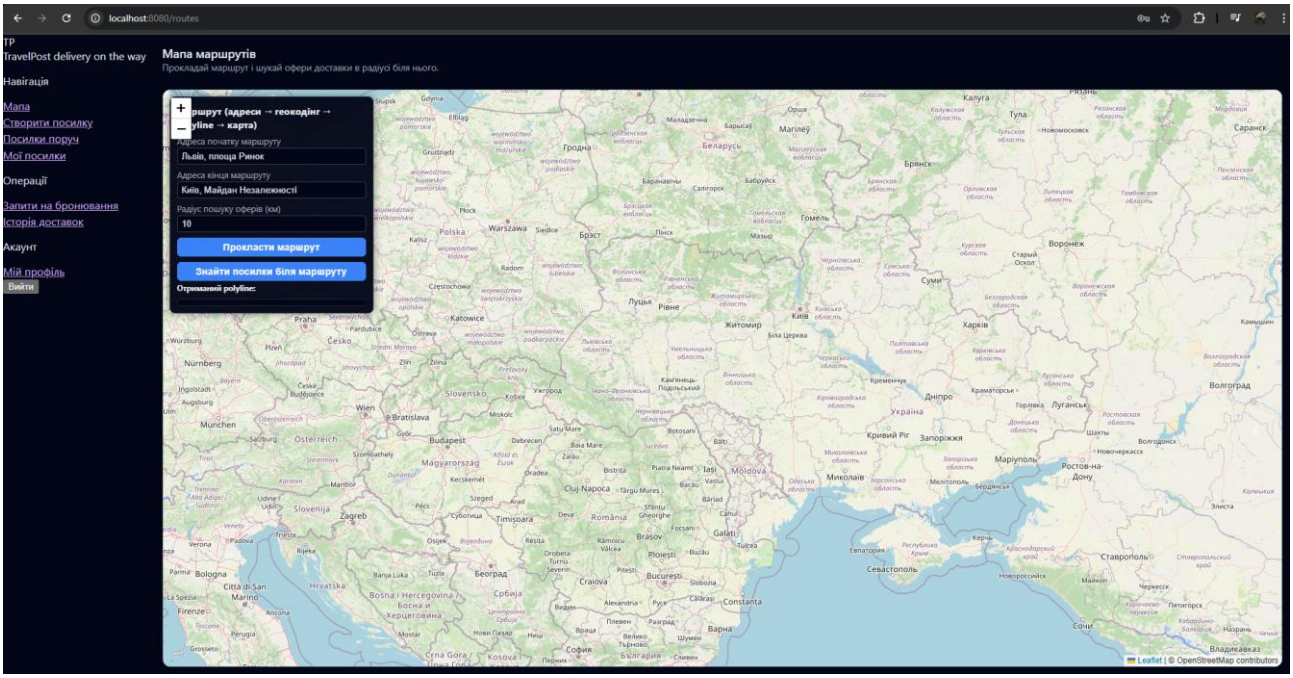

Рисунок 4.8 – Результат успішного логіну в систему

В результаті користувача було перенаправлено на головний екран системи, де він має змогу взаємодіяти зі всіма функціями.

Спроба побудови маршруту з неіснуючою адресою. Однією з ключових функцій застосунку є правильна робота з адресами та обробка помилок, щоб дати змогу користувачу зрозуміти, якщо він щось зробив неправильно. Так як адреса приймається довільним текстом, то користувачу легко помилитися та вписати щось зайве, або неправильне. Зімітуємо ситуацію, коли користувач забув змінити обрану мову вводу на клавіатурі та вводить адресу. Форма вводу зображена на рисунку 4.9.

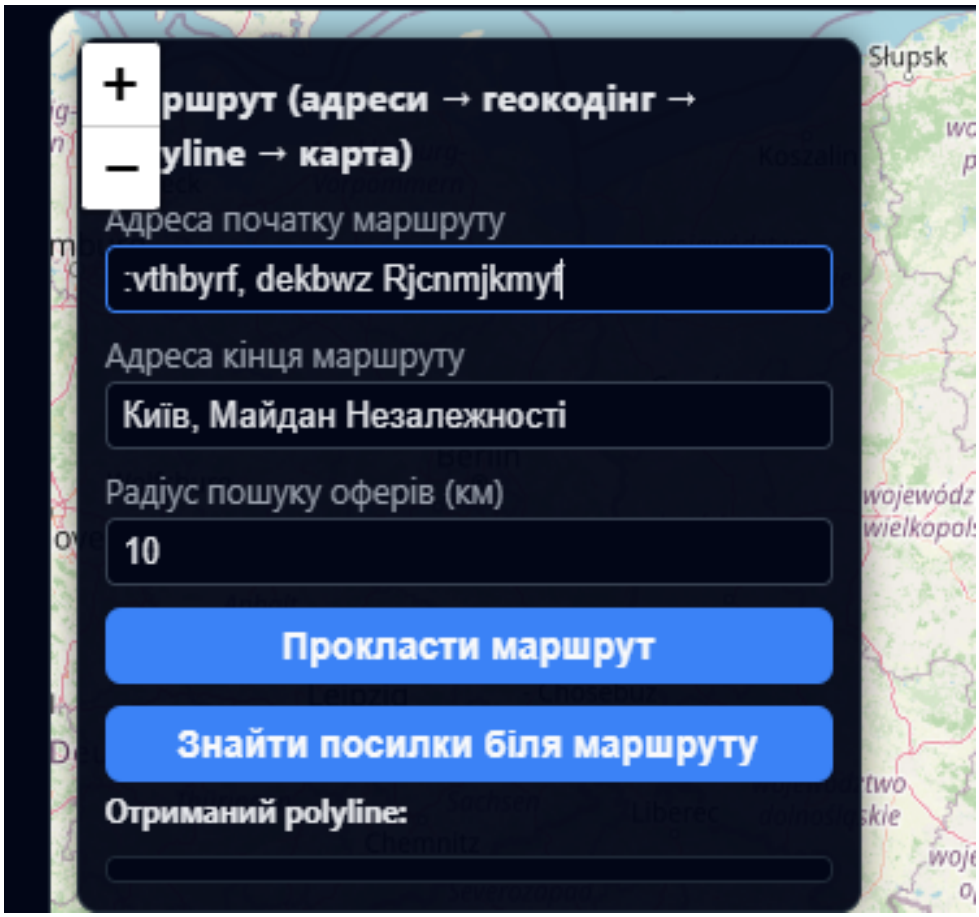

Рисунок 4.9 – Форма вводу адрес з неправильним вводом

Як результат система повертає повідомлення про помилку користувача, що надає детальну інформацію про те, що потрібно змінити. Повідомлення про помилку зображене на рисунку 4.10.

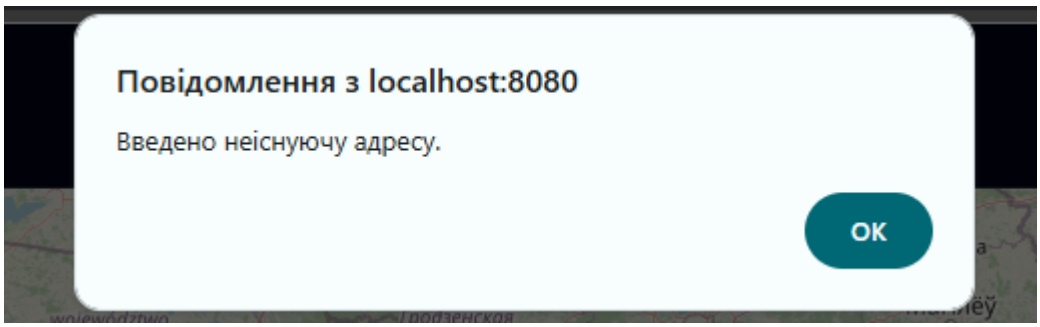

Рисунок 4.10 – Повідомлення про помилку

Побудова маршруту з існуючою адресою. Якщо вписати правильну інформацію про адреси, то застосунок повинен побудувати маршрут між початковою та кінцевою адресами та відобразити його на карті. Форма вводу пошуку маршруту зображена на рисунку 4.11.

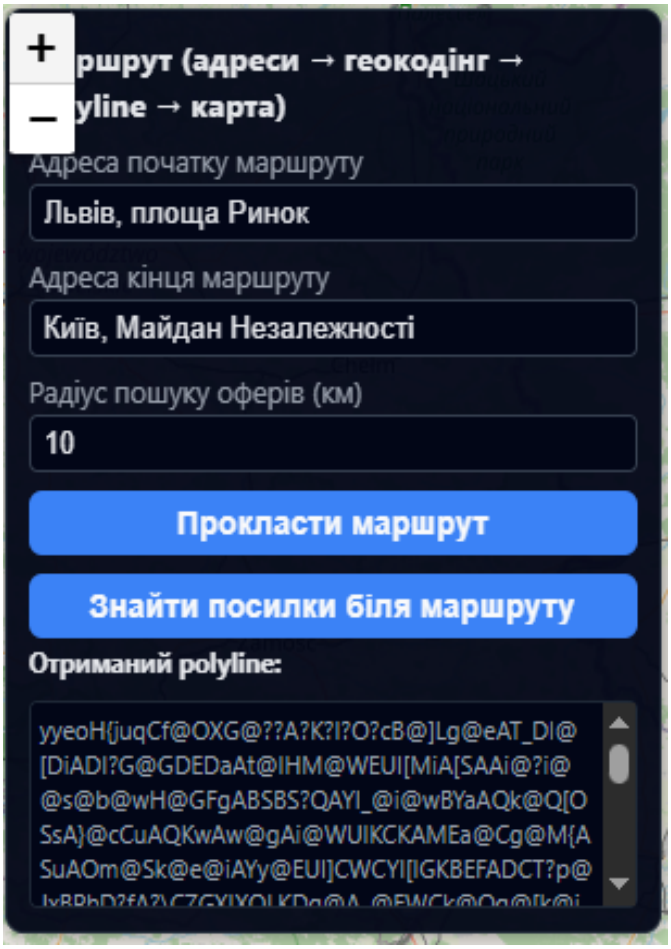

Рисунок 4.11 – Форма вводу адрес з правильним можливим вводом

Результат функції прокладання маршруту зображено на рисунку 4.12.

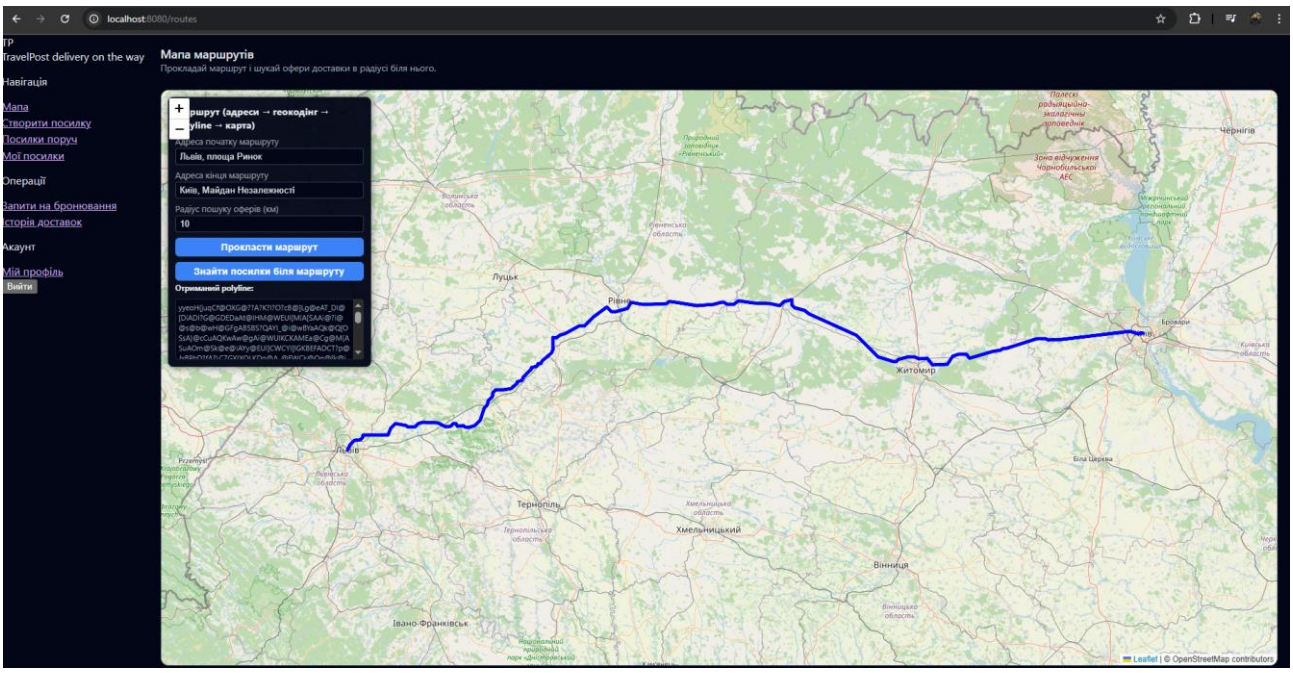

Рисунок 4.12 – Приклад візуалізації маршруту на карті

Створення посилки. Посилки відіграють ключову роль в моделі системи – їх можна створювати, шукати, бронювати пропозиції їх доставки тощо. Тому необхідно перевірити функцію їх успішного створення та збереження в системі.

Форма для створення посилки з заповненими даними зображена на рисунку 4.13.

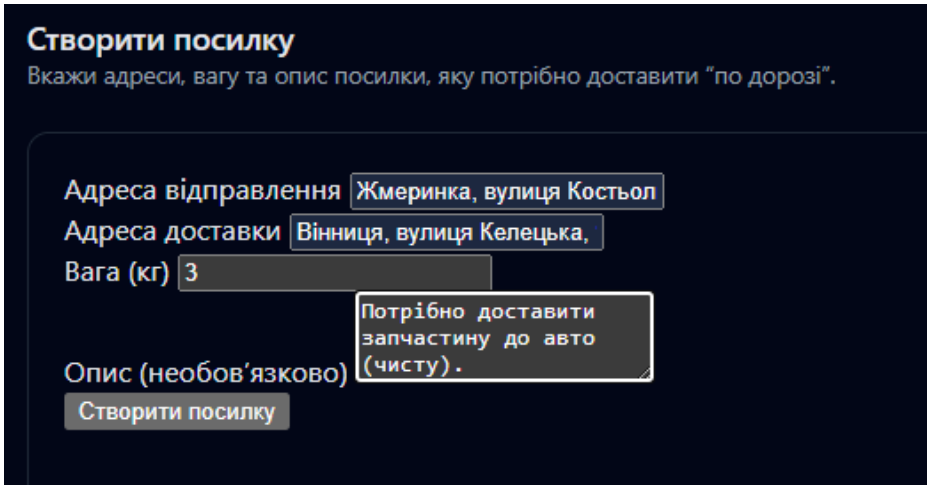

Рисунок 4.13 – Форма для створення посилки з заповненими даними

В результаті натиснення на кнопку створення, дані про посилку зберігаються в системі. Результат збереження даних про посилку в системі

зображено на рисунку 4.14.

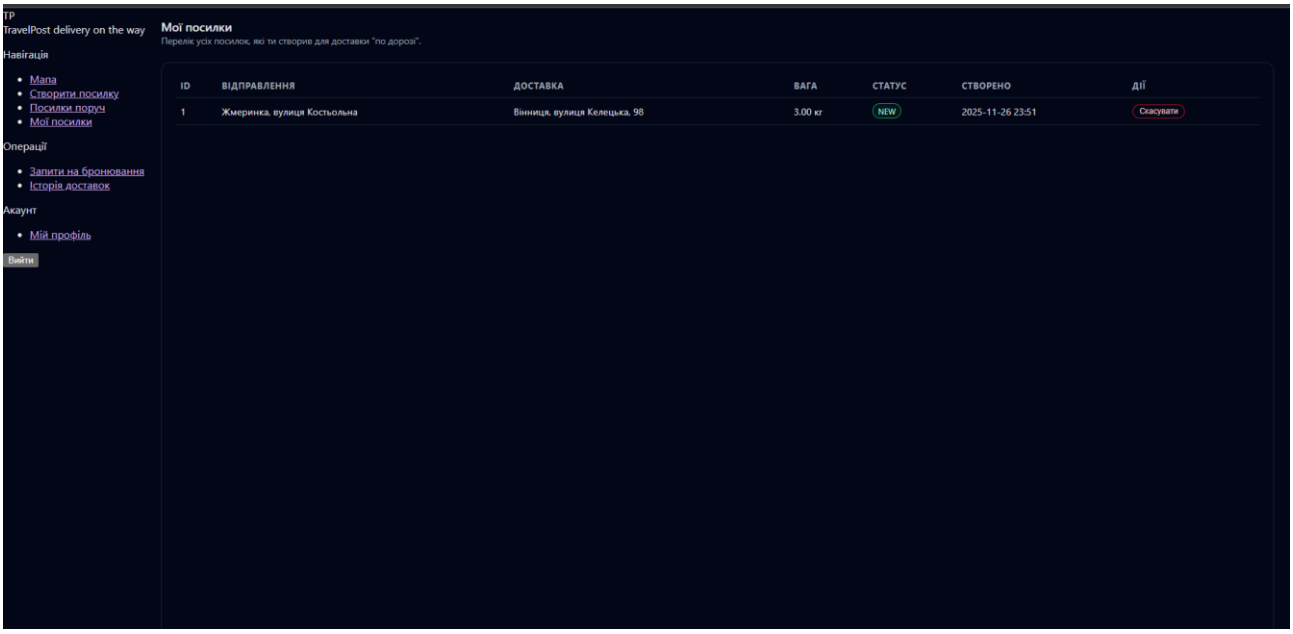

Рисунок 4.14 – Результат збереження даних про посилку в системі

Знаходження посилки в заданому радіусі від маршруту. Після того як посилку було створено, перевізники мають мати можливість її знайти. Для цього було розроблено функцію пошуку доступних посилок в вказаному радіусі на протязі всього маршруту. Результат пошуку зображений на рисунку 4.15.

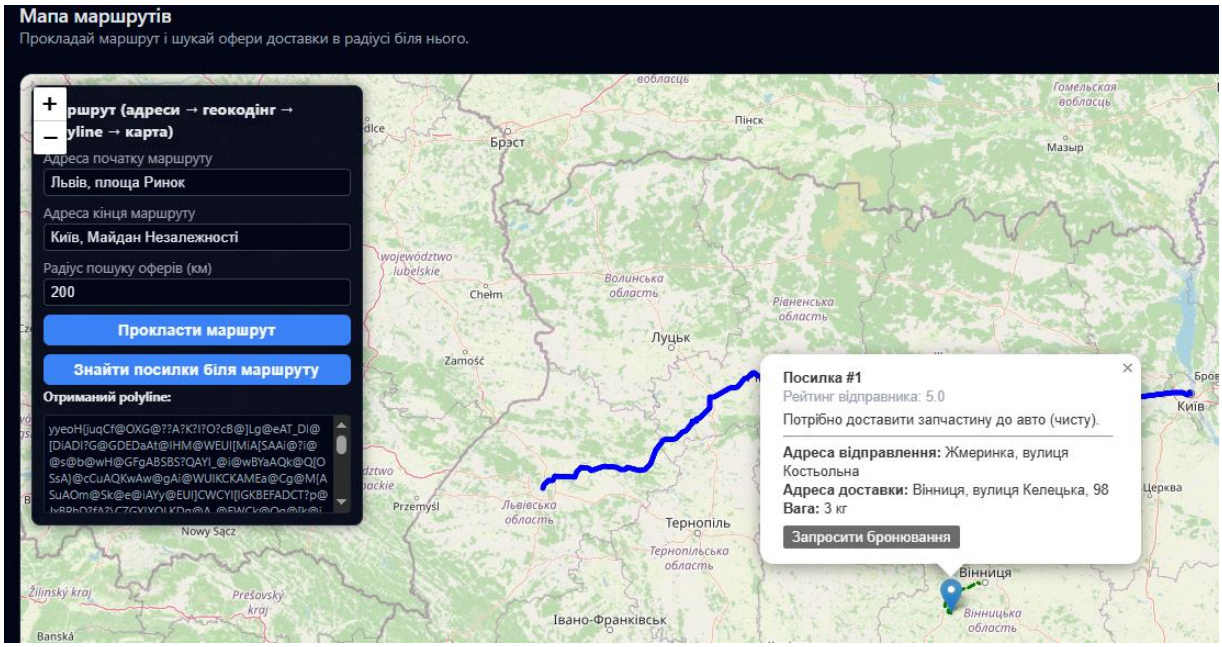

Рисунок 4.15 – Результат пошуку посилок в системі

Пошук за радіусом розроблений такий чином, що місце отримання та доставки посилки має бути на відстані не більшій за вказану на хоча б одній з точок маршруту. Це унеможливлює ситуацію, коли система пропонує посилки, початкові адреси яких знаходять поруч з початком маршруту перевізника, але кінцева точка знаходиться на великій відстані від запланованого маршруту, що приносило б незручності для перевізника.

4.5 Висновки

У четвертому розділі було проведено аналіз основних видів програмного тестування, також було описано обрані види тестування – написання модульних та інтеграційних тестів, що забезпечує стійкість функцій системи до регресії в наслідок створення нових функцій. Також було протестовано роботу системи окремих функцій системи разом з інтерфейсом, який відповідає за ці функції, що підтвердило коректність реалізації функціоналу та стабільність роботи системи.

5 ЕКОНОМІЧНА ЧАСТИНА

Науково-технічна розробка має сенс та може бути впроваджена лише за умови, що вона відповідає актуальним вимогам часу, підтримуючи науково-технічний прогрес і враховуючи економічні аспекти. Тому для науково-дослідної роботи необхідно оцінювати економічну ефективність результатів виконаної роботи.

Магістерська кваліфікаційна робота за темою «Розробка методів і програмних засобів вебсистеми організованого доставлення посилок» відноситься до науково-технічних робіт, які орієнтовані на виведення на ринок (або рішення про виведення науково-технічної розробки на ринок може бути прийнято у процесі проведення самої роботи).

Цей напрямок є пріоритетним, оскільки вебсистеми організованого доставлення посилок дозволяють користувачам керувати даними своїї відправлень, отримувати інформацію про посилки та виконувати пошук координат за текстовою адресою.

Однак, задля реалізації цього проекту, необхідно довести його економічну доцільність, щоб знайти інвесторів, які будуть готові взяти участь у реалізації проекту.

Для наведеного випадку мають бути виконані такі етапи робіт:

1) комерційний аудит науково-технічної розробки, тобто встановлення її науково-технічного рівня та комерційного потенціалу;

2) розраховано витрати на здійснення науково-технічної розробки;

3) розрахована економічна ефективність науково-технічної розробки у випадку її впровадження і комерціалізації потенційним інвестором і проведено обґрунтування економічної доцільності комерціалізації інвестором.

5.1 Проведення комерційного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою «Розробка методів і програмних засобів вебсистеми організованого

доставлення посилок» є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями, наведеними в таблиці 5.1 [29].

Таблиця 5.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в реальних умовах
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі влас-тивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів

Продовження таблиці 5.1

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування

Продовження таблиці 5.1

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь- які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки було зведено до таблиці 5.2.

Таблиця 5.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада))		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	4	4	4
2. Ринкові переваги (наявність аналогів)	2	2	2
3. Ринкові переваги (ціна продукту)	3	3	3
4. Ринкові переваги (технічні властивості)	4	4	4
5. Ринкові переваги (експлуатаційні витрати)	4	4	3
6. Ринкові перспективи (розмір ринку)	3	3	2
7. Ринкові перспективи (конкуренція)	4	3	4
8. Практична здійсненність (наявність фахівців)	4	4	4
9. Практична здійсненність (наявність фінансів)	3	4	3
10. Практична здійсненність (необхідність нових матеріалів)	4	3	3
11. Практична здійсненність (термін реалізації)	3	3	4
12. Практична здійсненність (розробка документів)	4	4	4
Сума балів	42	41	40
Середньоарифметична сума балів $СБ_c$	41		

За результатами розрахунків, наведених в таблиці 5.2, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в таблиці 5.3 [30].

Таблиця 5.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ ,розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Розробка методів і програмних засобів вебсистеми організованого доставлення посилок» становить 41 бал, що, відповідно до таблиці 5.3, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки високий).

5.2 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему «Розробка методів і програмних засобів вебсистеми організованого доставлення посилок», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

5.2.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою 5.1 [30]:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \quad (5.1)$$

де k – кількість посад дослідників залучених до процесу досліджень;

M_{ni} – місячний посадовий оклад конкретного дослідника, грн;

t_i – число днів роботи конкретного дослідника, дні;

T_p – середнє число робочих днів в місяці, T_p=22 дні.

$$З_о = 30000 \cdot 60 / 22 = 81818{,}18 \text{ грн.}$$

Проведені розрахунки зведено до таблиці 5.4.

Таблиця 5.4 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проекту	32500	1477,273	60	88636,36
Інженер-розробник програмного забезпечення	25000	1136,364	60	68181,82
Консультант	19500	886,3636	30	26590,91
Дизайнер користувацького інтерфейсу	22500	1022,727	20	20454,55
Всього				203863,6

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР розраховуємо за формулою 5.2:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i, \quad (5.2)$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою 5.3:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}}, \qquad (5.3)$$

де M_M – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо *Мм*=8000,00 грн;

K_i – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду [29];

K_c – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об'єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

T_p – середнє число робочих днів в місяці, приблизно T_p = 22 дні;

$t_{зм}$ – тривалість зміни, год.

$$C_1 = 8000{,}00 \cdot 1{,}09 \cdot 1{,}65 / (22*8) = 81{,}75 \text{ грн.}$$

$$З_{р1} = 82{,}5 \cdot 8{,}00 = 654 \text{ грн.}$$

Величина витрат на заробітну плату робітників наведена в таблиці 5.5.

Таблиця 5.5 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника, грн
Підготовка робочого місця дослідника	8	2	1,09	81,75	654

Продовження таблиці 5.5

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника, грн
Інсталяція програмного забезпечення	3	5	1,36	272	816
Підключення апаратного забезпечення	8	3	1,18	88,5	708
Всього					2178

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою 5.4:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \qquad (5.4)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 11%.

$З_{дод} = (203863,6 + 2178) \cdot 11 / 100\% = 22664,58$ грн.

5.2.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою 5.5:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \qquad (5.5)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

$Зн = (203863,6 + 2178 + 22664,58) \cdot 22 / 100\% = 50315,36$ грн.

5.2.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень.

Витрати на матеріали (*М*), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою 5.6:

$$M = \sum_{j=1}^{n} H_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} B_j \cdot Ц_{вj}, \qquad (5.6)$$

де H_j – норма витрат матеріалу *j*-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу *j*-го найменування, грн/кг;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ = *1,1 ... 1,15*);

B_j – маса відходів *j*-го найменування, кг;

$Ц_{вj}$ – вартість відходів *j*-го найменування, грн/кг.

$$М_1 = 2 \cdot 208{,}00 \cdot 1{,}15 - 0{,}000 \cdot 0{,}00 = 478{,}4 \text{ грн.}$$

Проведені розрахунки зведемо до таблиці 5.6.

Таблиця 5.6 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за од, грн	Норма витрат, од	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Папір офісний "Crystal Pro" PRO80 A4 80 г/м білий 500 аркушів	208	2	0	0	478,4
Канцелярське приладдя	300	3	0	0	1035
Картридж для принтера HP LaserJet P1102 Black	625	1	0	0	718,75

Продовження таблиці 5.6

Найменування матеріалу, марка, тип, сорт	Ціна за од	Норма витрат, од	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Флеш пам'ять USB Kingston DataTraveler Exodia 64GB	189	1	0	0	217,35
Папка-бокс пластикова Economix A4 60 мм на гумці чорна	93	1	0	0	106,95
Всього					2556,45

5.2.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі *($К_в$)*, які використовують при проведенні НДР на тему «Розробка методів і програмних засобів вебсистеми організованого доставлення посилок» відсутні.

5.2.5 Спецустаткування для наукових (експериментальних) робітника

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування необхідного для проведення досліджень, також витрати на їх проектування, виготовлення, транспортування, монтаж та встановлення.

Витрати на спецустаткування, які використовують при проведенні НДР на тему «Розробка методів і програмних засобів вебсистеми організованого доставлення посилок» відсутні.

5.2.6 Програмне забезпечення для наукових (експериментальних) робіт

До статті «Програмне забезпечення для наукових (експериментальних)

робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, (програм, алгоритмів, баз даних) необхідних для проведення досліджень, також витрати на їх проектування, формування та встановлення.

Балансову вартість програмного забезпечення розраховуємо за формулою 5.8:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i , \qquad (5.8)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, ($K_i = 1,10…1,12$);

k – кількість найменувань програмних засобів.

$B_{прг1} = 7700 \cdot 4 \cdot 1,12 = 34496$ грн.

Отримані результати зведемо до таблиці 5.7.

Таблиця 5.7 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
ОС Windows 11 Professional	4	7700	34496
Прикладний пакет Microsoft 365 Premium	4	9099	40763,5
Сервіс розробки інтерфейсів Figma	1	842	943
Всього			76202,5

5.2.7 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою 5.9:

$$А_{обл} = \frac{Ц_б}{Т_в} \cdot \frac{t_{вик}}{12}, \qquad (5.9)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$Т_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$А_{обл} = (140000 \cdot 3) / (3 \cdot 12) = 11666,7$ *грн.*

Проведені розрахунки зведемо до таблиці 5.8.

Таблиця 5.8 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Персональний ноутбук (4 шт.)	140000	3	3	11666,7
Робоче місце дослідника	18000	5	3	900
Оргтехніка	7500	4	3	468,75
ОС Windows 11 (4 шт.)	34496	2	3	4312
Microsoft 365 Premium (4 шт.)	40763,5	2	3	5095,4

Продовження таблиці 5.8

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Сервіс розробки інтерфейсів Figma	943	1	3	235,75
Всього				22678,6

5.2.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію (B_e) розраховуємо за формулою 5.10:

$$\text{В}_\text{е} = \sum_{i=1}^{n} \frac{W_{\text{уі}} \cdot t_i \cdot \text{Ц}_\text{е} \cdot \text{К}_{\text{впі}}}{\eta_i}, \quad (5.10)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_е$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $\text{Ц}_\text{е} = 8{,}4$ грн;

$К_{впі}$ – коефіцієнт, що враховує використання потужності, $К_{впі} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$B_e = 4 \cdot 0{,}25 \cdot 480{,}0 \cdot 8{,}4 \cdot 0{,}95 / 0{,}97 = 3948{,}87$ *грн.*

Проведені розрахунки зведемо до таблиці 5.9.

Таблиця 5.9 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Персональний ноутбук (4 шт.)	0,25	480	3948,87

Продовження таблиці 5.9

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Робоче місце дослідника	0,2	480	789,77
Оргтехніка	0,4	16	52,65
Всього			4791,29

5.2.9 Службові відрядження

До статті «Службові відрядження» дослідної роботи належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов'язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з'їзди, конференції, наради, пов'язані з виконанням конкретних досліджень.

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою 5.11:

$$B_{св} = (З_о + З_р) \cdot \frac{H_{св}}{100\%}, \qquad (5.11)$$

де $H_{св}$ – норма нарахування за статтею «Службові відрядження», приймемо $Н_{св} = 20\%$.

$B_{св}$ = 203863,6 + 2178) · 20 / 100% =41208,32 грн.

5.2.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» розраховуємо як 30…45% від суми основної заробітної плати дослідників та робітників за формулою 5.12:

$$B_{сп} = (З_о + З_р) \cdot \frac{Н_{сп}}{100\%}, \quad (5.12)$$

де $Н_{сп}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації», приймемо $Н_{сп}$= 35%.

$В_{сп}$ = (203863,6+ 2178) · 35 / 100% = 72114,56 грн.

5.2.11 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою 5.13:

$$І_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (5.13)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 55%.

$І_в$ = (203863,6+ 2178) · 55 / 100% = 113322,88 грн.

5.2.12 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою 5.14:

$$B_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.14)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 120%.

$В_{нзв}$ = (203863,6 + 2178) · 100 / 120% = 171701,33 грн.

Витрати на проведення науково-дослідної роботи розраховуємо як суму всіх попередніх статей витрат за формулою 5.15:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв}. \quad (5.15)$$

$В_{заг}$=203863,6+2178+22664,58+50315,36+2556,45+0,00+0,00+76202,5+22678,6+4791,29+72114,56+41208,32+112233,9+171701,33= 783597,5 грн.

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою 5.16:

$$ЗВ = \frac{B_{заг}}{\eta}, \quad (5.16)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,7.

ЗВ =783597,5 / 0,7 = 1119425 грн.

5.3 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

Результати дослідження передбачають вихід продукту на ринок протягом наступних трьох років. При цьому очікуваний економічний ефект буде ґрунтуватися на таких показниках:

ΔN – збільшення кількості споживачів продукту, у періоди часу, що

аналізуються, від покращення його певних характеристик;

1-й рік – 2750 користувачів/рік;

2-й рік – 6500 користувачів/рік;

3-й рік – 12000 користувачів/рік.

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки, приймемо 100000 користувачів;

$Ц_б$ – вартість програмного продукту у році до впровадження результатів розробки, приймемо 2500,00 грн /за рік користування;

$\pm\Delta Ц_о$ – зміна вартості програмного продукту від впровадження результатів науково-технічної розробки, приймемо 500,00 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 3-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою 5.17 [29]:

$$\Delta П_i = (\pm\Delta Ц_о \cdot N + Ц_о \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (5.17)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2025 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0,8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту. Приймемо ρ =0,4;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2025 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1$ = (500,00·100000+2500·2750)·0,83·0,4·(1-0,18/100%) = 18848512 грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2 =$ (500,00·100000+2500·(2750+6500))·0,83·0,4·(1-0,18/100%)= 24233801 грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3 =$ (500,00·100000+2500·(2750+6500+12000))·0,83·0,4·(1-0,18/100%)= 341785873 грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки, розраховується за формулою 5.18:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (5.18)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =0,13;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$ПП =$ 18848512/(1+0,13)^1+24233801/(1+0,13)^2+ 34175873/(1+0,13)^3= 59344314 грн.

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки, розраховується за формулою 5.19:

$$PV = k_{інв} \cdot ЗВ, \qquad (5.19)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=5;

$ЗВ$ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 1119425 грн.

$PV = k_{інв} \cdot ЗВ$ = 5·1119425 = 5597125 грн.

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки розраховується за формулою 5.20:

$$E_{абс} = ПП - PV \qquad (5.20)$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 72620031,15 грн;

PV – теперішня вартість початкових інвестицій, 5597125 грн.

$E_{абс} = ПП - PV$ = 59344314 - 5597125 = 53747189 грн.

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки, розраховується за формулою 5.21:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \qquad (5.21)$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, 53747189 грн;

PV – теперішня вартість початкових інвестицій, 5597125 грн;

$T_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її

розробки до закінчення отримування позитивних результатів від її впровадження, 3 роки.

$$E_в = \sqrt[Тж]{1+\frac{E_{абс}}{PV}} - 1 = (1+53747189/5597125)^{1/3} - 1 = 1{,}2.$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$ розраховується за формулою 5.22:

$$\tau_{мін} = d + f, \qquad (5.22)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2025 році в Україні d =0,1;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,2.

$\tau_{мін}$ = 0,1+0,2 = 0,3 < 1,2 свідчить про те, що внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Розробка методів і програмних засобів вебсистеми організованого доставлення посилок» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки, розраховується за формулою 5.23:

$$T_{ок} = \frac{1}{E_в}, \qquad (5.23)$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$ = 1 / 1,2 = 0,83 року.

$T_{ок}$ < 3-х років, що свідчить про комерційну привабливість науково-

технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

5.4 Висновки

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Розробка методів і програмних засобів вебсистеми організованого доставлення посилок» становить 41 бал, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки високий).

Також термін окупності становить 0,83 р., що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Розробка методів і програмних засобів вебсистеми організованого доставлення посилок».

ВИСНОВКИ

У результаті виконання магістерської кваліфікаційної роботи було розроблено методи і програмні засоби вебсистеми організованого доставлення посилок. Робота оформлена згідно методичних вказівок [31, 32].

Було проведено аналіз предметної області та стану вебсистеми організованого доставлення посилок, порівняльний аналіз аналогів, у результаті якого було доведено актуальність власної розробки. Проведено аналіз методів розв’язання задачі та виконано постановку задача дослідження.

Розроблено вебсистеми організованого доставлення посилок, яка включає функціонал для пошуку координат за текстовою адресою, пошуку замовлень в радіусі встановленого відхилення від маршруту, формування рейтингу користувачів системи та формування статистики профілю користувача.

Розроблено метод пошуку координат за текстовою адресою, який надає користувачеві можливість зручно працювати зі звичними поштовими даними замість географічних координат, автоматично перетворюючи введену адресу на точку на карті, що спрощує побудову маршрутів, пошук і створення оголошень та підвищує загальну зручність взаємодії з системою.

Розроблено метод пошуку замовлень в радіусі встановленого відхилення від маршруту, який дозволяє автоматично виявляти релевантні точки відправлення та доставки вздовж шляху руху виконавця, мінімізувати відхилення від основного маршруту, підвищити ефективність використання поїздок та збалансувати інтереси як відправників, так і перевізників.

Розроблено метод формування рейтингу користувачів системи, який забезпечує об’єктивне оцінювання їхньої надійності та якості виконання зобов’язань на основі історії замовлень, відгуків і показників активності, сприяє підвищенню рівня довіри між учасниками платформи, зменшенню ризиків недобросовісної поведінки та формуванню прозорого конкурентного середовища.

Розроблено метод формування статистики профілю користувача, який

агрегує ключові показники його активності та взаємодії в системі (кількість замовлень, успішних доставок, відгуків, середній рейтинг тощо), подає їх у наочній формі, що дозволяє користувачеві оцінювати власну ефективність, відстежувати динаміку участі в сервісі та підвищує прозорість роботи платформи загалом.

Тестування програми довело повну працездатність цього програмного продукту та відповідність поставленому технічному завданню.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. The Share-a-Ride Problem: People and parcels sharing taxis. URL: https://www.sciencedirect.com/science/article/abs/pii/S0377221714002173 (дата звернення: 21.09.2025).

2. Логістика майбутнього: ефективні рішення для торгівлі [Електронний ресурс] : тези доп. II Міжнар. наук.-практ. інтернет-конф. (Київ, 18 квіт. 2024 р.) / відп. ред. Н. Б. Ільченко. – Київ : Держ. торг.-екон. ун-т, 2024. – 287 с.

3. Holovachov M.O. ANALYSIS OF METHODS AND SOFTWARE FOR ORGANIZED PARCEL DELIVERY. / Holovachov M.O., Voitko V.V., Romaniuk O.V. // Комп'ютерні ігри та мультимедіа як інноваційний підхід до комунікації – 2025 / Матеріали V Всеукраїнської науково-технічної конференції молодих вчених, аспірантів і студентів, Одеса, 25–26 вересня 2025 р. – Одеса, Видавництво ОНТУ, 2025. – 231–233 с.

4. Головачьов М. О. Використання бібліотеки Lombok при створенні Java-застосунків на платформі Spring Framework / Романюк О. В., Бурбело С. М., Головачьов М. О. // Матеріали Міжнародної науково-практичної Інтернет-конференції Електронні інформаційні ресурси: створення, використання, доступ та управління. Збірник матеріалів Міжнародної науково-практичної Інтернет конференції 20-21 листопада 2025р. – Суми/Вінниця: НІКО / КЗВО «Вінницька академія безперервної освіти», 2025. – 292–295 с.

5. What is a Delivery Service & How Does It Work? [Online]. Available: https://freshdrop.com.au/blogs/what-is-delivery-service/ (дата звернення: 24.09.2025).

6. TravelPost. URL: https://en.wikipedia.org/wiki/TravelPost (дата звернення: 29.09.2025).

7. Fleetli. URL: https://fleetli.app/ (дата звернення: 01.10.2025).

8. GoGoBag. URL: https://gogobag.eu/en (дата звернення: 01.10.2025).

9. Nova Post. URL: https://novapost.com/ (дата звернення: 01.10.2025).

10. MVC Design Pattern URL: https://www.geeksforgeeks.org/system-

design/mvc-design-pattern/ (дата звернення: 05.10.2025).

11. HTML. URL: https://developer.mozilla.org/en-US/docs/Web/HTML/Reference/Global_attributes/is (дата звернення: 05.10.2025).

12. Java. URL: https://www.java.com/en/ (дата звернення: 06.10.2025).

13. C Sharp. URL: https://uk.wikipedia.org/wiki/C_Sharp (дата звернення: 06.10.2025).

14. PHP. URL: https://www.php.net/ (дата звернення: 07.10.2025).

15. Go. URL: https://go.dev/ (дата звернення: 09.10.2025).

16. Spring Boot – Spring MVC Framework. URL: https://www.geeksforgeeks.org/springboot/spring-mvc-framework/ (дата звернення: 10.10.2025).

17. Quarkus. URL: https://quarkus.io/ (дата звернення: 10.10.2025).

18. Micronaut. URL: https://micronaut.io/ (дата звернення: 14.10.2025).

19. Thymeleaf. URL: https://www.thymeleaf.org/ (дата звернення: 14.10.2025).

20. PostgreSQL. URL: https://www.postgresql.org/ (дата звернення: 14.10.2025).

21. Annotation-based Container Configuration. URL: https://docs.spring.io/spring-framework/reference/core/beans/annotation-config.html (дата звернення: 20.10.2025).

22. CSS. URL: https://developer.mozilla.org/en-US/docs/Web/CSS (дата звернення: 20.10.2025).

23. JavaScript. URL: https://en.wikipedia.org/wiki/JavaScript (дата звернення: 20.10.2025).

24. Web Based Testing – Software Testing. URL: https://www.geeksforgeeks.org/software-testing/software-testing-web-based-testing/ (дата звернення: 25.10.2025).

25. Unit Testing – Software Testing. URL: https://www.geeksforgeeks.org/software-testing/unit-testing-software-testing/ (дата звернення: 28.10.2025).

26. Integration Testing – Software Testing. URL: https://www.geeksforgeeks.org/software-testing/software-engineering-integration-testing/ (дата звернення: 28.10.2025).

27. Mockito. URL: https://site.mockito.org/ (дата звернення: 04.11.2025).

28. JUnit. URL: https://junit.org/ (дата звернення: 04.11.2025).

29. Кавецький В. В. Економічне обґрунтування інноваційних рішень: практикум / В. В. Кавецький, В. О. Козловський, І. В. Причепа. – Вінниця: ВНТУ, 2016. – 113 с.

30. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця: ВНТУ, 2021. – 42 с.

31. Положення про кваліфікаційну роботу на другому (вищому) рівні вищої освіти. Розробники: А. О. Семенов, Л. П. Громова, Т. В. Макарова, О. В. Сердюк. – Вінниця: ВНТУ, 2021. – 60 с.

32. Методичні вказівки до виконання магістерської кваліфікаційної роботи для студентів спеціальності 121 «Інженерія програмного забезпечення» / уклад. О. Н. Романюк, Г. О. Черноволик. – Вінниця: ВНТУ, 2022. – 50 с.

ДОДАТКИ

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.
«26» вересня 2025 р.

Технічне завдання
на магістерську кваліфікаційну роботу «Розробка методів і програмних засобів вебсистеми організованого доставлення посилок»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:
к.т.н., доц. О.В. Романюк
«26» вересня 2025 року.
Виконав:
студент гр. 1ПІ-24м М.О. Головачьов
«26» вересня 2025 року.

м. Вінниця – 2025 рік

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Розробка методів і програмних засобів вебсистеми організованого доставлення посилок».

Галузь застосування – вебсистеми.

2. Підстава для розробки.

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ №313 від 24 вересня 2025 р. ректора ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки.

Метою магістерської кваліфікаційної роботи є підвищення рівня доставлення посилок шляхом розробки та впровадження вебсистеми організованого доставлення посилок, що дозволить підвищити якість доставлення посилок.

4. Вихідні дані для проведення НДР

1. What is a Delivery Service & How Does It Work? URL: https://freshdrop.com.au/blogs/what-is-delivery-service/

2. Головачьов М. О. Використання бібліотеки Lombok при створенні Java-застосунків на платформі Spring Framework / Романюк О. В., Бурбело С. М., Головачьов М. О. // Матеріали Міжнародної науково-практичної Інтернет-конференції Електронні інформаційні ресурси: створення, використання, доступ та управління. Збірник матеріалів Міжнародної науково-практичної Інтернет конференції 20-21 листопада 2025р. – Суми/Вінниця: НІКО / КЗВО «Вінницька академія безперервної освіти», 2025. – 292–295 с.

3. Holovachov M.O. ANALYSIS OF METHODS AND SOFTWARE FOR ORGANIZED PARCEL DELIVERY. / Holovachov M.O., Voitko V.V., Romaniuk O.V. // Комп'ютерні ігри та мультимедіа як інноваційний підхід до комунікації – 2025 / Матеріали V Всеукраїнської науково-технічної конференції молодих вчених, аспірантів і студентів, Одеса, 25–26 вересня 2025 р. – Одеса,

Видавництво ОНТУ, 2025. – 231–233 с.

4. Методичні вказівки до виконання магістерської кваліфікаційної роботи для студентів спеціальності 121 «Інженерія програмного забезпечення» / уклад. : О. Н. Романюк, Г. О. Черноволик. – Вінниця : ВНТУ, 2022. – 50 с.

5. Технічні вимоги

Вхідні дані – користувач системи, дані про посилки, дані про користувацькі маршрути.

Вихідні дані – графічний інтерфейс користувача з можливістю створення та виконання замовлень з доставлення посилок.

6. Конструктивні вимоги.

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

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

- пояснювальна записка до МКР;
- 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 магістерської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз стану вебсистем організованого доставлення посилок	26.09.2025 30.09.2025
2	Розробка методів, моделі та алгоритмів роботи системи	01.10.2025 17.10.2025
3	Розробка програмних засобів системи	18.10.2025 04.11.2025
4	Тестування системи	05.11.2025 21.11.2025
5	Економічна частина	22.11.2025 01.12.2025
6	Оформлення матеріалів до захисту МКР	01.12.2025 09.12.2025

10. Порядок контролю та прийняття.

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

ДОДАТОК Б

Протокол перевірки на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка методів і програмних засобів вебсистеми організованого доставлення посилок

Тип роботи: магістерська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра програмного забезпечення, ФІТКІ, 1ПІ–24м
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 1,4 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

- ■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту
- ☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.
- ☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(–на)

Керівник _____ Романюк О.В., к.т.н., доц. каф. ПЗ
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Головачьов М.О.
(підпис) (прізвище, ініціали)

ДОДАТОК В

Лістинг коду

AddressDto.class

```
@Data
@NoArgsConstructor
@AllArgsConstructor
@JsonIgnoreProperties(ignoreUnknown = true)
public class AddressDto {
    private double lat;
    private double lon;
}
```

GeocodingService.class

```
@Service
@RequiredArgsConstructor
@Slf4j
public class GeocodingService {

    private final RestTemplate restTemplate = new RestTemplate();
    private final GeometryService geometryService;

    @Value("${geocoding.api-key}")
    private String apiKey;

    @Value("${geocoding.base-url:https://geocode.maps.co/search}")
    private String baseUrl;

    /**
     * Повертає геометричну точку для заданої текстової адреси.
     * Якщо результат не знайдено або сталася помилка — повертає Optional.empty().
     */
    public Optional<Point> getPointByAddress(String address) {
        if (!StringUtils.hasText(address)) {
            log.warn("Empty address was passed to geocoding method");
            return Optional.empty();
        }

        return fetchFirstAddress(address)
                .map(dto -> geometryService.createPoint(dto.getLon(), dto.getLat()));
    }
```

```
    /**
     * Викликає зовнішній геокодер і повертає перший знайдений результат.
     */
    private Optional<AddressDto> fetchFirstAddress(String address) {
        try {
            URI uri = UriComponentsBuilder
                    .fromHttpUrl(baseUrl)
                    .queryParam("api_key", apiKey)
                    .queryParam("q", address)
                    .encode(StandardCharsets.UTF_8)// буде коректно URL-закодовано
                    .build()
                    .toUri();

            ResponseEntity<AddressDto[]> response =
                    restTemplate.getForEntity(uri, AddressDto[].class);

            if (!response.getStatusCode().is2xxSuccessful()) {
                log.warn("Geocoding request failed with status {} for address '{}'",
                        response.getStatusCode(), address);
                return Optional.empty();
            }

            AddressDto[] body = response.getBody();
            if (body == null || body.length == 0) {
                log.info("No geocoding results for address '{}'", address);
                return Optional.empty();
            }

            return Optional.of(body[0]); // беремо найрелевантніший (перший) результат
        } catch (RestClientException ex) {
            log.error("Error while calling geocoding service for address '{}'", address, ex);
            return Optional.empty();
        }
    }
}
```

GeometryService.class

```
@Service
public class GeometryService {

    // SRID 4326 — стандартна WGS84 (широта/довгота)
    private final GeometryFactory geometryFactory =
```

```
        new GeometryFactory(new PrecisionModel(), 4326);

    public Point createPoint(double longitude, double latitude) {
        return geometryFactory.createPoint(new Coordinate(longitude, latitude));
    }
}
```

ProfileController.class

```
@Controller
@RequiredArgsConstructor
public class ProfileController {

    private final AppUserRepository userRepository;
    private final UserRatingService userRatingService;
    private final ParcelRepository parcelRepository;
    private final BookingRepository bookingRepository;

    @GetMapping("/profile")
    public String profile(Authentication authentication, Model model) {
        String email = authentication.getName();
        AppUser user = userRepository.findByEmail(email)
                .orElseThrow(() -> new IllegalStateException("User not found: " + email));

        ProfileUpdateForm form = new ProfileUpdateForm();
        form.setFirstName(user.getFirstName());
        form.setLastName(user.getLastName());
        form.setPhone(user.getPhone());

        model.addAttribute("user", user);
        model.addAttribute("form", form);
        fillStats(model, user);

        return "profile";
    }

    @PostMapping("/profile")
    public String updateProfile(@Valid @ModelAttribute("form") ProfileUpdateForm form,
            BindingResult bindingResult,
            Authentication authentication,
            Model model) {
        String email = authentication.getName();
        AppUser user = userRepository.findByEmail(email)
                .orElseThrow(() -> new IllegalStateException("User not found: " + email));
```

```
        if (bindingResult.hasErrors()) {
            model.addAttribute("user", user);
            fillStats(model, user);
            return "profile";
        }

        user.setFirstName(form.getFirstName());
        user.setLastName(form.getLastName());
        user.setPhone(form.getPhone());
        userRepository.save(user);

        return "redirect:/profile";
    }

    private void fillStats(Model model, AppUser user) {
        Long userId = user.getId();

        double rating = userRatingService.getAverageRatingForUser(userId);
        long ratingCount = userRatingService.getReviewCountForUser(userId);

        long rating1 = userRatingService.getRatingCountForUserAndValue(userId, 1);
        long rating2 = userRatingService.getRatingCountForUserAndValue(userId, 2);
        long rating3 = userRatingService.getRatingCountForUserAndValue(userId, 3);
        long rating4 = userRatingService.getRatingCountForUserAndValue(userId, 4);
        long rating5 = userRatingService.getRatingCountForUserAndValue(userId, 5);

        model.addAttribute("rating", rating);
        model.addAttribute("ratingCount", ratingCount);
        model.addAttribute("rating1Count", rating1);
        model.addAttribute("rating2Count", rating2);
        model.addAttribute("rating3Count", rating3);
        model.addAttribute("rating4Count", rating4);
        model.addAttribute("rating5Count", rating5);

        long parcelsTotal = parcelRepository.countBySenderId(userId);
        long parcelsDelivered = parcelRepository.countBySenderIdAndStatus(userId, Parcel.STATUS_DELIVERED);
        long parcelsCanceled = parcelRepository.countBySenderIdAndStatus(userId, Parcel.STATUS_CANCELED);

        long deliveriesTotal = bookingRepository.countByCarrierId(userId);
        long deliveriesCompleted = bookingRepository.countByCarrierIdAndStatus(userId,
BookingStatus.COMPLETED);
```

```
        long deliveriesCanceled = bookingRepository.countByCarrierIdAndStatus(userId, BookingStatus.CANCELED);

        model.addAttribute("parcelsTotal", parcelsTotal);
        model.addAttribute("parcelsDelivered", parcelsDelivered);
        model.addAttribute("parcelsCanceled", parcelsCanceled);

        model.addAttribute("deliveriesTotal", deliveriesTotal);
        model.addAttribute("deliveriesCompleted", deliveriesCompleted);
        model.addAttribute("deliveriesCanceled", deliveriesCanceled);
    }

}
```

ProfileUpdateForm.class

```
@Data
public class ProfileUpdateForm {

    @NotBlank(message = "Ім'я не може бути порожнім")
    private String firstName;

    @NotBlank(message = "Прізвище не може бути порожнім")
    private String lastName;

    private String phone;
}
```

AppUser.class

```
@Getter
@Setter
@Entity
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Table(name = "app_user")
public class AppUser {

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

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

```
    @Column(name = "phone")
    private String phone;

    @Column(name = "first_name")
    private String firstName;

    @Column(name = "last_name")
    private String lastName;

    @Enumerated(EnumType.STRING)
    @Column(name = "role")
    private Role role;

    @Column(nullable = false)
    private String password;

    @Column(name = "enabled", nullable = false)
    private boolean enabled = true;



    @Column(name = "created_at", nullable = false, updatable = false)
    @Builder.Default
    private OffsetDateTime createdAt = OffsetDateTime.now(ZoneOffset.UTC);

    @OneToMany(mappedBy = "carrier", fetch = FetchType.LAZY)
    private List<Offer> offers;
}
```

AppUserRepository.class

```
public interface AppUserRepository extends JpaRepository<AppUser, Long> {
    Optional<AppUser> findByEmail(String email);
    boolean existsByEmail(String email);
}
```

AuthController.class

```
@Controller
@RequestMapping("/auth")
@RequiredArgsConstructor
public class AuthController {

    private final AppUserRepository userRepository;
    private final AuthService authService;
```

```
    @GetMapping("/login")
    public String login() {
        return "auth/login";
    }

    @GetMapping("/register")
    public String registerForm(Model model) {
        model.addAttribute("registrationRequest", new RegistrationRequest());
        return "auth/register";
    }

    @PostMapping("/register")
    public String register(
            @Valid @ModelAttribute("registrationRequest") RegistrationRequest request,
            BindingResult bindingResult,
            Model model
    ) {
        if (userRepository.existsByEmail(request.getEmail())) {
            bindingResult.rejectValue("email", "email.exists",
                    "Користувач з такою поштою вже існує");
        }

        if (!Objects.equals(request.getPassword(), request.getConfirmPassword())) {
            bindingResult.rejectValue("confirmPassword", "password.mismatch",
                    "Паролі не співпадають");
        }

        if (bindingResult.hasErrors()) {
            return "auth/register";
        }

        authService.register(request);
        return "redirect:/auth/login?registered";
    }
}
```

RegistrationRequest.class

```
@Data
public class RegistrationRequest {

    @NotBlank
    @Email
    private String email;
```

```
    @NotBlank
    @Size(min = 8, max = 100)
    private String password;

    @NotBlank
    @Size(min = 8, max = 100)
    private String confirmPassword;

    @NotBlank
    @Size(min = 2, max = 50)
    private String firstName;

    @NotBlank
    @Size(min = 2, max = 50)
    private String lastName;

    private String phone;
}
```

AuthService.class

```
@Service
@RequiredArgsConstructor
public class AuthService {

    private final AppUserRepository userRepository;
    private final PasswordEncoder passwordEncoder;

    public AppUser register(RegistrationRequest request) {
        var user = AppUser.builder()
                .email(request.getEmail())
                .password(passwordEncoder.encode(request.getPassword()))
                .firstName(request.getFirstName())
                .lastName(request.getLastName())
                .phone(request.getPhone())
                .role(Role.USER)
                .enabled(true)
                .build();

        return userRepository.save(user);
    }
}
```

BookingController.class

```
@Controller
@RequiredArgsConstructor
@RequestMapping("/bookings")
public class BookingController {

    private final BookingService bookingService;

    // сторінка з двома таблицями
    @GetMapping
    public String bookingsPage(Authentication authentication, Model model) {
        String email = authentication.getName();

        List<BookingSummaryDto> asSender = bookingService.getBookingsAsSender(email);
        List<BookingSummaryDto> asCarrier = bookingService.getBookingsAsCarrier(email);

        model.addAttribute("bookingsAsSender", asSender);
        model.addAttribute("bookingsAsCarrier", asCarrier);

        return "bookings";
    }

    // створення бронювання з мапи (AJAX)
    @PostMapping
    @ResponseBody
    public void createBooking(@RequestBody CreateBookingRequest request,
            Authentication authentication) {
        String email = authentication.getName();
        bookingService.createBooking(
                request.getParcelId(),
                email,
                request.getOfferedPrice(),
                request.getCarrierComment()
        );
    }

    // відправник приймає
    @PostMapping("/{id}/accept")
    public String accept(@PathVariable Long id, Authentication auth) {
        bookingService.acceptBooking(id, auth.getName());
        return "redirect:/bookings";
    }
```

```
    // відправник відхиляє
    @PostMapping("/{id}/reject")
    public String reject(@PathVariable Long id, Authentication auth) {
        bookingService.rejectBooking(id, auth.getName());
        return "redirect:/bookings";
    }

    // перевізник скасовує
    @PostMapping("/{id}/cancel")
    public String cancel(@PathVariable Long id, Authentication auth) {
        bookingService.cancelBooking(id, auth.getName());
        return "redirect:/bookings";
    }
}
```

BookingSummaryDto.class

```
@Data
@AllArgsConstructor
public class BookingSummaryDto {

    private Long id;
    private Long parcelId;
    private String pickupAddress;
    private String deliveryAddress;
    private String counterpartyEmail; // для відправника – перевізник, для перевізника – відправник
    private String status;
    private OffsetDateTime createdAt;
    private BigDecimal offeredPrice;
    private String carrierComment;
    private double counterpartyRating;
}
```

CreateBookingRequest.class

```
@Data
public class CreateBookingRequest {
    private Long parcelId;
    private BigDecimal offeredPrice;
    private String carrierComment;
}
```

Booking.class

```
@Getter
@Setter
```

```
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Entity
@Table(name = "booking")
public class Booking {

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

    // до якої посилки відноситься запит
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "parcel_id", nullable = false)
    private Parcel parcel;

    // хто хоче її везти
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "carrier_id", nullable = false)
    private AppUser carrier;

    @Enumerated(EnumType.STRING)
    @Column(name = "status", nullable = false)
    private BookingStatus status;

    @Column(name = "offered_price", precision = 10, scale = 2)
    private BigDecimal offeredPrice;

    @Column(name = "carrier_comment")
    private String carrierComment;

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

    @Column(name = "updated_at", nullable = false)
    private OffsetDateTime updatedAt;

    @PrePersist
    void prePersist() {
        OffsetDateTime now = OffsetDateTime.now(ZoneOffset.UTC);
        if (createdAt == null) {
            createdAt = now;
```

```
        }
        updatedAt = now;
        if (status == null) {
            status = BookingStatus.REQUESTED;
        }
    }

    @PreUpdate
    void preUpdate() {
        updatedAt = OffsetDateTime.now(ZoneOffset.UTC);
    }
}
```

BookingRepository.class

```
public interface BookingRepository extends JpaRepository<Booking, Long> {

    List<Booking> findByParcelSenderId(Long senderId);

    List<Booking> findByCarrierId(Long carrierId);

    List<Booking> findByParcelId(Long parcelId);

    boolean existsByParcelIdAndStatus(Long parcelId, BookingStatus status);

    List<Booking> findByCarrierIdAndStatusIn(Long carrierId, Collection<BookingStatus> statuses);
    List<Booking> findByParcelSenderIdAndStatusIn(Long senderId, Collection<BookingStatus> statuses);
    long countByCarrierId(Long carrierId);

    long countByCarrierIdAndStatus(Long carrierId, BookingStatus status);
}
```

BookingService.class

```
@Service
@RequiredArgsConstructor
public class BookingService {

    private final BookingRepository bookingRepository;
    private final ParcelRepository parcelRepository;
    private final AppUserRepository userRepository;
    private final UserRatingService userRatingService;

    @Transactional
    public Booking createBooking(Long parcelId,
```

```
        String carrierEmail,
        BigDecimal offeredPrice,
        String carrierComment) {

    Parcel parcel = parcelRepository.findById(parcelId)
            .orElseThrow(() -> new IllegalArgumentException("Посилку не знайдено: id=" + parcelId));

    if (STATUS_DELIVERED.equals(parcel.getStatus()) || STATUS_CANCELED.equals(parcel.getStatus())) {
        throw new IllegalStateException("Неможливо забронювати посилку зі статусом " + parcel.getStatus());
    }

    if (bookingRepository.existsByParcelIdAndStatus(parcelId, BookingStatus.ACCEPTED)) {
        throw new IllegalStateException("Для цієї посилки вже є прийняте бронювання.");
    }

    AppUser carrier = userRepository.findByEmail(carrierEmail)
            .orElseThrow(() -> new IllegalStateException("Користувача не знайдено: " + carrierEmail));

    if (parcel.getSender().getId().equals(carrier.getId())) {
        throw new IllegalStateException("Відправник не може забронювати власну посилку як перевізник.");
    }

    if (offeredPrice != null && offeredPrice.signum() <= 0) {
        throw new IllegalArgumentException("offeredPrice має бути > 0, якщо заданий.");
    }

    Booking booking = Booking.builder()
            .parcel(parcel)
            .carrier(carrier)
            .status(BookingStatus.REQUESTED)
            .offeredPrice(offeredPrice)
            .carrierComment(carrierComment)
            .build();

    return bookingRepository.save(booking);
}

@Transactional(readOnly = true)
public List<BookingSummaryDto> getBookingsAsSender(String senderEmail) {
    AppUser sender = userRepository.findByEmail(senderEmail)
            .orElseThrow(() -> new IllegalStateException("User not found: " + senderEmail));
```

```
    var activeStatuses = List.of(BookingStatus.REQUESTED, BookingStatus.ACCEPTED);

    return bookingRepository.findByParcelSenderIdAndStatusIn(sender.getId(), activeStatuses)
            .stream()
            .map(b -> {
                double carrierRating = userRatingService.getAverageRatingForUser(b.getCarrier().getId());
                return new BookingSummaryDto(
                        b.getId(),
                        b.getParcel().getId(),
                        b.getParcel().getPickupAddressText(),
                        b.getParcel().getDeliveryAddressText(),
                        b.getCarrier().getEmail(),
                        b.getStatus().name(),
                        b.getCreatedAt(),
                        b.getOfferedPrice(),
                        b.getCarrierComment(),
                        carrierRating
                );
            })
            .toList();
}

@Transactional(readOnly = true)
public List<BookingSummaryDto> getBookingsAsCarrier(String carrierEmail) {
    AppUser carrier = userRepository.findByEmail(carrierEmail)
            .orElseThrow(() -> new IllegalStateException("User not found: " + carrierEmail));

    var activeStatuses = List.of(BookingStatus.REQUESTED, BookingStatus.ACCEPTED);

    return bookingRepository.findByCarrierIdAndStatusIn(carrier.getId(), activeStatuses)
            .stream()
            .map(b -> {
                double senderRating = userRatingService.getAverageRatingForUser(
                        b.getParcel().getSender().getId()
                );
                return new BookingSummaryDto(
                        b.getId(),
                        b.getParcel().getId(),
                        b.getParcel().getPickupAddressText(),
                        b.getParcel().getDeliveryAddressText(),
                        b.getParcel().getSender().getEmail(),
                        b.getStatus().name(),
```

```
                b.getCreatedAt(),
                b.getOfferedPrice(),
                b.getCarrierComment(),
                senderRating
            );
        })
        .toList();
}


@Transactional(readOnly = true)
public List<CarrierHistoryDto> getCarrierHistory(String carrierEmail,
        List<BookingStatus> statuses) {
    AppUser carrier = userRepository.findByEmail(carrierEmail)
            .orElseThrow(() -> new IllegalStateException("User not found: " + carrierEmail));

    List<Booking> bookings;
    if (statuses == null || statuses.isEmpty()) {
        bookings = bookingRepository.findByCarrierId(carrier.getId());
    } else {
        bookings = bookingRepository.findByCarrierIdAndStatusIn(carrier.getId(), statuses);
    }

    return bookings.stream()
            .map(b -> {
                String uiStatus;
                if (b.getStatus() == BookingStatus.COMPLETED) {
                    uiStatus = "DELIVERED";
                } else if (b.getStatus() == BookingStatus.CANCELED) {
                    uiStatus = "CANCELED";
                } else {
                    uiStatus = b.getStatus().name();
                }

                boolean canRate = !userRatingService.hasReviewForBooking(carrier.getId(), b.getId());

                return new CarrierHistoryDto(
                        b.getId(),
                        b.getParcel().getId(),
                        b.getParcel().getPickupAddressText(),
                        b.getParcel().getDeliveryAddressText(),
                        b.getParcel().getWeightKg(),
```

```
                uiStatus,
                b.getCreatedAt(),
                b.getOfferedPrice(),
                b.getCarrierComment(),
                b.getParcel().getSender().getEmail(),
                b.getParcel().getSender().getId(),
                canRate
            );
        })
        .toList();
}

// Відправник приймає бронювання
@Transactional
public void acceptBooking(Long bookingId, String senderEmail) {
    Booking booking = bookingRepository.findById(bookingId)
            .orElseThrow(() -> new IllegalArgumentException("Booking not found: " + bookingId));

    Parcel parcel = booking.getParcel();

    if (!parcel.getSender().getEmail().equals(senderEmail)) {
        throw new IllegalStateException("Немає прав приймати це бронювання.");
    }

    if (booking.getStatus() != BookingStatus.REQUESTED) {
        throw new IllegalStateException("Можна приймати тільки бронювання в статусі REQUESTED.");
    }

    if (bookingRepository.existsByParcelIdAndStatus(parcel.getId(), BookingStatus.ACCEPTED)) {
        throw new IllegalStateException("Для цієї посилки вже є прийняте бронювання.");
    }

    booking.setStatus(BookingStatus.ACCEPTED);
    parcel.setStatus(STATUS_IN_DELIVERY);

    // інші REQUESTED для цієї посилки відхиляємо
    bookingRepository.findByParcelId(parcel.getId())
            .stream()
            .filter(b -> !b.getId().equals(bookingId) && b.getStatus() == BookingStatus.REQUESTED)
            .forEach(b -> b.setStatus(BookingStatus.REJECTED));
}
```

```
    // Відправник відхиляє бронювання
    @Transactional
    public void rejectBooking(Long bookingId, String senderEmail) {
        Booking booking = bookingRepository.findById(bookingId)
                .orElseThrow(() -> new IllegalArgumentException("Booking not found: " + bookingId));

        Parcel parcel = booking.getParcel();

        if (!parcel.getSender().getEmail().equals(senderEmail)) {
            throw new IllegalStateException("Немає прав відхиляти це бронювання.");
        }

        if (booking.getStatus() != BookingStatus.REQUESTED) {
            throw new IllegalStateException("Можна відхиляти тільки бронювання в статусі REQUESTED.");
        }

        booking.setStatus(BookingStatus.REJECTED);
        // статус посилки лишається NEW (якщо немає інших ACCEPTED)
    }

    // Перевізник скасовує свій запит
    @Transactional
    public void cancelBooking(Long bookingId, String carrierEmail) {
        Booking booking = bookingRepository.findById(bookingId)
                .orElseThrow(() -> new IllegalArgumentException("Booking not found: " + bookingId));

        if (!booking.getCarrier().getEmail().equals(carrierEmail)) {
            throw new IllegalStateException("Немає прав скасовувати це бронювання.");
        }

        if (booking.getStatus() != BookingStatus.REQUESTED) {
            throw new IllegalStateException("Можна скасовувати тільки бронювання в статусі REQUESTED.");
        }

        booking.setStatus(BookingStatus.CANCELED);
    }
}
```

SecurityConfig.class

```
@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
public class SecurityConfig {
```

```
private final AppUserRepository userRepository;

@Bean
public PasswordEncoder passwordEncoder() {
   return new BCryptPasswordEncoder();
}

@Bean
public UserDetailsService userDetailsService() {
   return username -> userRepository.findByEmail(username)
         .filter(AppUser::isEnabled)
         .map(user -> User.withUsername(user.getEmail())
               .password(user.getPassword())
               .roles(user.getRole().name()) // USER / ADMIN
               .build()
         )
         .orElseThrow(() -> new UsernameNotFoundException("User not found"));
}

@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
   http
         .csrf(AbstractHttpConfigurer::disable)
         .authorizeHttpRequests(auth -> auth
               .requestMatchers(
                     "/auth/login",
                     "/auth/register",
                     "/css/**",
                     "/js/**",
                     "/images/**"
               ).permitAll()
               .anyRequest().authenticated()
         )
         .formLogin(form -> form
               .loginPage("/auth/login")
               .loginProcessingUrl("/auth/login")
               .usernameParameter("email")
               .passwordParameter("password")
               .defaultSuccessUrl("/routes", true)
               .permitAll()
         )
```

```
            .logout(logout -> logout
                .logoutUrl("/auth/logout")
                .logoutSuccessUrl("/auth/login?logout")
                .permitAll()
            )
            .userDetailsService(userDetailsService());

        return http.build();
    }
}
```

HistoryController.class

```
@Controller
@RequiredArgsConstructor
public class HistoryController {

    private final ParcelService parcelService;
    private final BookingService bookingService;

    @GetMapping("/history")
    public String history(@RequestParam(name = "status", required = false, defaultValue = "ALL") String status,
                          Authentication authentication,
                          Model model) {

        String email = authentication.getName();

        List<String> parcelStatuses;
        List<BookingStatus> bookingStatuses;

        switch (status) {
            case "DELIVERED" -> {
                parcelStatuses = List.of(Parcel.STATUS_DELIVERED);
                bookingStatuses = List.of(BookingStatus.COMPLETED);
            }
            case "CANCELED" -> {
                parcelStatuses = List.of(Parcel.STATUS_CANCELED);
                bookingStatuses = List.of(BookingStatus.CANCELED);
            }
            default -> {
                parcelStatuses = List.of(Parcel.STATUS_DELIVERED, Parcel.STATUS_CANCELED);
                bookingStatuses = List.of(BookingStatus.COMPLETED, BookingStatus.CANCELED);
                status = "ALL";
            }
```

```
        }

        List<ParcelHistoryDto> senderHistory =
                parcelService.getParcelHistory(email, parcelStatuses);

        List<CarrierHistoryDto> carrierHistory =
                bookingService.getCarrierHistory(email, bookingStatuses);

        model.addAttribute("senderHistory", senderHistory);
        model.addAttribute("carrierHistory", carrierHistory);
        model.addAttribute("statusFilter", status);

        return "history";
    }
}
```

CarrierHistoryDto.class

```
@Data
@AllArgsConstructor
public class CarrierHistoryDto {
    private Long bookingId;
    private Long parcelId;
    private String pickupAddress;
    private String deliveryAddress;
    private BigDecimal weightKg;
    private String status;          // DELIVERED / CANCELED (UI-статус)
    private OffsetDateTime createdAt;
    private BigDecimal offeredPrice;
    private String carrierComment;
    private String senderEmail;
    private Long senderId;
    private boolean canRate;
}
```

ParcelHistoryDto.class

```
@Data
@AllArgsConstructor
public class ParcelHistoryDto {
    private Long id;
    private String pickupAddress;
    private String deliveryAddress;
    private BigDecimal weightKg;
    private String status;          // DELIVERED / CANCELED
```

```
    private OffsetDateTime createdAt;
    private Long carrierId;
    private String carrierEmail;
    private Long bookingId;
    private boolean canRate;
}
```

OfferController.class

```
@RestController
@RequiredArgsConstructor
@RequestMapping("/offers")
public class OfferController {

    private final OfferService offerService;


    @GetMapping
    public List<OfferMapDto> getAllOffers() {
        return offerService.getAllOffersForMap();
    }

    /**
     * Оферu біля маршруту в радіусі radiusKm.
     */
    @GetMapping("/near-route")
    public List<OfferMapDto> getOffersNearRoute(@RequestParam String startAddress,
            @RequestParam String endAddress,
            @RequestParam double radiusKm) {
        return offerService.getOffersNearRoute(startAddress, endAddress, radiusKm);
    }
}
```

OfferMapDto.class

```
@Data
@AllArgsConstructor
public class OfferMapDto {
    private Long id;

    private double startLat;
    private double startLon;
    private double endLat;
    private double endLon;

    private BigDecimal capacityKg;
```

```
    private BigDecimal pricePerKm;
    private String status;

    private String name;
    private String description;
    private String startAddressText;
    private String endAddressText;
}
```

ДОДАТОК Г

Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА МЕТОДІВ І ПРОГРАМНИХ ЗАСОБІВ ВЕБСИСТЕМИ ОРГАНІЗОВАНОГО ДОСТАВЛЕННЯ ПОСИЛОК

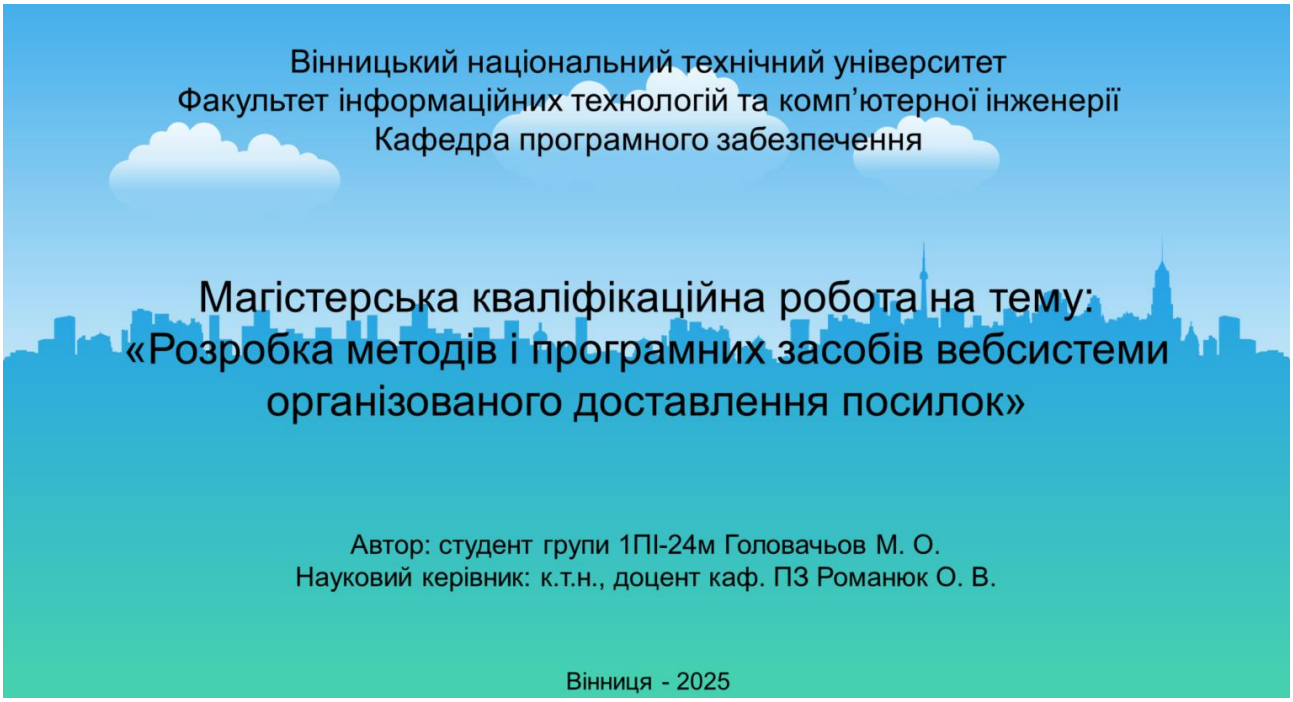

Рисунок Г.1 – Титульний слайд

Рисунок Г.2 – Актуальність теми

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

Рисунок Г.4 – Завдання дослідження

Рисунок Г.5 – Наукова новизна

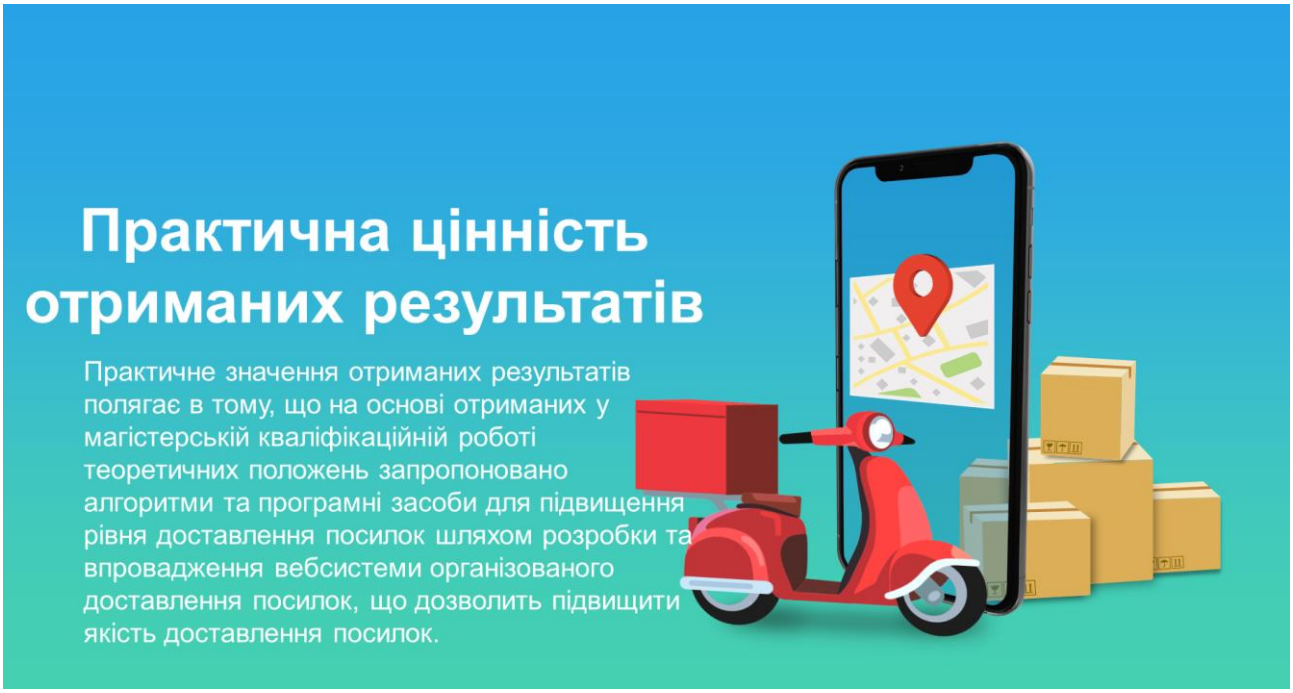

Рисунок Г.6 – Практична цінність отриманих результатів

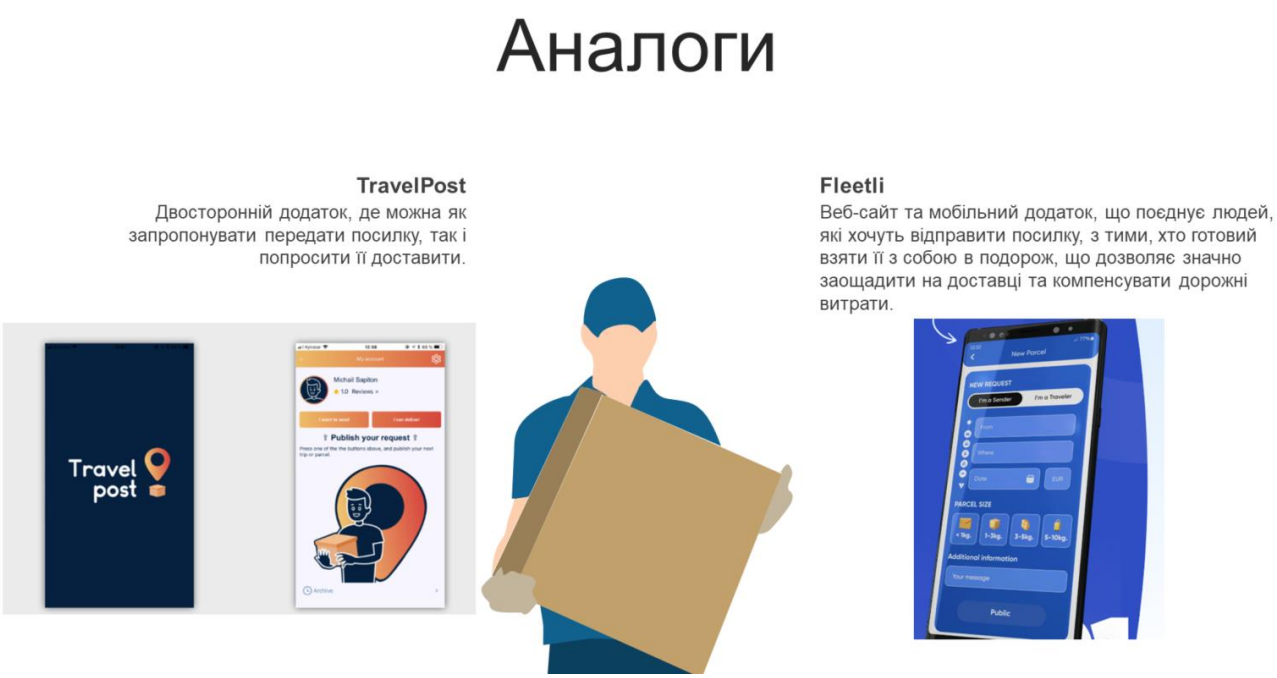

Рисунок Г.7 – Аналоги (частина 1)

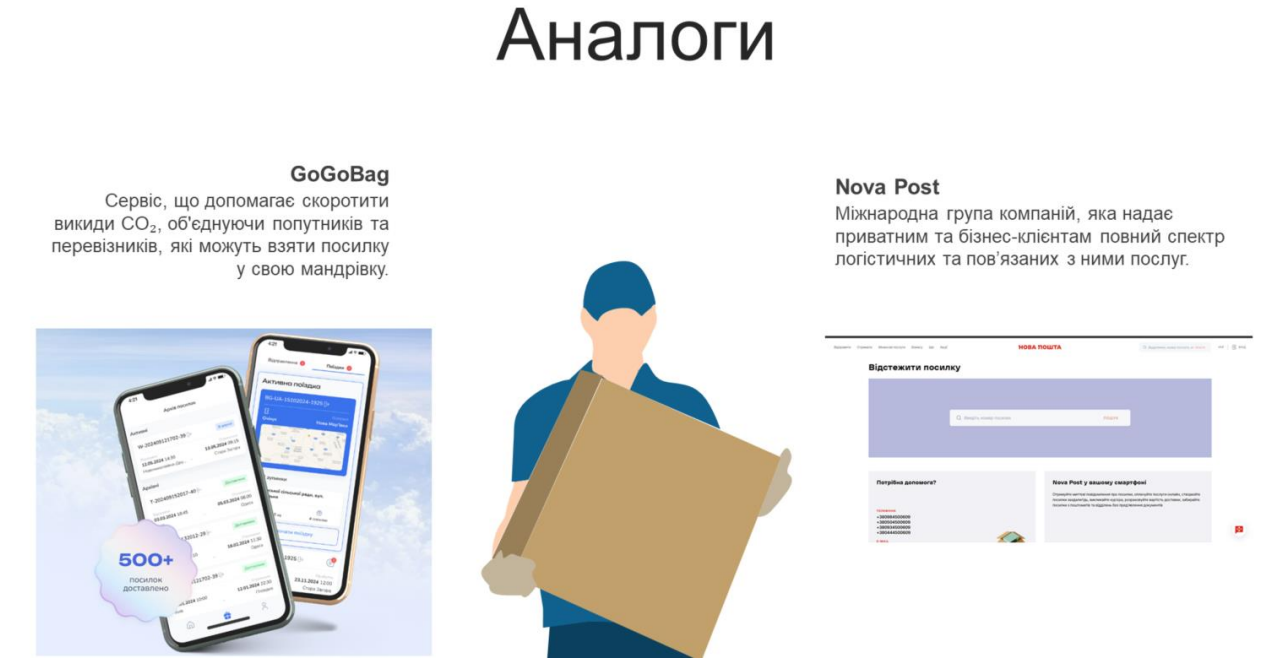

Рисунок Г.8 – Аналоги (частина 2)

Порівняльний аналіз аналогів

Критерій	TravelPost	Fleetli	GoGoBag	Nova Post	Власна розробка
Можливість використання системи з браузера	0	0	0	1	1
Можливість відображення доставок на інтерактивній мапі	0	0	1	0	1
Можливість пошуку замовлення поблизу маршруту користувача	0	1	0	0	1
Наявність статистики про виконані доставки	0	0	0	0	1
Наявність рейтингової системи користувачів	1	1	0	0	1
Загальна оцінка	**1**	**2**	**1**	**1**	5

Рисунок Г.9 – Порівняльний аналіз аналогів

Використані технології

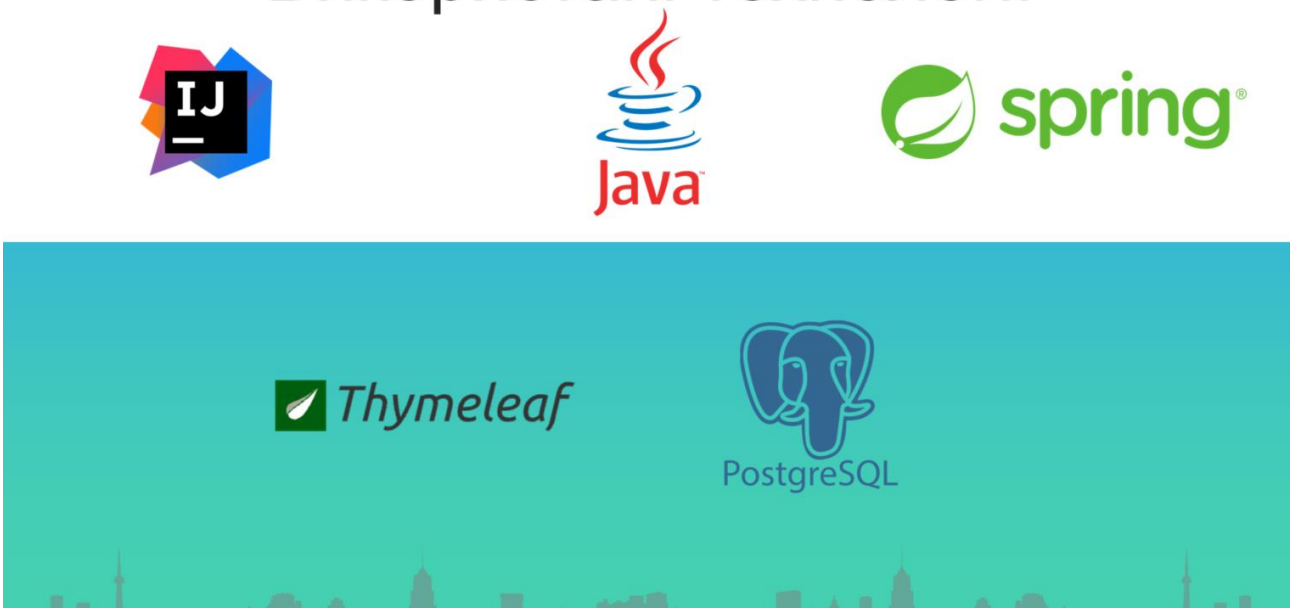

Рисунок Г.10 – Використані технології

Рисунок Г.11– Розробка методу пошуку координат за текстовою адресою

Рисунок Г.12 – Блок-схема методу пошуку координат за текстовою адресою

Рисунок Г.13 – Діаграма послідовностей методу пошуку координат за текстовою адресою

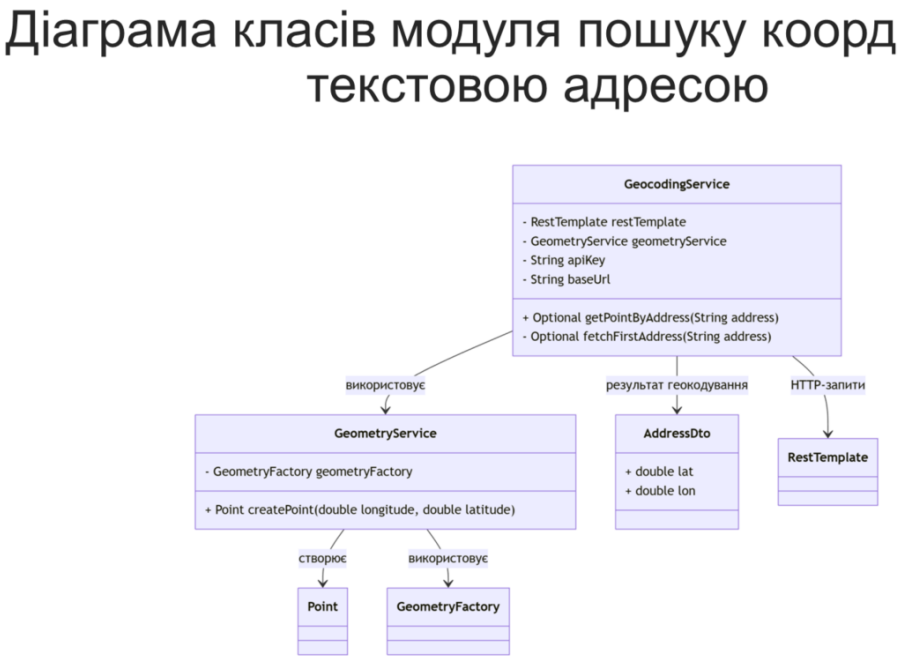

Рисунок Г.14 – Діаграма класів методу пошуку координат за текстовою адресою

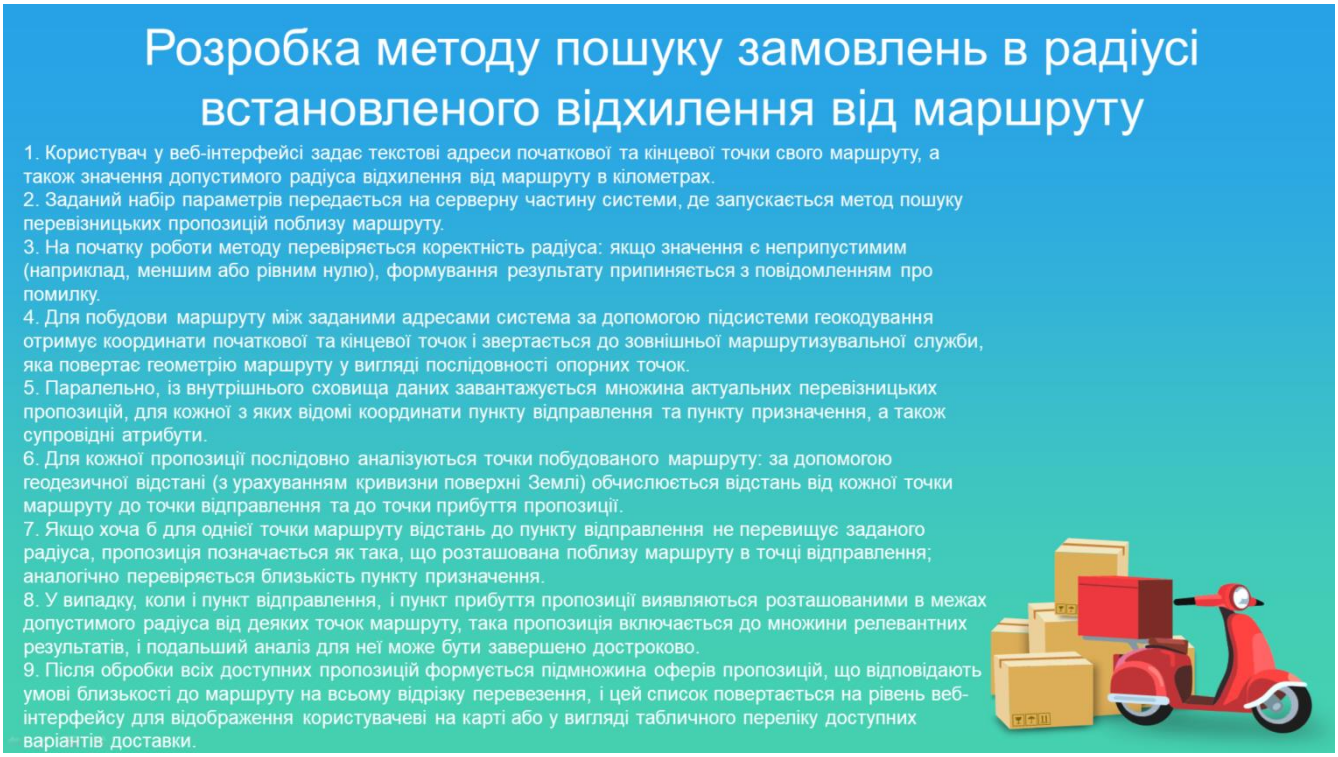

Рисунок Г.15 – Розробка методу пошуку замовлень в радіусі встановленого відхилення від маршруту

Рисунок Г.16 – Блок-схема методу пошуку замовлень в радіусі встановленого відхилення від маршруту

Діаграма послідовностей методу пошуку замовлень в радіусі встановленого відхилення від маршруту

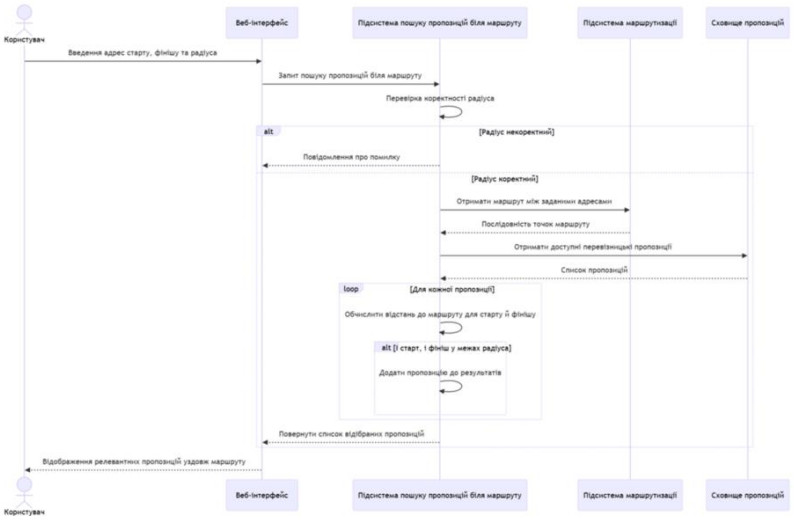

Рисунок Г.17 – Діаграма послідовностей методу пошуку замовлень в радіусі встановленого відхилення від маршруту

Діаграма класів модуля пошуку замовлень в радіусі встановленого відхилення від маршруту

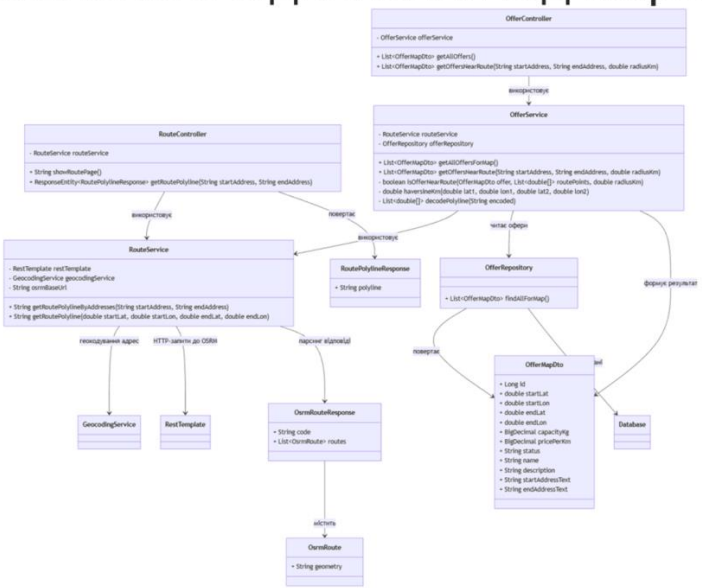

Рисунок Г.18 – Діаграма класів методу пошуку замовлень в радіусі встановленого відхилення від маршруту

Рисунок Г.19 – Розробка методу формування рейтингу користувачів системи

Рисунок Г.20 – Блок-схема методу формування рейтингу користувачів системи

Рисунок Г.21 – Діаграма послідовностей методу формування рейтингу користувачів системи

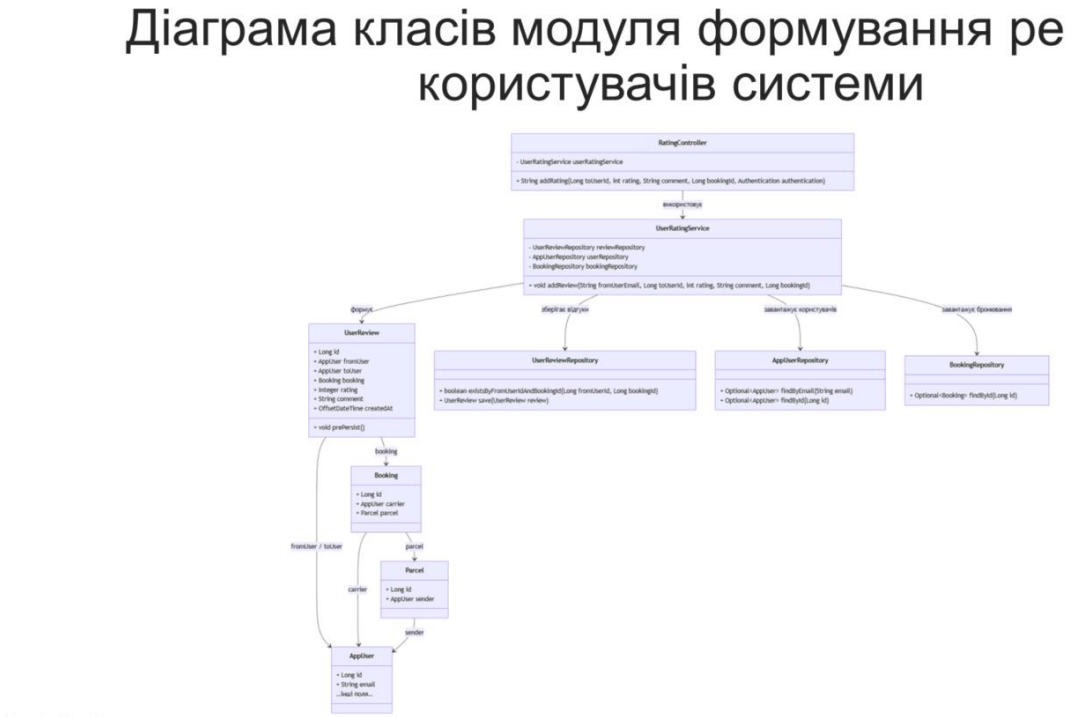

Рисунок Г.22 – Діаграма класів методу формування рейтингу користувачів системи

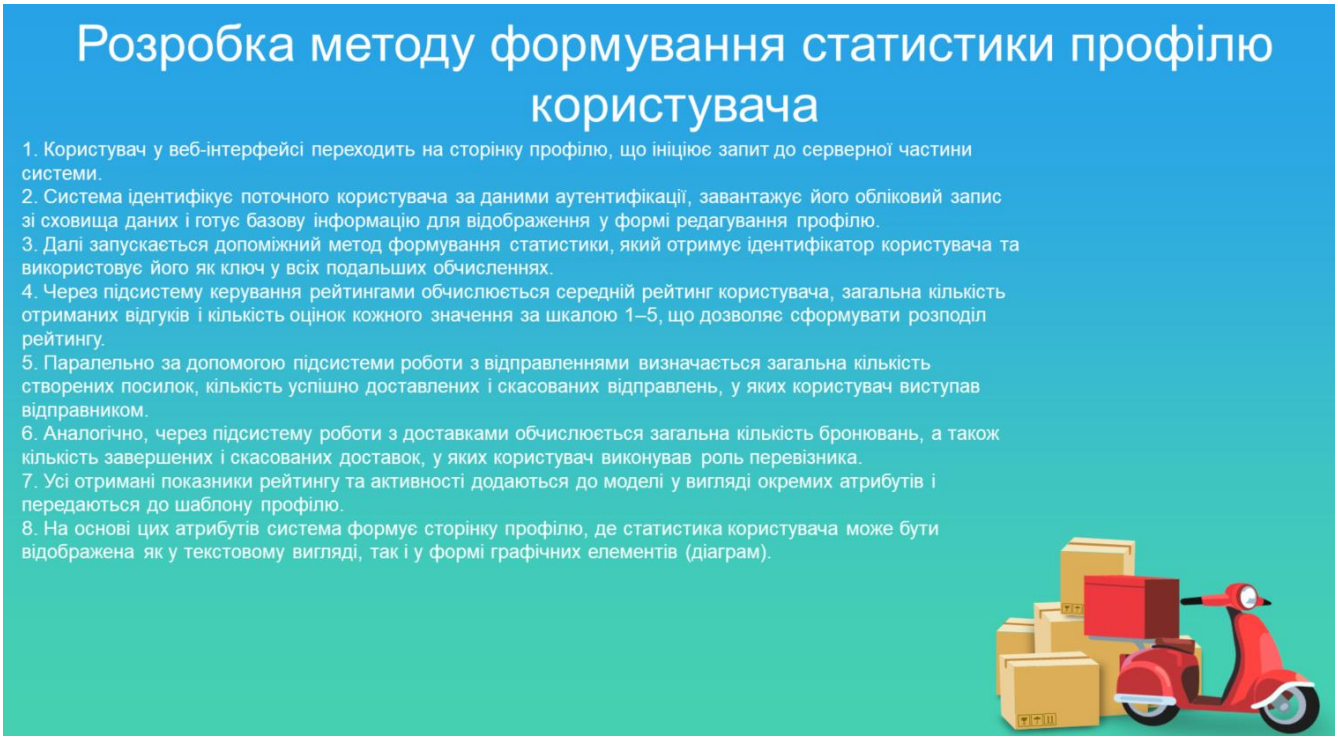

Рисунок Г.23 – Розробка методу формування статистики профілю користувача

Рисунок Г.24 – Блок-схема методу формування рейтингу користувачів системи

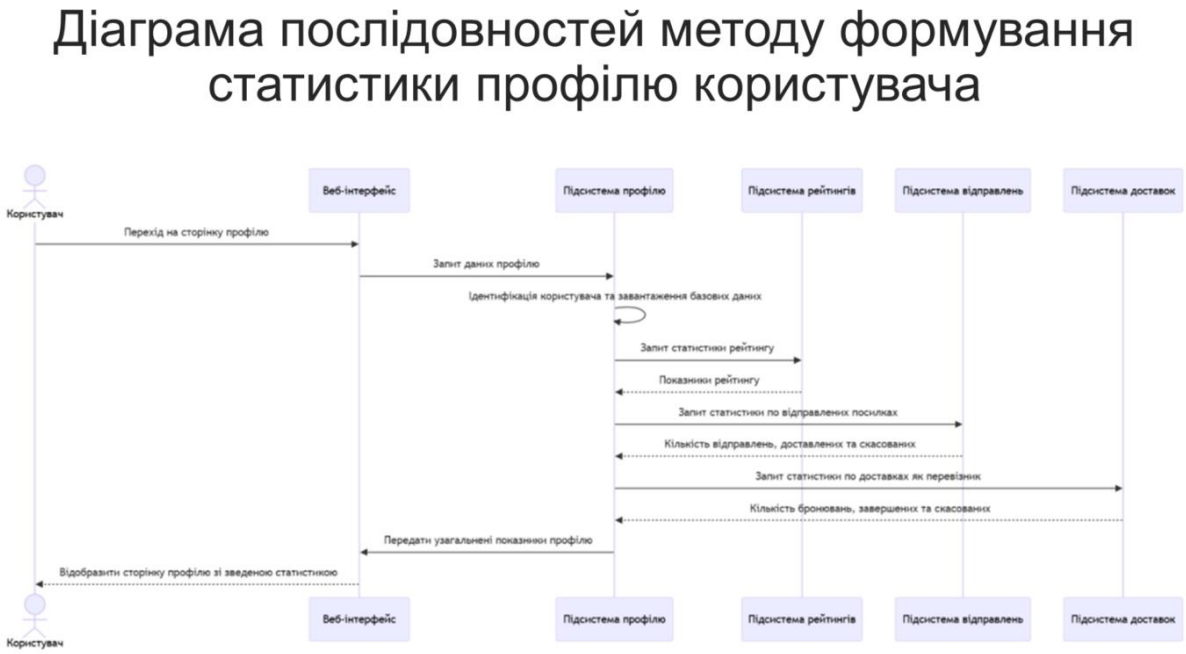

Рисунок Г.25 – Діаграма послідовностей методу формування рейтингу користувачів системи

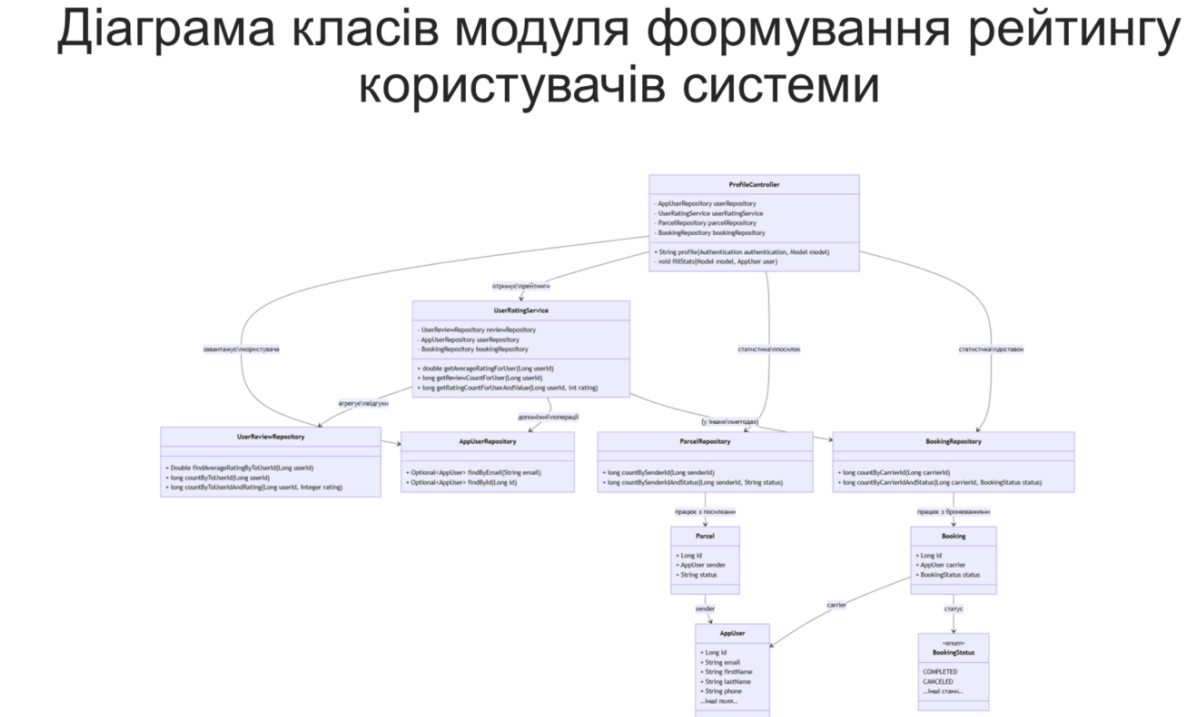

Рисунок Г.26 – Діаграма класів методу формування рейтингу користувачів системи

Модель системи

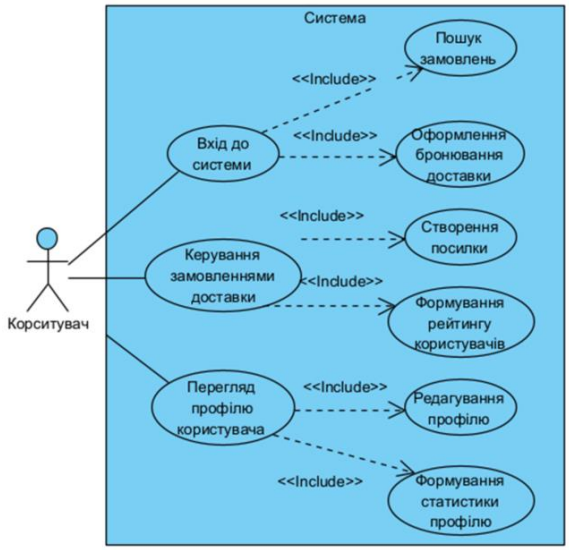

Рисунок Г.27 – Модель системи

Діаграма станів загального алгоритму роботи системи

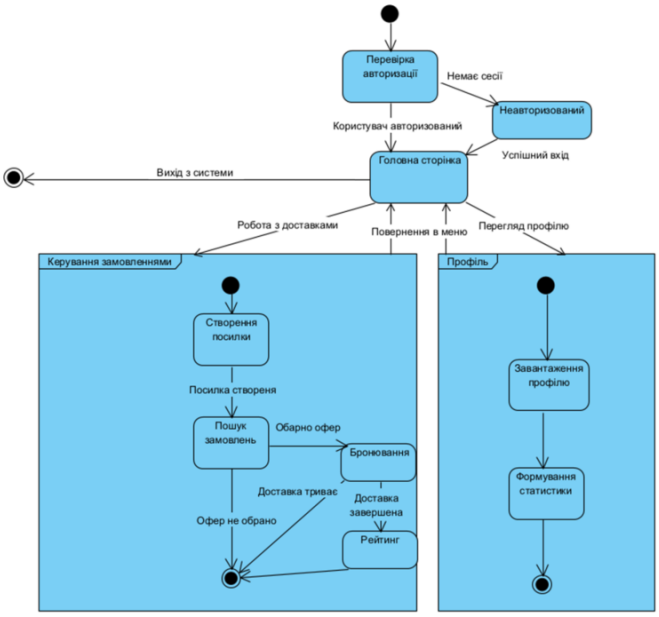

Рисунок Г.28 – Загальний алгоритм роботи системи

Загальна схема таблиць бази даних та їх зв’язків

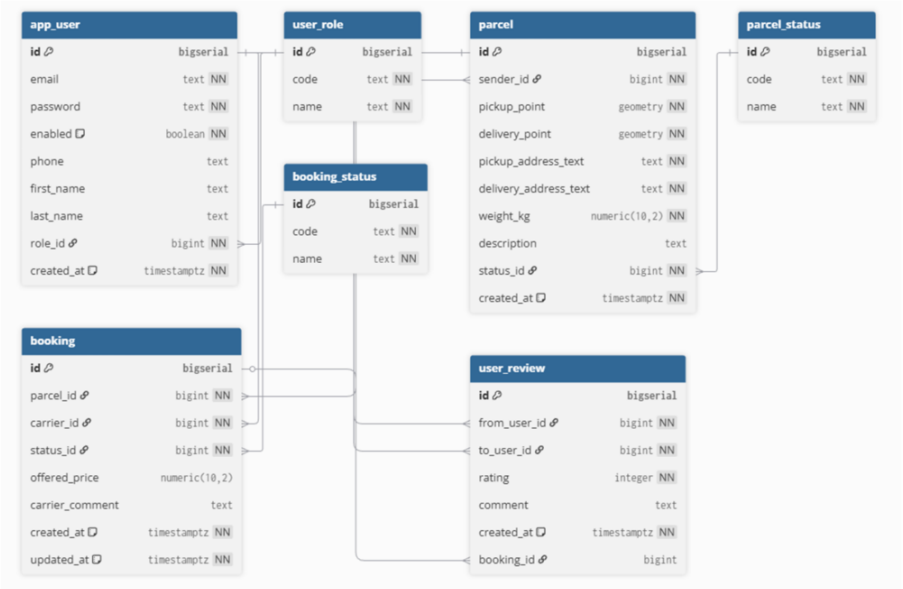

Рисунок Г.29 – Загальна схема таблиць бази даних та їх зв’язків

Сервісна топологія системи

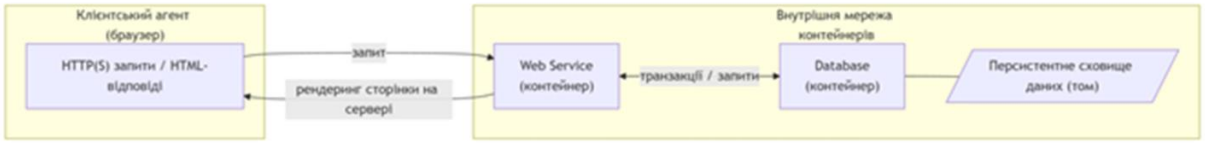

Рисунок Г.30 – Сервісна топологія системи

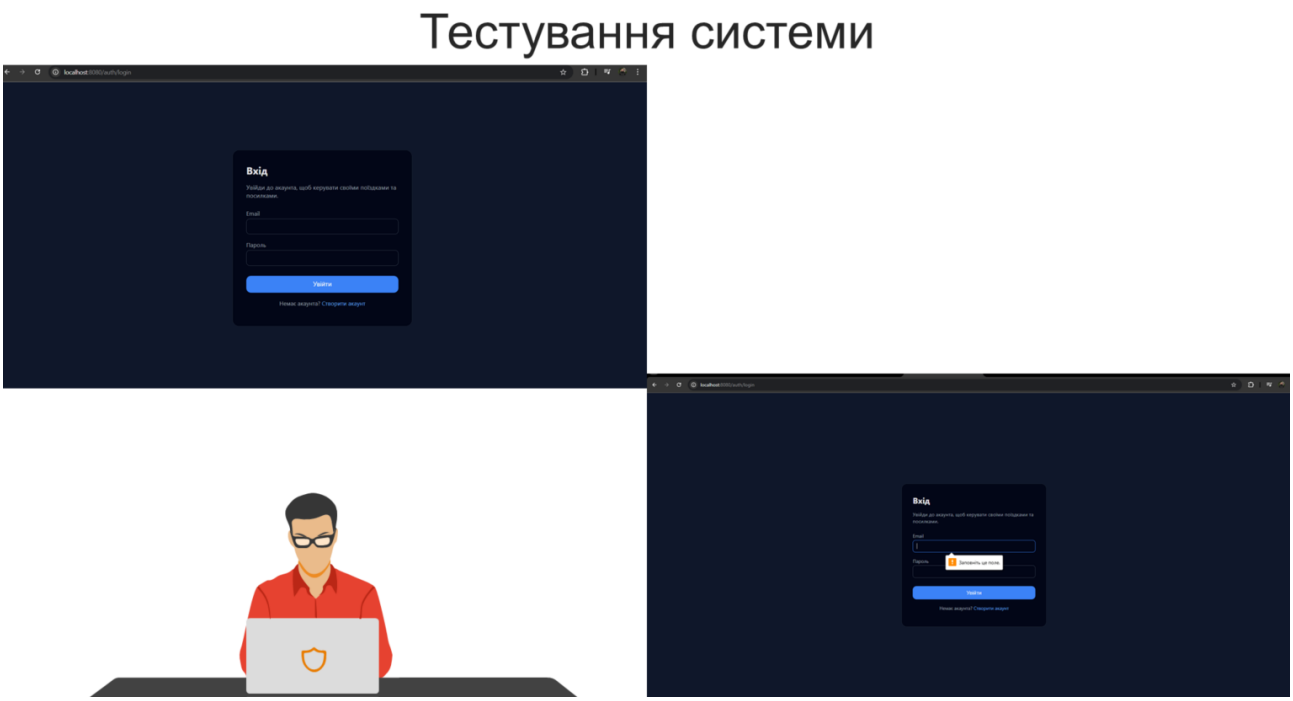

Рисунок Г.31 – Тестування системи (частина 1)

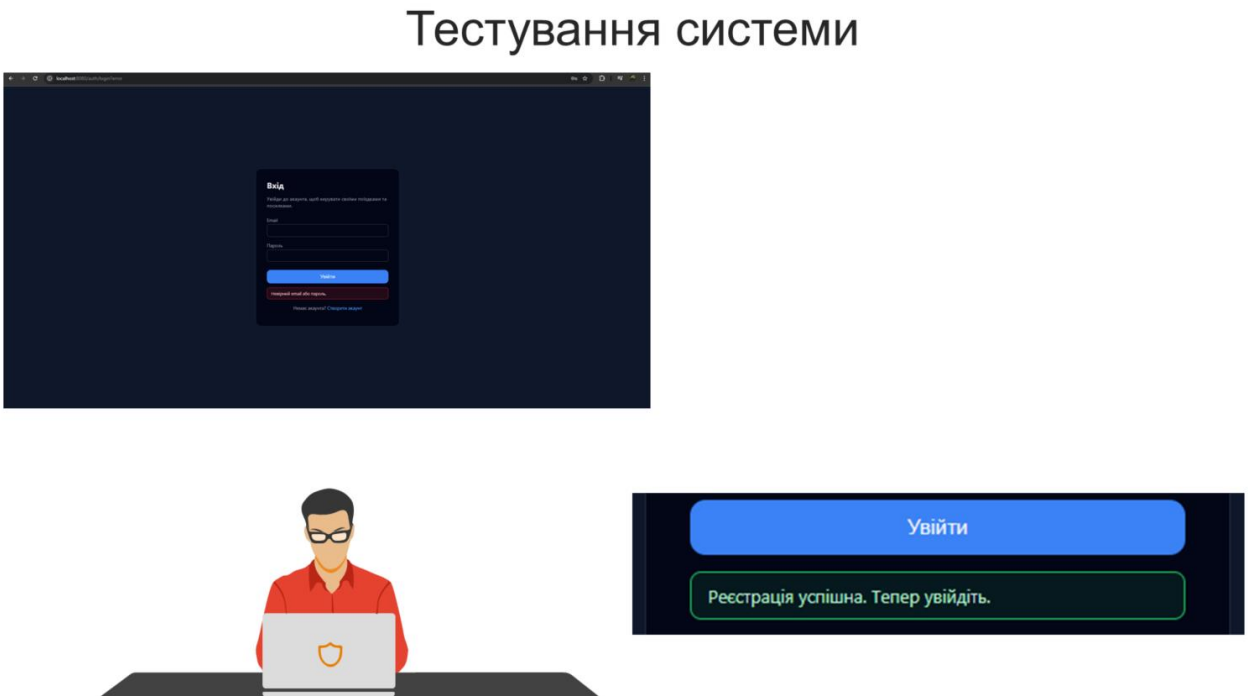

Рисунок Г.32 – Тестування системи (частина 2)

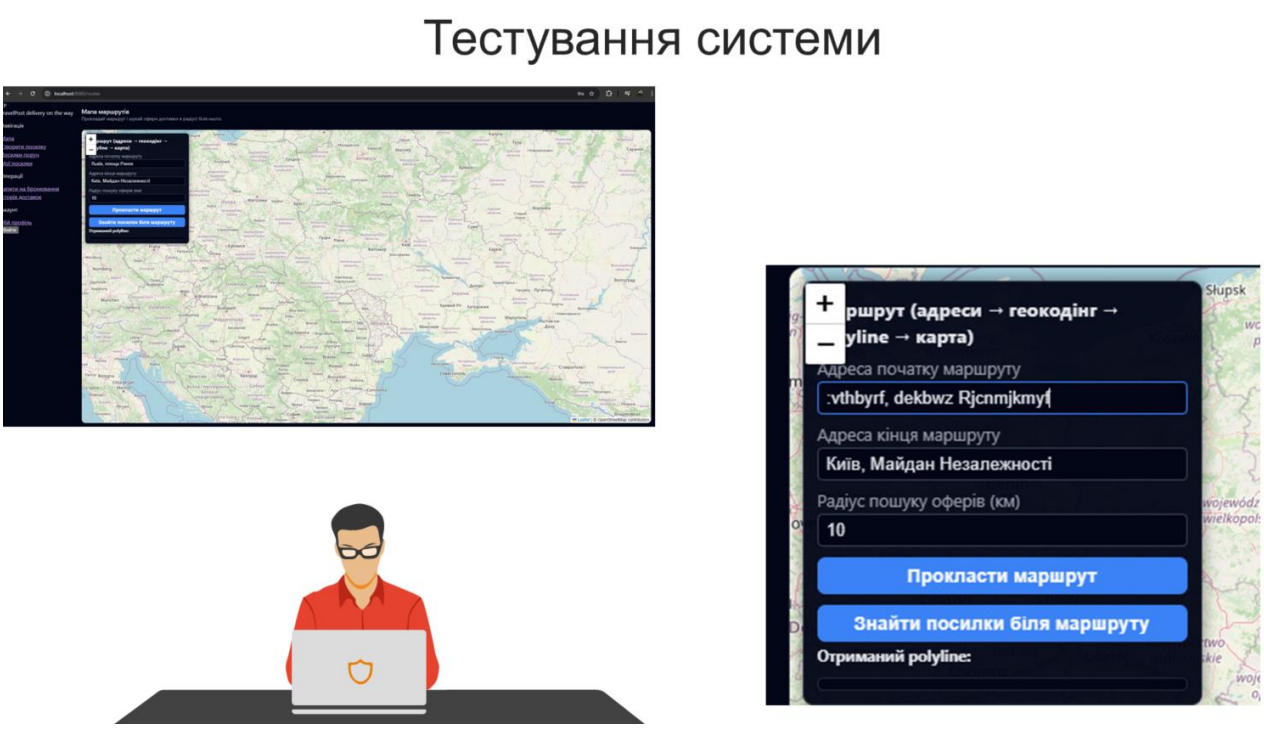

Рисунок Г.33 – Тестування системи (частина 3)

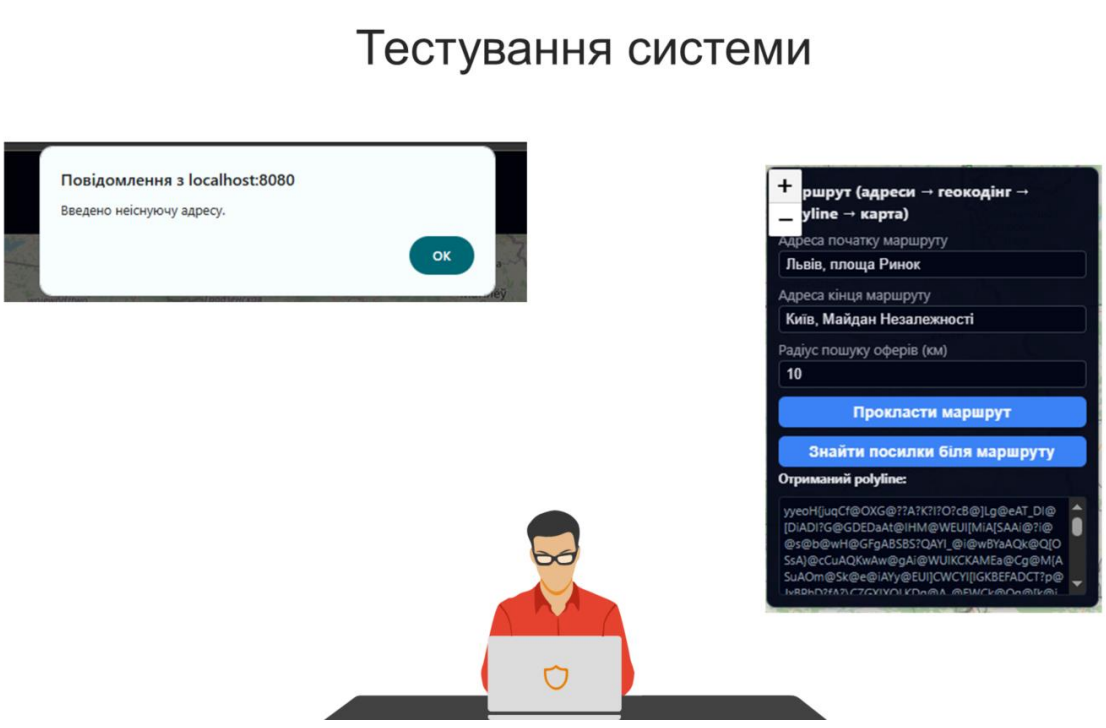

Рисунок Г.34 – Тестування системи (частина 4)

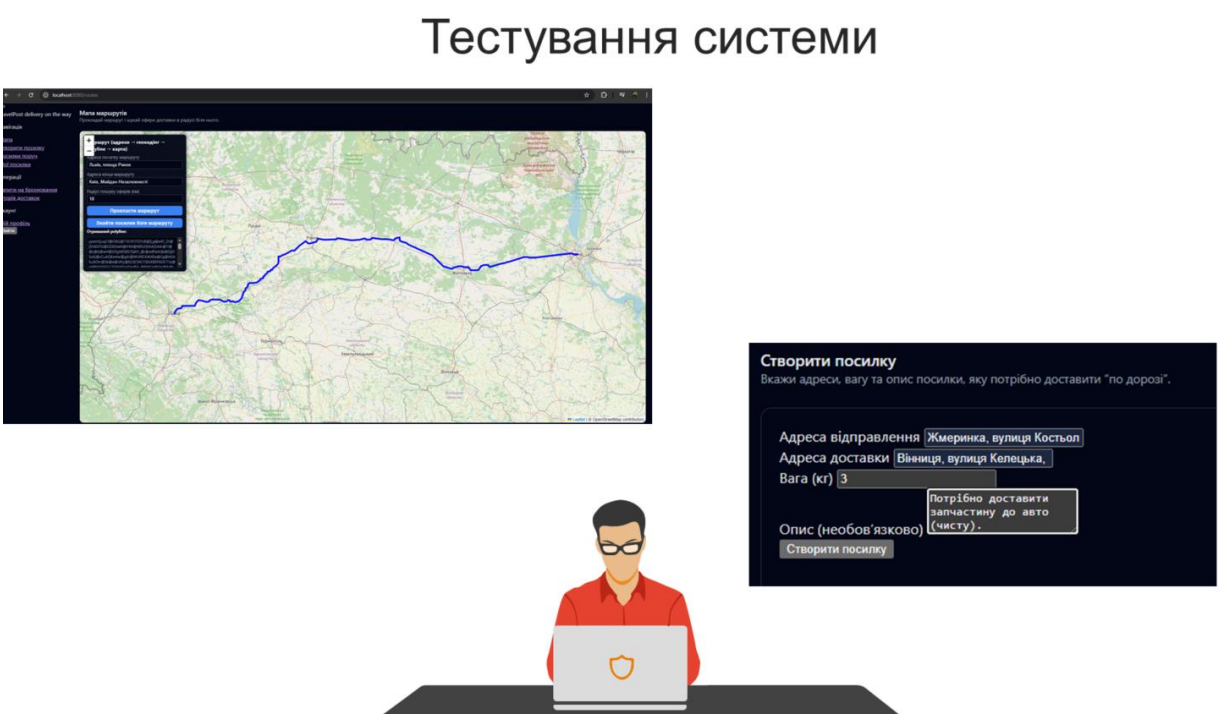

Рисунок Г.35 – Тестування системи (частина 5)

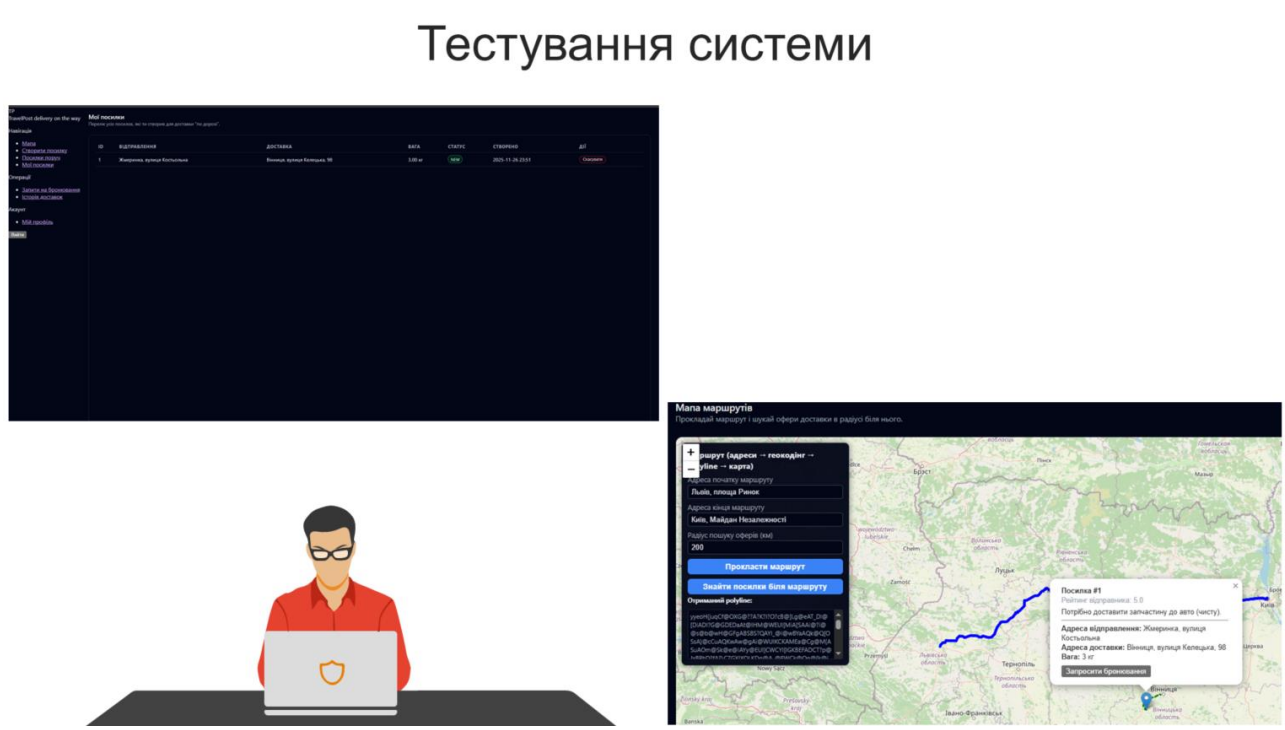

Рисунок Г.36 – Тестування системи (частина 6)

Демонстрація роботи

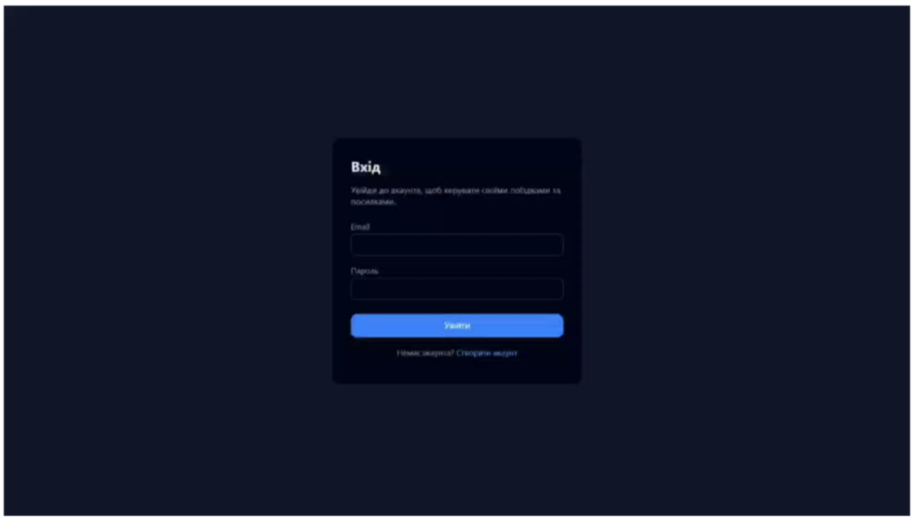

Рисунок Г.37 – Демонстрація роботи

Рисунок Г.38 – Апробація результатів роботи і публікації

Рисунок Г.39 – Висновки

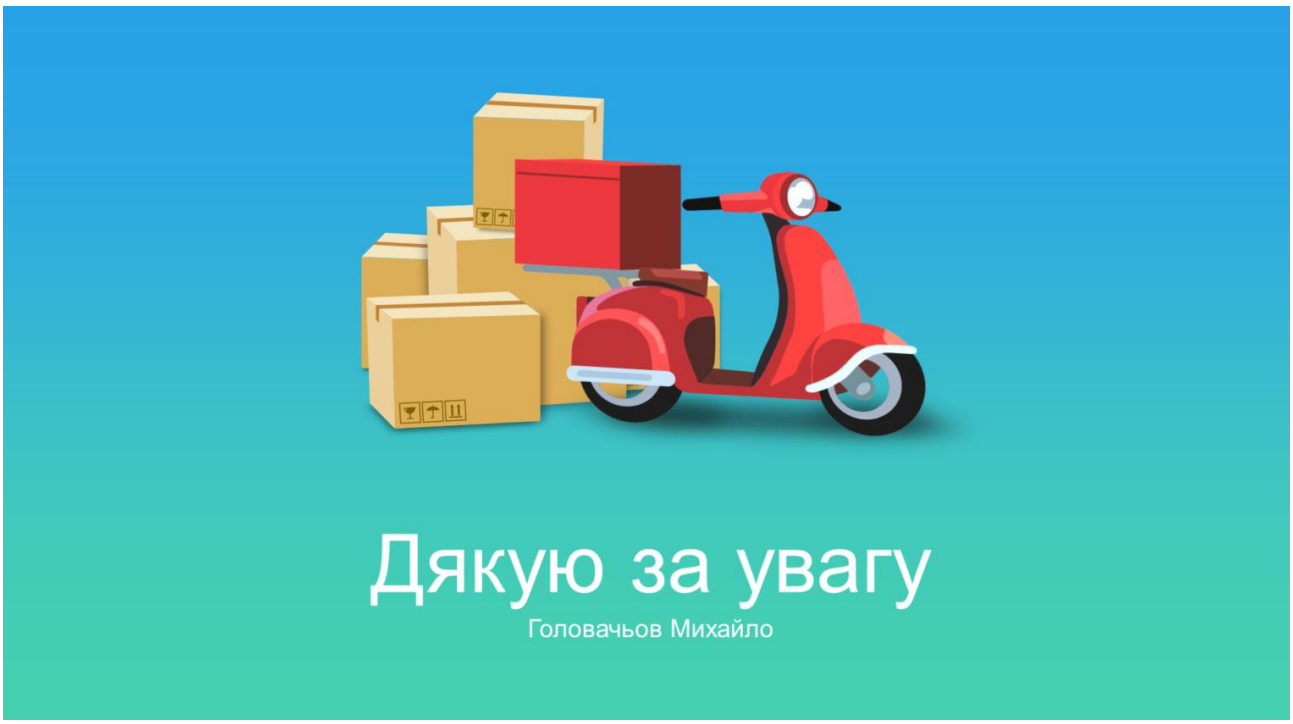

Рисунок Г.39 – Фінальний слайд