Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:
«Дослідження методів оптимізації траєкторії руху транспортних засобів у програмній системі позиціювання в режимі реального часу»

Виконав: студент II курсу
групи 1ПІ-22М спеціальності
121 – Інженерія програмного забезпечення
Гончар С.І.

Керівник: к.т.н., доц. каф. ПЗ Рейда О.М.
«12» грудня 2023 р.

Опонент: к.т.н., доц. каф. ОТ Войцеховська О.В.
«12» грудня 2023 р.

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О. Н.
(прізвище та ініціали)
«12» 12 2023 р.

ВНТУ – 2023

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти II-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.
« 19 » вересня 2023 р.

ЗАВДАННЯ
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ
СТУДЕНТУ

Гончару Сергію Івановичу

1. Тема роботи – Дослідження методів оптимізації траєкторії руху транспортних засобів у програмній системі позиціювання в режимі реального часу.

Керівник роботи: Рейда Олександр Миколайович, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від «18» вересня 2023 р. № 247.

2. Строк подання студентом роботи
5 грудня 2023 р.

3. Вихідні дані до роботи: покращений алгоритм розрахунку траєкторії транспортних засобів; середовища розробки – Visual Studio; мови розробки - Java; JavaScript; операційна система – Windows 10; вихідні дані – програмна система розрахунку траєкторії руху транспортних засобів в режимі реального часу.

4. Зміст текстової частини: дослідження існуючих способів побудови маршруту; постановка задачі дослідження; розробка структури системи; програмна реалізація системи; тестування системи; економічна частина.

5. Перелік ілюстративного матеріалу: контекстна діаграма систем контейнерна діаграма системи; компонентна діаграма системи; скріншо роботи програми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1 - 4	Рейда О. М., к.т.н., доцент кафедри ПЗ	19.09.2023	19.09.2023
5	Причепа І.В., к.е.н., доцент кафедри ЕПВМ	20.11.2023	01.12.2023

7. Дата видачі завдання 19 вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи	Примітка
1	Стан питання та постановка задач дослідження	20.09.2023 – 30.09.2023	вик
2	Розробка структури системи	30.09.2023 – 15.10.2023	вик
3	Програмна реалізація системи	15.10.2023 – 15.11.2023	вик
4	Тестування системи	15.11.2023 – 20.11.2023	вик
5	Економічна частина	20.11.2023 - 1.12.2023	вик

Студент _____ **Гончар С.І.**
(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ **Рейда О.М.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.5

Гончар С.І. Дослідження методів оптимізації траєкторії руху транспортних засобів у програмній системі позиціювання в режимі реального часу. Магістерська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2023. 95 с.

На укр. мові. Бібліогр.: 32 назви; рис.: 35 табл.: 15.

У магістерській кваліфікаційній роботі проведено дослідження та розроблено програмне середовище для оптимізації траєкторії руху транспортних засобів у режимі реального часу з врахуванням різних параметрів, що впливають на ваги графів. Головною метою цього дослідження було покладено вдосконалення двонапрямленого алгоритму Дейкстри з врахуванням різних параметрів, які можуть визначати оптимальність маршруту для користувача.

Досліджено можливості оптимізації траєкторій руху транспортних засобів з використанням алгоритму Дейкстри, ураховуючи фактори, такі як відстань, час, дорожні знаки, тип дороги тощо.

Розроблено програмну систему, яка здатна розраховувати оптимальні траєкторії руху в режимі реального часу для транспортних засобів.

У роботі також проведено економічний аналіз, в якому описано доцільність розробки системи, план витрат на реалізацію науково-дослідної роботи, а також можливі доходи від її впровадження. Досліджено ефективність вкладення інвестицій у проєкт і можливість залучення інвесторів.

Ключові слова: оптимізація траєкторії руху, програмне середовище, алгоритм Дейкстри, візуалізація мапи, економічний аналіз.

ABSTRACT

УДК 004.5

Honchar S.I. Research of optimization methods for the trajectory of vehicles in the positioning system in real-time. Master's qualification work in specialty 121 - software engineering, educational program - software engineering. Vinnytsia: VNTU, 2023. 95 c.

In Ukrainian. Bibliography: 32 titles; Figures: 35; Tables: 15.

In the master's qualification work, a study was conducted, and a software environment was developed to optimize the trajectory of vehicles in real-time, taking into account various parameters that affect the weights of graphs. The main purpose of this study was to improve the bidirectional Dijkstra algorithm, considering various parameters that can determine the optimal route for the user.

The possibilities to optimize vehicle trajectories using the Dijkstra algorithm, considering factors such as distance, time, road signs, road type, etc. were investigated.

A software system has been developed that can calculate optimal trajectories in real time for vehicles.

The paper also conducts an economic analysis, which describes the feasibility of developing the system, the cost plan for the implementation of the research work, and the possible income from its implementation. The effectiveness of investing in the project and the possibility of attracting investors are also investigated.

Keywords: trajectory optimization, software environment, Dijkstra algorithm, map visualization, economic analysis.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Вибір теми дослідження обумовлений необхідністю вдосконалення транспортних систем у зв'язку з їх зростаючою роллю та прогресом у галузі технологій. У сучасному динамічному світі, ключовим аспектом є ефективне управління маршрутами транспортних засобів, що сприяє підвищенню продуктивності перевезень, зменшенню часу подорожі та зниженню витрат на паливо.

Алгоритм побудови маршруту відіграє вирішальну роль в оптимізації траєкторій транспортних засобів у програмній системі позиціонування в реальному часі.

Двонаправлений алгоритм Дейкстри - алгоритм пошуку в графах, який знаходить найкоротший шлях між вузлами графа. Включення цього алгоритму в програмну систему позиціонування в реальному часі дозволяє визначити оптимальний маршрут для транспортних засобів на основі різних факторів.

Крім того, на вагу ребер графу можуть впливати різні параметри, такі як тип дороги, дорожні знаки, обмеження швидкості тощо. Враховуючи ці параметри, процес оптимізації траєкторії можна налаштувати відповідно до конкретних вимог, що призведе до більш точних результатів.

Запропоноване дослідження має на меті модифікувати методи, які використовують двонаправлений алгоритм Дейкстри та налаштування параметрів для ефективної оптимізації траєкторій руху транспортних засобів. Це дослідження зробить внесок у сферу оптимізації перевезень та програмних систем позиціонування в реальному часі, надаючи цінні ідеї та практичні рішення для підвищення ефективності перевезень та зменшення операційних витрат.

Тому актуальними є питання підвищення ефективності методів оптимізації траєкторії руху транспортних засобів у режимі реального часу, оскільки існуючі методи не задовольняють потреби багатьох галузей їх застосування. Це передбачає покращення вже існуючих методів оптимізації траєкторій руху транспортних

засобів.

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

Мета та завдання дослідження. Метою даної роботи є підвищення рівня оптимізації траєкторії руху транспортних засобів у режимі реального часу. Основними задачами дослідження є:

- провести аналіз існуючих методів і засобів оптимізації траєкторії руху транспортних засобів у режимі реального часу;
- модифікувати метод двонаправленого алгоритму Дейкстри для оптимізації траєкторій руху транспортних засобів;
- модифікувати метод адаптивної мультимодальної оптимізації траєкторій, що впливає на вагу ребер графів;
- розробити програмні засоби та систему візуалізації на основі запропонованих методів;
- провести тестування розроблених програмних засобів;

Об’єкт дослідження – процес оптимізації траєкторії руху транспортних засобів в програмній системі позиціонування в реальному часі.

Предмет дослідження – методи та засоби підвищення ефективності оптимізації траєкторій руху транспортних засобів.

Методи дослідження включають теорію графів для вирішення задачі оптимізації, теорія чисел та чисельних методів, методи аналітичної геометрії для розробки моделей та методів оптимізації графів; комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Наукова новизна одержаних результатів.

1. Подальшого розвитку отримав метод двонаправленого алгоритму Дейкстри для оптимізації траєкторій руху транспортних засобів, що на відміну від існуючого, використовує процес оптимізації з ваговими коефіцієнтами і дозволяє швидше і точніше планувати маршрути.

2. Подальшого розвитку отримав метод адаптивної мультимодальної оптимізації траєкторій, що оптимізує ваги ребер графів, використовуючи таку інформацію як: дорожні знаки, обмеження швидкості, об’їзд об’єктів для побудови альтернативних маршрутів і адаптації до типу дороги, що на відміну від існуючого методу, дозволяє підвищувати точність та адаптованість маршрутів при навігації.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в магістерській кваліфікаційній роботі теоретичних положень модифіковано метод двонаправленого алгоритму Дейкстри та метод адаптивної мультимодальної оптимізації траєкторій для оптимізації траєкторій руху транспортних засобів у програмній системі позиціонування в реальному часі та розроблено програмні засоби та систему візуалізації на основі запропонованих методів для комп’ютерних систем позиціювання у режимі реального часу.

Особистий внесок здобувача. Усі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто. У друкованих працях, опублікованих у співавторстві, автору належать такі результати: розглянуто алгоритм «A*»[1]; вдосконалено алгоритм Дейкстри[2]; комп’ютерна програма для побудови маршрутів транспортних засобів у режимі реального часу.

Апробація матеріалів магістерської кваліфікаційної роботи. Основні положення магістерської кваліфікаційної роботи доповідалися та обговорювалися на всеукраїнських конференціях: VI Всеукраїнській науково-практичній інтернет-конференції молодих вчених та студентів «Сучасні інформаційні системи та технології» (Херсонський національний технічний університет, 2023); Науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії, секція програмного забезпечення (Вінниця, 2023).

Публікації. Основні результати досліджень опубліковано в 2 наукових працях, у матеріалах конференцій.

1 СТАН ПИТАННЯ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану питання

У сучасному швидкоплинному світі ефективна навігація та оптимізація траєкторій руху транспортних засобів набули першочергового значення. Програмні системи позиціонування в реальному часі відіграють вирішальну роль у досягненні цієї мети, забезпечуючи точне і своєчасне прокладання маршруту.

Оптимізація траєкторії являє собою процес визначення найбільш ефективного і практичного маршруту для досягнення транспортним засобом місця призначення. У контексті програмних систем позиціонування в реальному часі вона передбачає використання алгоритмів і методів для розрахунку оптимальних маршрутів на основі різних факторів, таких як дорожні умови, завантаженість доріг і переваги користувачів. Оптимізуючи маршрут транспортних засобів, ці системи підвищують ефективність пересування, зменшують споживання пального та підвищують загальну задоволеність користувачів. Оптимізовані траєкторії руху транспортних засобів скорочують час у дорозі, визначаючи найкоротші та найшвидші маршрути, заощаджуючи час та зменшуючи споживання пального. Крім того, оптимізація траєкторії забезпечує точні покрокові вказівки, що дозволяє водіям впевнено орієнтуватися в незнайомих районах. Персоналізовані рекомендації щодо маршруту, засновані на вподобаннях користувача, ще більше покращують користувацький досвід [3].

Алгоритми оптимізації траєкторії допомагають ефективніше керувати транспортними потоками, розподіляючи транспортні засоби за альтернативними маршрутами, розвантажуючи затори та зменшуючи вузькі місця. Крім того, ці алгоритми враховують такі важливі параметри, як дорожні знаки, обмеження швидкості та потенційні небезпеки, сприяючи більш безпечним поїздкам для водіїв і пасажирів.

1.2 Аналіз алгоритмів побудови шляху

Алгоритми побудови шляхів відіграють важливу роль в оптимізації процесів планування маршрутів, навігації та прийняття рішень. Ці алгоритми визначають найефективніші шляхи між точками в різних додатках, включаючи програмні системи позиціонування в реальному часі. Основним завданням цих алгоритмів є знаходження найкоротшого або найшвидшого шляху між двома точками, враховуючи різні обмеження та параметри, такі як довжина шляху, час подорожі, пропускна спроможність доріг та актуальні дорожні умови.

Сучасні системи позиціонування в реальному часі вимагають від алгоритмів побудови шляху не тільки точності, але й швидкості обчислень, адже умови на дорогах можуть швидко змінюватися. Тому важливо використовувати алгоритми, які можуть швидко адаптуватися до змін у дорожньому русі та інших зовнішніх факторах. Це включає інтеграцію з системами збору даних в реальному часі, такими як датчики трафіку, GPS-трекінг та інші джерела інформації.

Крім того, важливим аспектом є масштабованість алгоритмів. З розвитком міської інфраструктури та зростанням кількості транспортних засобів, системи позиціонування повинні ефективно обробляти великі обсяги даних та забезпечувати точне планування маршрутів у складних мережах доріг. Це вимагає від алгоритмів побудови шляху високої ефективності та здатності швидко обробляти зміни в дорожніх умовах.

Алгоритм «A*» (A-зірка) це популярний алгоритм побудови шляхів, який поєднує методи пошуку в глибину і в ширину. Він використовує евристики для оцінки вартості досягнення мети з певної точки шляху. Враховуючи як фактичні витрати, понесені до цього моменту, так і передбачувану вартість досягнення мети, алгоритм «A*» ефективно знаходить оптимальний шлях [4]. Цей алгоритм широко використовується в різних сферах, включаючи робототехніку, розробку ігор та GPS-навігаційні системи.

Блок-схема алгоритму «A*»[5] зображено на рисунку 1.1.

Рисунок 1.1 – Блок-схема алгоритму «A*»

Головна перевага алгоритму «A*» полягає в тому, що він ефективно використовує інформацію про оцінку вартості досягнення мети (евристику). Це дозволяє йому спрямовувати пошук у напрямку мети, зменшуючи кількість розглянутих точок шляху і забезпечуючи більш ефективний пошук найкоротшого шляху.

Алгоритм Дейкстри, який використовується для пошуку найкоротшого шляху в графі з невід’ємними вагами ребер, є одним з найбільш відомих та широко використовуваних алгоритмів у області комп'ютерних наук. Його основний принцип роботи полягає у встановленні початкових значень відстаней до всіх вузлів графу та їх подальшому ітеративному оновленні шляхом дослідження сусідніх вузлів.

На початку роботи алгоритму, відстань до початкового вузла встановлюється рівною нулю, а відстані до всіх інших вузлів - нескінченністю. Потім алгоритм послідовно переглядає всі вузли графу, починаючи з вузла з найменшою відстанню, яка ще не була оброблена. Для кожного такого вузла алгоритм розглядає всі його сусідні вузли та оновлює відстані до них, якщо відстань через поточний вузол є коротшою.

Однією з ключових особливостей алгоритму Дейкстри є його ефективність у випадках, коли ваги ребер не є від'ємними. Це робить його ідеальним для застосувань, де ваги ребер можуть представляти відстані, час, вартість або інші метрики, що не можуть бути від'ємними.

Алгоритм Дейкстри може бути реалізований з використанням різних структур даних, таких як пріоритетні черги, що може значно підвищити його продуктивність, особливо у великих графах. Це робить його популярним вибором у багатьох областях, включаючи комп'ютерну графіку, мережеве маршрутизування та планування маршрутів.

Кроки алгоритму Дейкстри можна описати наступним чином [6] (рисунок. 1.2)

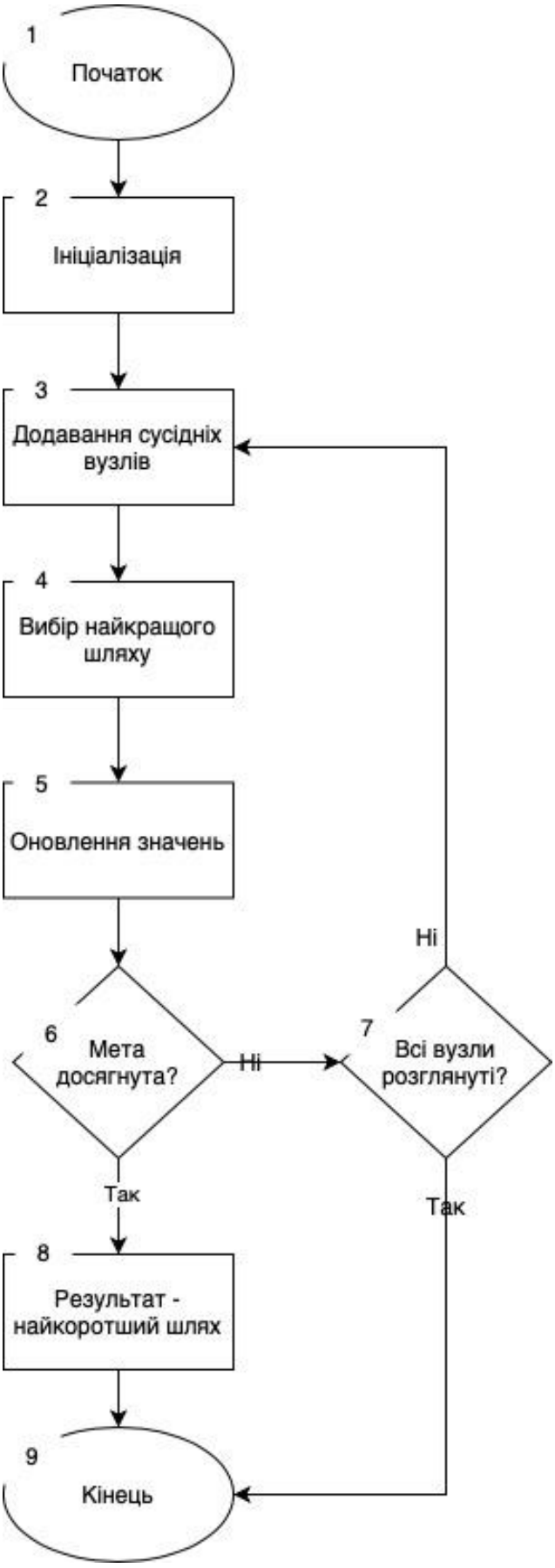

Рисунок 1.2 – Блок-схема алгоритму Дейкстри

Головна перевага алгоритму Дейкстри полягає в тому, що він гарантує знаходження найкоротшого шляху в графі з невід'ємними вагами ребер. Він також ефективно працює на графах без циклів з від'ємними ребрами. Однак, алгоритм Дейкстри не враховує евристики або конкретних цілей, що може призвести до пошуку некоректного шляху у деяких випадках.

Алгоритм Беллмана-Форда є ще одним важливим алгоритмом у сфері пошуку найкоротших шляхів, особливо в графах, де ребра можуть мати негативну вагу. Цей алгоритм відрізняється від алгоритму Дейкстри тим, що він може ефективно обробляти графи з ребрами негативної ваги, що робить його більш універсальним у певних ситуаціях. Основний принцип роботи алгоритму Беллмана-Форда полягає в ітеративному оновленні вартостей шляхів до всіх вузлів графу. Алгоритм послідовно проходить через всі ребра графу та оновлює відстані до вузлів, якщо виявляється, що існуючий шлях через дане ребро є коротшим. Цей процес повторюється кілька разів, зазвичай стільки, скільки вузлів у графі, мінус один.

Алгоритм Беллмана-Форда може бути використаний для розробки більш гнучких та адаптивних систем маршрутизації. Наприклад, у міському плануванні та управлінні трафіком, де умови дорожнього руху швидко змінюються, алгоритм може допомогти в ідентифікації альтернативних маршрутів, які враховують змінні умови, такі як затори або дорожні роботи.

Крім того, алгоритм Беллмана-Форда може бути інтегрований з іншими технологіями, такими як машинне навчання та аналіз великих даних, для створення більш точних та ефективних систем планування маршрутів.

Його застосування в програмних системах позиціонування в реальному часі може значно підвищити ефективність та адаптивність цих систем, забезпечуючи більш точне та гнучке планування маршрутів.

Блок схема алгоритму Беллмана-Форда[7] зображено на рисунку 1.3

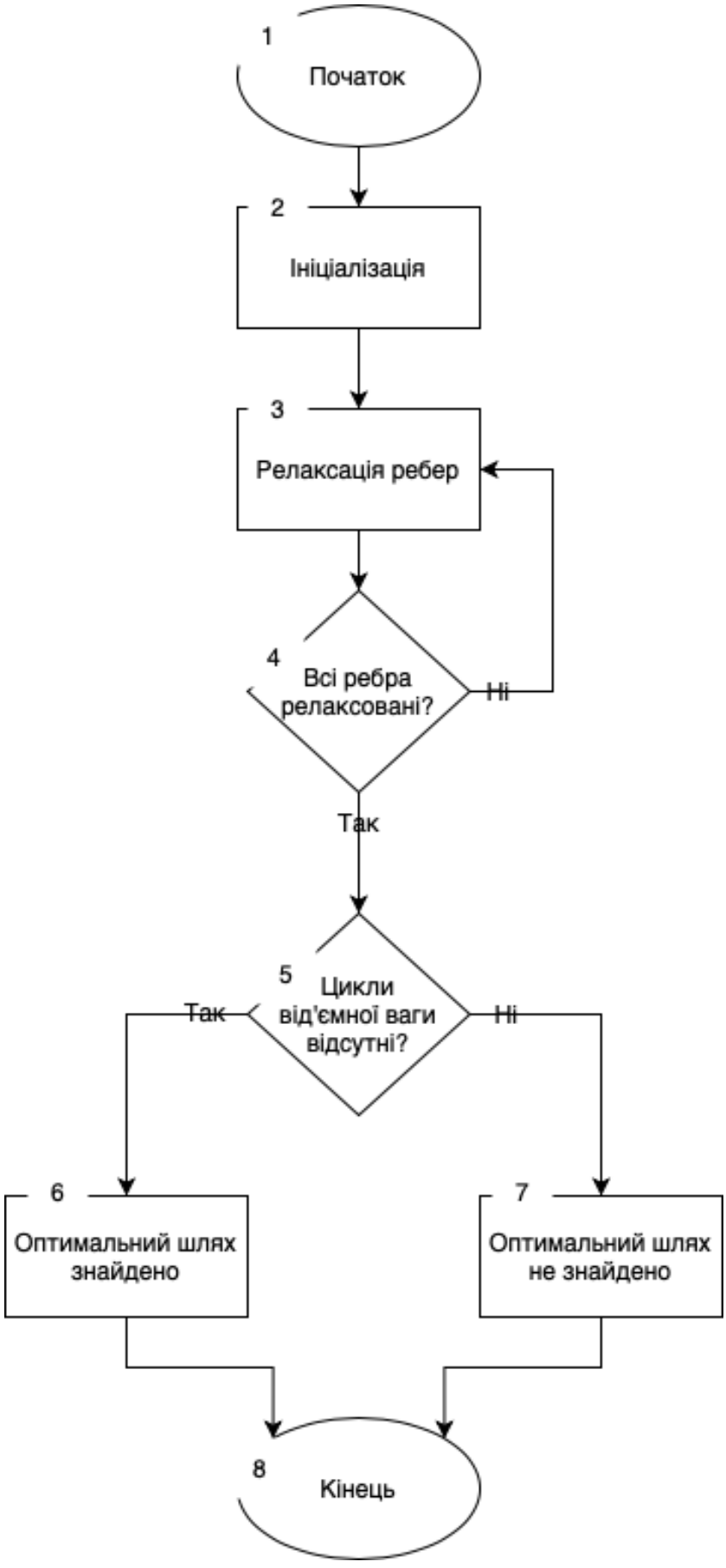

Рисунок 1.3 - Блок схема алгоритму Беллмана-Форда

Алгоритм Беллмана-Форда дозволяє знайти найкоротший шлях в графі із ваговими ребрами, включаючи випадки, коли граф може містити ребра негативної ваги. Однак, його час виконання є більшим ніж для алгоритмів, які працюють з графами без ребер від’ємної ваги.

Генетичні алгоритми[8], які є частиною ширшої категорії еволюційних алгоритмів, використовують принципи природного відбору та генетики для розв'язання задач оптимізації. Ці алгоритми імітують процеси, які відбуваються в природі, такі як спадковість, мутації, відбір та кросинговер, для пошуку оптимальних або близьких до оптимальних рішень у складних проблемах.

У контексті побудови шляхів, генетичні алгоритми починають з генерації популяції потенційних рішень, які представляють різні маршрути. Кожен маршрут оцінюється за допомогою функції пристосування, яка визначає, наскільки добре він вирішує поставлену задачу. Наприклад, у випадку пошуку найкоротшого шляху, функція пристосування може оцінювати довжину шляху або час, необхідний для його проходження.

Процес еволюції в генетичних алгоритмах включає вибір найкращих індивідів (маршрутів) для створення наступного покоління. Це може включати використання операцій кросинговеру (комбінування частин двох маршрутів для створення нового) та мутацій (випадкові зміни в маршруті). Ці операції допомагають забезпечити генетичне різноманіття в популяції та збільшують шанси знайти оптимальне рішення.

Генетичні алгоритми особливо ефективні у випадках, коли простір пошуку є великим або дуже складним для традиційних методів оптимізації. Вони можуть знаходити хороші рішення навіть у випадках, коли точне рішення неможливо знайти або воно вимагає неприпустимо великих обчислювальних ресурсів.

Загальний принцип роботи цих алгоритмів зображений на рисунку 1.4

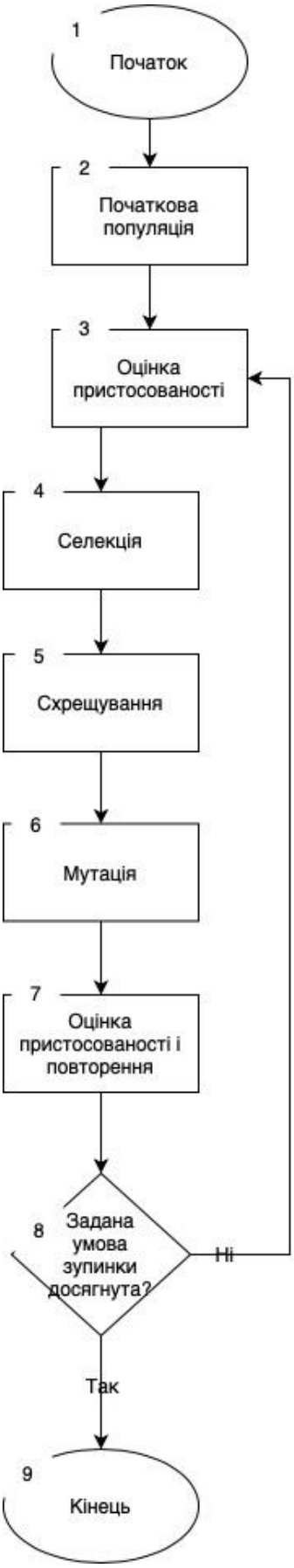

Рисунок 1.4 – Блок-схема генетичних алгоритмів

Алгоритм Флойда-Уоршалла[9] це алгоритм для пошуку оптимального маршруту в графі, який вирішує проблему найкоротшого шляху між усіма парами вершин. Основний принцип роботи алгоритму зображений на рисунку 1.5.

Рисунок 1.5 – Блок-схема алгоритму Флойда-Уоршалла

Алгоритм Флойда-Уоршалла є ефективним для використання в графах з вагами ребер, які можуть бути від’ємними або додатними. Він дозволяє знайти найкоротший шлях між усіма парами вершин і може бути корисним для задач таких як маршрутизація в комп’ютерних мережах або планування маршрутів транспорту.

Підходи на основі штучного інтелекту, включаючи методи машинного навчання і глибокого навчання, відкривають нові можливості у вирішенні проблем побудови маршрутів. Ці технології використовують великі масиви даних для вивчення закономірностей та прийняття інтелектуальних рішень, що може значно підвищити ефективність та точність у плануванні маршрутів. Одним з ключових напрямків у цій області є алгоритми навчання з підкріпленням, які вивчають оптимальні стратегії поведінки, взаємодіючи з навколишнім середовищем і отримуючи зворотний зв’язок від нього. Ці алгоритми здатні адаптуватися до змін у середовищі та вчитися з досвіду, що робить їх особливо ефективними в динамічних і невизначених умовах.

Підходи на основі штучного інтелекту також включають використання нейронних мереж для моделювання складних залежностей та визначення оптимальних маршрутів. Наприклад, глибоке навчання може бути використане для аналізу великих обсягів даних про трафік, погодні умови, дорожні інциденти тощо, щоб передбачити оптимальні маршрути в реальному часі[10]. Це дозволяє системам позиціонування не тільки реагувати на поточні умови, але й антиципувати майбутні зміни, підвищуючи тим самим загальну ефективність та надійність планування маршрутів.

Крім того, використання штучного інтелекту в системах позиціонування відкриває можливості для персоналізації маршрутів. Завдяки аналізу індивідуальних звичок та переваг користувачів, системи можуть пропонувати маршрути, які не тільки є оптимальними з точки зору часу та відстані, але й враховують особисті переваги, такі як уникнення певних доріг або районів.

Основний алгоритм роботи таких систем зображений на рисунку 1.6

Рисунок 1.6 – Блок-схема основного алгоритму пошуку маршруту на основі штучного інтелекту

Застосування підходів на основі штучного інтелекту до оптимізації побудови маршрутів може значно покращити ефективність та точність вибору оптимального маршруту, особливо в динамічних і невизначених середовищах.

Це дозволяє забезпечити користувачам швидке та ефективне переміщення, зменшуючи час подорожі та ресурси, а також покращуючи загальний досвід користувачів.

1.3 Порівняння алгоритмів та шляхи вдосконалення

У сфері оптимізації траєкторій руху транспортних засобів, особливо в контексті програмних систем позиціювання в режимі реального часу, важливо вибрати найбільш ефективний алгоритм для визначення маршрутів [11].

Для цього потрібно враховувати ряд критеріїв, таких як:

– швидкість виконання;
– точність;
– складність реалізації;
– робота в динамічному середовищі;
– підтримка великих мереж;
– використання додаткових даних.

З урахуванням цих критерії та вище описаних алгоритмів було створено порівняльну таблицю 1.1.

Таблиця 1.1 — Порівняльна характеристика алгоритмів пошуку маршруту

Критерій / Алгоритм	**«A*»**	**Дейкстри**	**Беллмана Форда**	**Генетичні**	**Флойда Уоршалла**	**ШІ**
Швидкість виконання	8	9	6	5	3	5
Точність	9	9	9	9	9	9
Складність Реалізації	7	9	5	3	4	2
Робота в динамічному середовищі	9	7	7	9	1	8

Продовження таблиці 1.1 - Порівняльна характеристика алгоритмів пошуку маршруту

Критерій / Алгоритм	**«A*»**	**Дейкстри**	**Беллмана Форда**	**Генетичні**	**Флойда Уоршалла**	**ШІ**
Підтримка великих мереж	9	9	9	9	9	9
Використання додаткових даних	6	6	2	9	2	9
Сумарний коефіцієнт	48	49	38	44	28	42

На основі проведеного порівняння можна зробити висновок, що Алгоритм Дейкстри є найкращим вибором для реалізації системи оптимізації траєкторій руху транспортних засобів у програмній системі позиціювання в режимі реального часу. Хоча деякі алгоритми, такі як «A*» та алгоритми на основі ШІ, показують високу ефективність у певних сценаріях, Алгоритм Дейкстри переважає завдяки своїй високій швидкості виконання, простоті реалізації, а також здатності точно і ефективно обробляти великі мережі. Хоча він має певні обмеження у динамічних середовищах, для більшості застосувань в реальному часі ці обмеження не є критичними. Таким чином, Алгоритм Дейкстри є оптимальним вибором для забезпечення ефективного та надійного позиціювання в реальному часі.

Алгоритм Дейкстри є одним з найбільш використовуваних алгоритмів для пошуку найкоротших шляхів у графах, але як і будь-який алгоритм, він має потенціал для вдосконалення. Ось кілька шляхів, якими можна покращити його ефективність[12]:

– Використання черги з пріоритетом: Заміна простого списку черги з пріоритетом може значно зменшити час виконання алгоритму. Це дозволяє

швидше знаходити вершину з найменшою оцінкою відстані. Черга з пріоритетом оптимізує процес видалення вершини з найменшою відстанню, знижуючи цю операцію до O(log n) замість O(n).

– Зменшення кількості Релаксацій (кроку, який використовується для оновлення оцінок найкоротших відстаней від початкової вершини до всіх інших вершин у графі): Перед тим, як виконувати релаксацію ребра, можна перевірити, чи потенційно це ребро може зменшити відстань до вершини. Такий підхід вимагає додаткової перевірки перед кожною релаксацією, але це може зменшити загальну кількість операцій, особливо в густо з'єднаних графах.

– Використання хеш-таблиць: Застосування хеш-таблиць для зберігання вже оброблених вершин допоможе швидше перевіряти, чи вже була релаксована дана вершина. Хеш-таблиці можуть забезпечити майже миттєвий пошук і оновлення стану вершин, що значно підвищує швидкість виконання алгоритму в певних сценаріях.

– Двонапрямлений Пошук: Двонапрямлений пошук є одним із найефективніших способів покращення алгоритму Дейкстри. Він працює шляхом одночасного запуску пошуку з початкової та кінцевої точок і зустрічається десь посередині. Ініціалізація, пошук запускається одночасно від початкової та кінцевої точок. Пошук припиняється, коли дві пошукові хвилі зустрічаються. Шлях реконструюється шляхом об'єднання двох частин, знайдених обома пошуками. Цей метод значно зменшує кількість вершин, які потрібно обробити, що призводить до зниження часу виконання, особливо в великих графах.

Використання методів вдосконалення, таких як двонапрямлений пошук, може значно підвищити ефективність алгоритму Дейкстри, особливо у контексті систем оптимізації траєкторій руху в реальному часі. Цей підхід дозволяє алгоритму працювати одночасно з двох кінців шляху – від початкової та кінцевої точок, що значно знижує загальний час обчислень, необхідний для знаходження оптимального маршруту.

1.4 Постановка задачі

Після аналізу існуючих підходів та методів у сфері пошуку маршрутів, а також ретельного порівняння різних алгоритмів, було вирішено вибрати двонаправлений алгоритм Дейкстри як найбільш оптимальний для реалізації в програмній системі. Цей вибір обумовлений його високою ефективністю та гнучкістю в різних сценаріях використання. На основі цього рішення було визначено кілька ключових завдань, які необхідно виконати для успішної реалізації системи:

– Вибір шаблону проектування та інструментів розробки: Це включає в себе аналіз та вибір найбільш підходящих програмних мов, фреймворків та інструментів, які забезпечать гнучкість, масштабованість та ефективність розробки. Важливо також врахувати інтеграцію з іншими системами та можливість легкого оновлення системи в майбутньому.

– Реалізація відображення роботи алгоритму на мапі: Це завдання передбачає створення інтерактивного інтерфейсу, який дозволить користувачам візуалізувати маршрути, згенеровані алгоритмом. Важливо забезпечити точність та чіткість відображення, а також інтуїтивно зрозумілий інтерфейс.

– Заповнення мапи додатковою інформацією: Це включає інтеграцію даних про дорожні умови, затори, обмеження швидкості, дорожні знаки та інші важливі параметри, які можуть впливати на вибір маршруту. Це дозволить системі генерувати більш точні та ефективні маршрути.

– Програмна реалізація додаткового функціоналу алгоритму: Розробка додаткових функцій, таких як альтернативні маршрути, прогнозування часу в дорозі, адаптація до змін у дорожньому русі, що забезпечить більш гнучке та адаптивне планування маршрутів.

– Тестування розробленої системи: Це допоможе виявити та виправити можливі помилки та недоліки перед запуском системи в експлуатацію.

Детальний опис технічних вимог та специфікацій представлено у Додатку А. Цей етап є критично важливим для забезпечення успішної реалізації проекту,

оскільки він визначає основні напрямки роботи та очікувані результати.

Виконання цих завдань дозволить створити ефективну та надійну систему, яка зможе задовольнити потреби користувачів та вирішити актуальні проблеми в області транспортного планування.

1.5 Висновки

У першому розділі магістерської кваліфікаційної роботи здійснено детальний аналіз методів оптимізації траєкторій руху транспортних засобів у реальному часі. Враховуючи актуальність ефективної навігації, досліджено основні аспекти та виклики, пов'язані з вибором та імплементацією алгоритмів.

Було розглянуто значення програмних систем позиціонування в реальному часі для точного та своєчасного прокладання маршрутів та те, як оптимізація траєкторій сприяє підвищенню ефективності пересування, зниженню споживання пального та підвищенню задоволеності користувачів.

Проаналізовано різні алгоритми побудови маршрутів, висвітлюючи їхні переваги та обмеження в залежності від конкретних умов застосування.

Проведено порівняльний аналіз цих алгоритмів, враховуючи критерії, такі як швидкість обчислень, точність, складність реалізації, адаптивність до змін у середовищі, підтримка великих мереж та інтеграція додаткових даних.

Виявлено, що алгоритм Дейкстри є оптимальним для систем позиціонування в реальному часі, хоча інші алгоритми також можуть бути ефективними в певних ситуаціях. Запропоновано покращення для алгоритму Дейкстри, включаючи використання черги з пріоритетами, зменшення кількості релаксацій, застосування хеш-таблиць та двонапрямлений пошук.

Таким чином, у цьому розділі були розглянуті та проаналізовані сучасні методи оптимізації траєкторій руху транспортних засобів у режимі реального часу, а також надані рекомендації для їх покращення та ефективного застосування в практичних ситуаціях. Визначено завдання для подальшого виконання в рамках магістерської кваліфікаційної роботи.

2 РОЗРОБКА СТРУКТУРИ СИСТЕМИ

2.1 Вибір інструментів програмної реалізації

Вибір правильного технологічного стеку для веб-додатку, відіграє значну роль як у створенні так і у подальшій підтримці системи. Одним з вирішальних факторів є вибір відповідних мов програмування як для фронтенду, так і для бекенд-розробки. Для реалізації системи було обрано JavaScript для фронтенду та Java для бекенду.

JavaScript стала мовою для фронтенд-розробки завдяки кільком ключовим перевагам. Перш за все, JavaScript підтримується всіма сучасними веб-браузерами, що робить її дуже сумісною та доступною на різних платформах. Це забезпечує безперебійну роботу користувача незалежно від пристрою чи браузера, що використовується [13]. Враховуючи здатність JavaScript створювати динамічні та інтерактивні веб-сайти за допомогою таких фреймворків, як React (розроблений Facebook), Angular (розроблений Google) або Vue.js, робить вибір цієї мови для реалізації клієнтської сторони системи аргументованим та очевидним. Ще однією значною перевагою JavaScript можна вважати сильну та поширену спільноту. Існує незліченна кількість бібліотек, фреймворків та інструментів, які полегшують процес розробки. Від компонентів інтерфейсу користувача до візуалізації даних, JavaScript може похвалитися широким спектром ресурсів, які спрощують фронтенд-розробку та підвищують продуктивність.

Щодо серверної частини системи (бекенд), було обрано мову програмування Java, яка вже давно є популярним вибором для розробки, і на те є вагомі причини. Сила Java полягає в її надійності, продуктивності та масштабованості. Вона відома своєю здатністю з легкістю обробляти складні додатки корпоративного рівня. Сувора типізація та об’єктно-орієнтована природа Java сприяють підтримці коду та заохочують до належних практик програмної інженерії. Широка екосистема Java, включаючи фреймворк Spring Boot, розроблений Pivotal Software, надає безліч бібліотек, фреймворків та

інструментів, які спрощують бекенд-розробку. Spring Boot пропонує комплексні рішення для створення REST API, обробки взаємодії з базами даних та управління конфігураціями додатків [14]. Зрілість і стабільність Java роблять її чудовим вибором для масштабних проектів, які потребують надійності та довготривалої підтримки.

Також, в системі для клієнтської сторони були використані наступні додаткові інструменти:

– Vue.js, розроблений Еваном Ю, є прогресивним JavaScript-фреймворком, призначеним для розробки користувацьких інтерфейсів [15]. Його особливість полягає у гнучкості для поступового впровадження, дозволяючи використовувати його в окремих частинах системи з можливістю подальшого розширення. Vue.js пропонує реактивний та компонентно-орієнтований підхід до створення веб-додатків, що сприяє ефективній розробці;

– Leaflet, розроблена Володимиром Агафонкіним, є JavaScript бібліотекою, призначеною для створення інтерактивних карт [16]. Ця бібліотека відрізняється своєю простотою у використанні та легкістю, роблячи її ідеальним вибором для інтеграції карт у веб-додатки. Leaflet сумісна з даними від різноманітних постачальників картографічних даних, включаючи OpenStreetMap, Bing та Mapbox. Завдяки великій кількості доступних плагінів, Leaflet надає розробникам можливість додавати до карт різноманітні додаткові функції, такі як маркери, полігони та шари;

– Vuetify, розроблена Джоном Лейтхедом, є бібліотекою компонентів інтерфейсу користувача для Vue.js, яка відповідає принципам Material Design [17]. Ця бібліотека надає широкий спектр готових до використання компонентів, таких як кнопки, картки, діалогові вікна, форми та елементи навігації. Vuetify полегшує процес розробки, забезпечуючи консистентність стилю та функціональності на різноманітних пристроях і в браузерах. Використання Vuetify дозволяє розробникам швидко створювати прототипи та естетично привабливі, адаптивні користувацькі інтерфейси;

– Axios, створений Маттіасом Кріненбергом, є HTTP-клієнтом для браузера та Node.js. Цей інструмент забезпечує елегантний та зручний API для виконання HTTP-запитів та обробки відповідей. Завдяки Axios, розробники можуть легко виконувати різноманітні типи запитів, такі як GET, POST, PUT, DELETE [18]. Він пропонує такі можливості, як перехоплення запитів і відповідей, автоматичне конвертування в JSON та ефективну обробку помилок. Axios є популярним вибором у проектах, що використовують Vue.js, для здійснення API-викликів та управління обміном даними між фронтендом і бекендом;

Ці елементи працюють разом, щоб покращити функціональність та користувацький досвід інтерфейсу додатку. Vue.js слугує основним фреймворком для побудови користувацького інтерфейсу, тоді як Leaflet дозволяє інтегрувати інтерактивні карти. Vuetify надає набір попередньо визначених компонентів для узгодженого дизайну, а openrouteservice-js полегшує можливості маршрутизації. Нарешті, axios спрощує HTTP-зв’язок з внутрішнім сервером.

Для серверної сторони були використані наступні додаткові інструменти:

– Spring Boot, розроблений командою Pivotal Software, є фреймворком, який значно спрощує процес конфігурації та розгортання додатків на базі Spring [19]. Цей фреймворк пропонує інтуїтивно зрозумілі налаштування за замовчуванням та автоматизовану конфігурацію, що дозволяє швидко розпочати роботу над Spring-проектом без необхідності занурення у складні конфігураційні процеси. Spring Boot віддає перевагу конвенціям перед конфігурацією, пропонуючи такі можливості, як вбудовані сервери, ефективне управління залежностями та готові до виробництва метрики, що робить його відмінним вибором для розробки надійних та масштабованих корпоративних систем;

– SpringDoc OpenAPI, розроблена командою SpringDoc, є бібліотекою, що спрощує процес автоматизації створення документації API згідно зі специфікацією OpenAPI [20]. Ця бібліотека легко інтегрується з Spring Boot,

автоматично генеруючи документацію API безпосередньо з коду, що значно знижує необхідність ручного документування;

– Swagger Parser, розроблений командою SmartBear Software, є бібліотекою, призначеною для аналізу та обробки документів OpenAPI [21]. Ця бібліотека забезпечує можливість програмного читання та маніпулювання OpenAPI-файлами. Swagger Parser пропонує ряд корисних функцій, включаючи перевірку структури та синтаксису OpenAPI-файлів, екстракцію деталей про кінцеві точки, моделі та параметри, а також можливість генерації клієнтських SDK або серверних заглушок на основі визначень OpenAPI;

– SnakeYAML, розроблена Андрієм Сомовим, є Java-бібліотекою для роботи з YAML, яка часто застосовується у конфігураційних файлах [22]. Ця бібліотека забезпечує легкий у використанні API, який дозволяє розбирати YAML-файли на Java-об'єкти, а також серіалізувати Java-об'єкти назад у формат YAML;

– GeoTools, розроблена командою Open Source Geospatial Foundation, є Java-бібліотекою, призначеною для роботи з геопросторовими даними [23]. Ця бібліотека забезпечує різноманітні функції для читання, запису та обробки геопросторових форматів даних, таких як шейп-файли, GeoJSON та KML. GeoTools пропонує широкий спектр геопросторових операцій, включаючи виконання просторових запитів, трансформації координат та візуалізацію геоданих;

– Jackson, розроблена FasterXML, є високопродуктивною JSON-бібліотекою для Java [24]. Ця бібліотека забезпечує можливості серіалізації Java-об'єктів у формат JSON та десеріалізації JSON-даних назад у Java-об'єкти. Jackson підтримує різноманітні способи зв'язування даних JSON, що робить роботу з JSON більш гнучкою та ефективною.

Використовуючи ці потужні інструменти та бібліотеки, вдалось прискорити розробку бекенда, спростити управління конфігурацією, автоматизувати документування API, ефективно працювати з геопросторовими даними, безперешкодно обробляти серіалізацію/десеріалізацію JSON.

Система використовує платформу Geofabrik, розроблену компанією Geofabrik GmbH, зокрема карту України від OpenStreetMap (OSM) [26], для відображення необхідних об'єктів. Geofabrik відома як надійне джерело для отримання актуальних та деталізованих геопросторових даних. Однією з ключових переваг Geofabrik є регулярне оновлення даних, що забезпечує актуальність та точність інформації на карті. Дані з Geofabrik доступні у форматі Protocolbuffer Binary Format (PBF), який є одним з основних форматів для представлення даних в OSM.

Формат PBF (Protocolbuffer Binary Format), використовуваний у контексті OpenStreetMap (OSM), пропонує значні переваги порівняно з традиційним XML-форматом OSM, особливо у випадках роботи з великими наборами даних, такими як карта України. Однією з ключових переваг PBF є його високий ступінь стиснення даних без втрат, що робить файли значно меншими та швидшими для завантаження та зберігання. Це стає критично важливим у контексті великих геопросторових даних, де обсяги файлів можуть бути дуже великими.

Крім того, PBF оптимізований для швидкого читання та запису, що робить його ідеальним вибором для програмного забезпечення, яке вимагає високої продуктивності при обробці геоданих. Ця оптимізація є особливо важливою для додатків, які залежать від швидкого доступу до геопросторових даних, таких як системи географічної інформації (ГІС), навігаційні системи та інші додатки, що використовують картографічні дані.

Використання формату PBF у таких сценаріях не тільки підвищує ефективність завантаження та зберігання даних, але й сприяє більшій швидкості обробки та відображення карт, що є важливим для користувацького досвіду. Особливо це стає актуальним у випадках, коли потрібно швидко обробляти великі обсяги даних, наприклад, при візуалізації складних картографічних даних або при роботі з динамічними геопросторовими даними у реальному часі.

2.2 Проектування системи

Клієнт-серверна архітектура, що використовує REST API, є ключовим елементом у сучасній розробці веб-додатків. REST, що означає REpresentational State Transfer, є архітектурним стилем, який впроваджує принципи проектування для створення мережевих додатків. У цій архітектурі, взаємодія між клієнтом і сервером відбувається через протокол HTTP, використовуючи стандартні методи, такі як GET, POST, PUT, DELETE. Це дозволяє клієнтам відправляти запити та отримувати відповіді від сервера у стандартизованому форматі.

Однією з ключових характеристик REST є його безстановий характер. Кожен запит від клієнта містить всю необхідну інформацію для його обробки сервером, що означає, що сервер не зберігає стан клієнта між запитами. Це спрощує архітектуру сервера та підвищує масштабованість, оскільки сервер не потребує зберігати інформацію про стан для кожного клієнта.

Використання REST API в клієнт-серверних архітектурах також сприяє гнучкості та легкості інтеграції. Оскільки REST використовує стандартні HTTP протоколи, він може бути легко інтегрований з різними платформами та мовами програмування. Це робить REST популярним вибором для публічних API, які можуть бути використані різними клієнтами, включаючи веб-браузери, мобільні додатки та інші серверні додатки.

Крім того, REST API сприяє розробці модульних та розширюваних систем. Завдяки своїй стандартизованій структурі, REST дозволяє розробникам легко додавати нові функції та компоненти до системи без необхідності переписування існуючого коду. Це особливо важливо у великих та складних системах, де потреба у постійному розвитку та адаптації є критичною.

У підсумку, використання REST API у клієнт-серверних архітектурах є важливим для створення ефективних, масштабованих та гнучких веб-додатків. Цей підхід не тільки полегшує взаємодію між клієнтом та сервером, але й сприяє легкості розширення та інтеграції системи, що є ключовими факторами для успішної розробки сучасних веб-додатків.

Для зображення архітектури програмної системи існує безліч видів діаграм, в залежності від потреби зацікавлених сторін (стейкхолдерів).

Контейнерну діаграму системи зображено на рисунку 2.1.

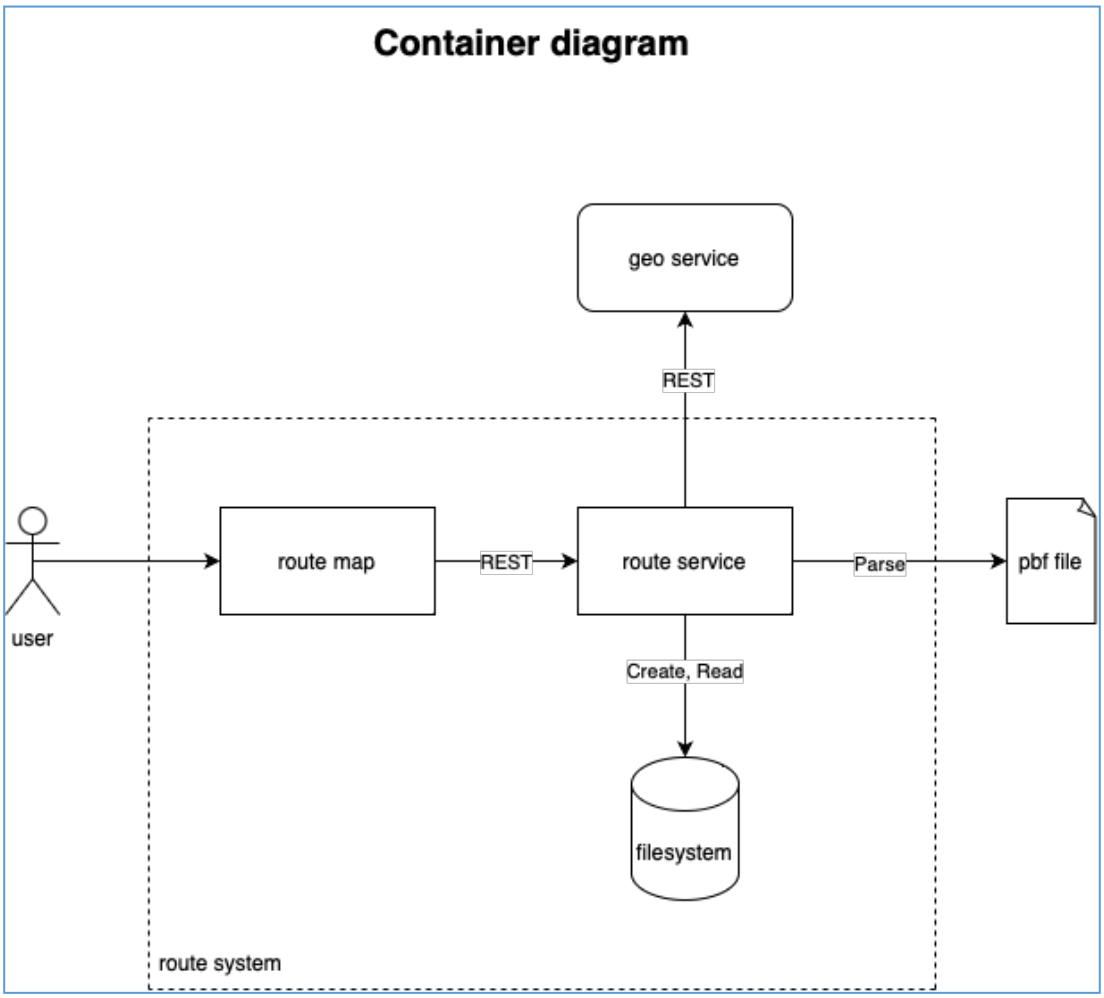

Рисунок 2.1 — Контейнерна діаграма архітектури системи

Система складається з декількох основних компонентів:

– Geo Service: Цей зовнішній компонент відповідає за надання географічної інформації, такої як переведення координат у назви об’єктів і навпаки (геокодування та зворотне геокодування). Взаємодія з Geo Service відбувається через REST API, що забезпечує стандартизований спосіб запиту і отримання даних. Цей сервіс є критично важливим для систем, які залежать від точної та актуальної географічної інформації, наприклад, для навігаційних

систем, логістичного планування, а також для додатків, що використовують геолокаційні дані для надання персоналізованих послуг. Геокодування дозволяє перетворювати адреси або інші географічні описи в географічні координати, що можуть бути використані для розміщення об'єктів на карті або для визначення маршрутів. Зворотне геокодування, у свою чергу, перетворює географічні координати назад у читабельні адреси або описи місць. Ці процеси є фундаментальними для багатьох сучасних геоінформаційних систем та мобільних додатків. Інтеграція з Geo Service через REST API надає системі гнучкість та масштабованість, оскільки дозволяє легко додавати або змінювати функціональність без необхідності переписування основного коду системи. Крім того, використання стандартизованого API спрощує взаємодію з різними географічними сервісами, що дозволяє системі вибирати найбільш підходящий сервіс в залежності від конкретних потреб та вимог.

– Route Service: Це серце системи маршрутизації, яке обробляє вхідні дані, зокрема PBF файли з OpenStreetMap. Воно використовує функціональність парсингу для читання і інтерпретації структурованих геоданих, а потім взаємодіє з файловою системою для збереження або витягу даних, які використовуються для маршрутизації;

– Route Map: Це клієнтський компонент, який взаємодіє з Route Service через REST API для відображення маршрутів. Він відправляє запити на отримання маршрутів та отримує дані, які можна відобразити на карті для кінцевих користувачів;

– Filesystem: Це сховище для геоданих, включаючи PBF файли, які містять інформацію, необхідну для маршрутизації. Route Service має можливість читати з цієї системи та записувати в неї.

Щодо Geo Service це важливий зовнішній компонент який дозволяє отримувати дані про об’єкт через надіслані координати, і навпаки отримувати координати через надіслані дані. Структурно взаємодія з цим компонентом відображена на контекстній діаграмі (див. рисунок 2.2.). На цій діаграмі

представлено високорівневу структуру системи маршрутизації та її взаємодію з зовнішніми сервісами.

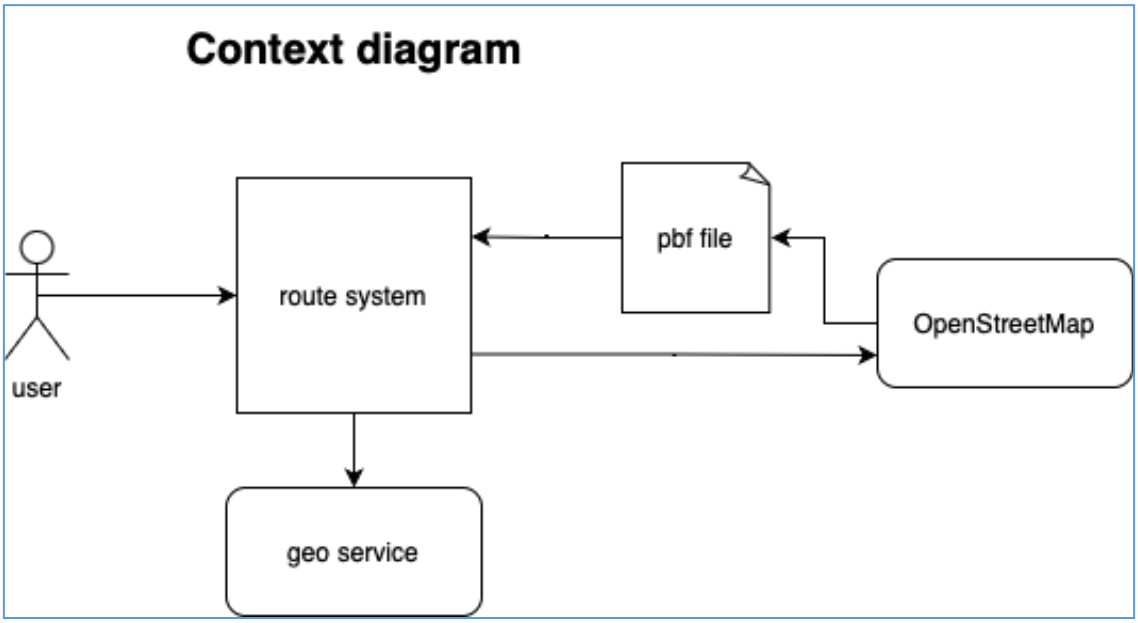

Рисунок 2.2 — Діаграма взаємодії системи з зовнішніми компонентами

На діаграмі представлено архітектуру системи маршрутизації, що включає наступні компоненти та їх взаємодію:

– Route System: Це ядро системи маршрутизації, відповідальне за прийом, обробку та відправку запитів на маршрутизацію. Воно використовує дані з файлів PBF для побудови маршрутів та взаємодіє з geo service для отримання додаткових геоданих.

– Geo Service: Це сторонній сервіс, який надає функції геокодування. Він взаємодіє з route system через REST API.

– OpenStreetMap: Це джерело вільних картографічних даних. Файли PBF з цього сервісу використовуються як основа для обчислення маршрутів у route system.

– PBF File: Це формат файлу, який містить стиснуті геодані з OpenStreetMap. Він використовується route system для побудови маршрутів.

Після обробки запитів та отримання даних, вони передаються іншим компонентам системи (див. рисунок 2.3), де присутні:

– Route Map: Це клієнтський інтерфейс для відображення маршрутів. Він відправляє запити на маршрутизацію до route service та отримує відповіді для користувача.

– File Parser: Компонент, який обробляє PBF файли, екстрагуючи з них необхідну інформацію для маршрутизації.

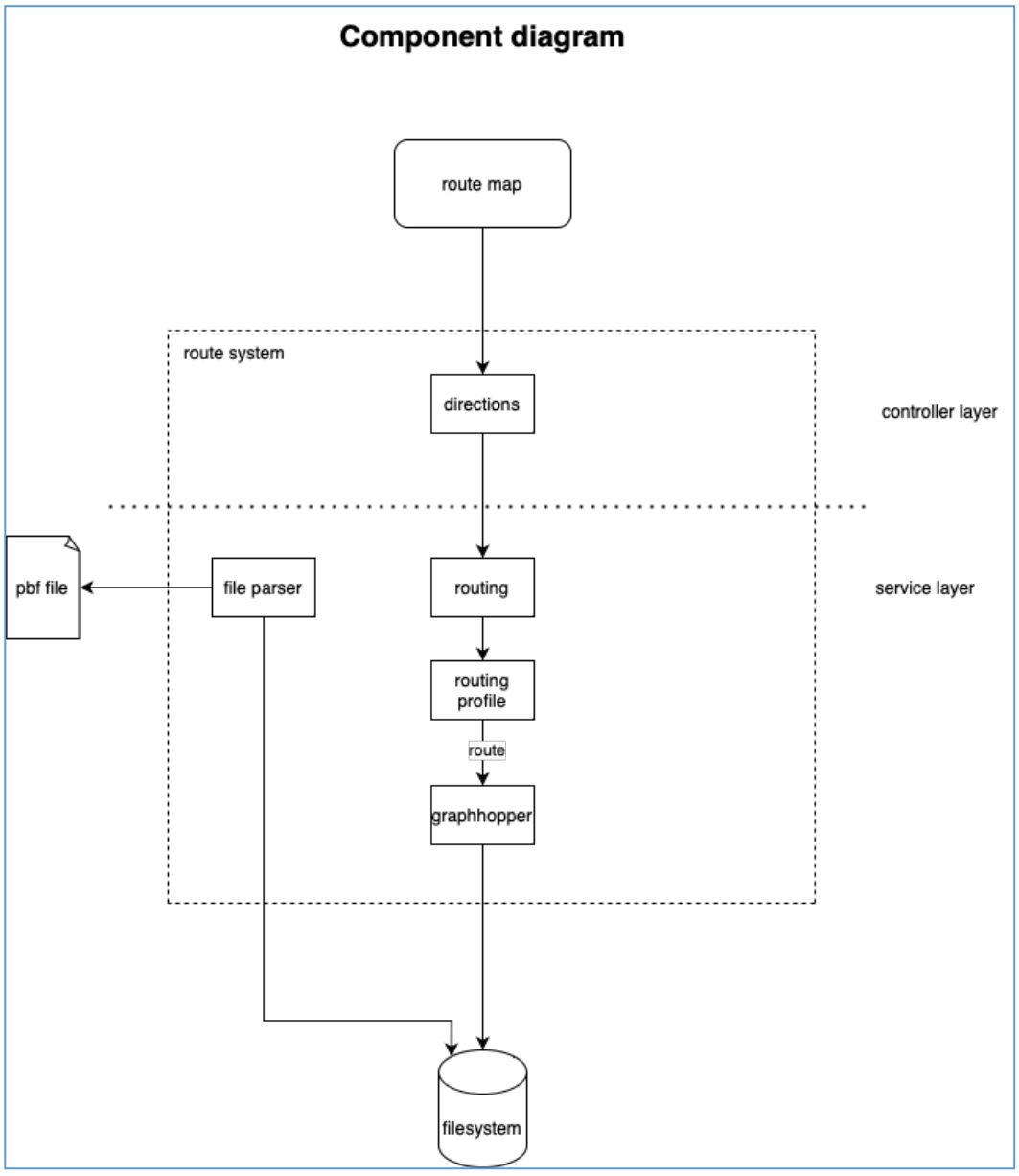

Рисунок 2.3 — Діаграма взаємодії внутрішніх компонентів

– Routing: Модуль, який відповідає за логіку маршрутизації, використовує дані з file parser для обчислення маршрутів;

– Routing Profile: Набір параметрів для маршрутизації, які визначають спосіб обчислення маршрутів (наприклад, найшвидший, найкоротший шлях);

– GraphHopper: Бібліотека для роботии з графами, які використовуються для обчислення оптимального маршруту на основі даних з file parser та routing profile;

– Filesystem: Сховище для збереження та доступу до геоданих, включаючи PBF файли та інші дані, які використовуються системою;

Розглянувши архітектуру, її основні компоненти та частково взаємодію між клієнтською та серверною стороною, можна зрозуміти, що система представляє собою добре інтегрований набір сервісів, орієнтований на обробку та аналіз геопросторових даних. Ключові елементи архітектури включають в себе взаємодію з відкритими геоданими OpenStreetMap, використання спеціалізованого сервісу для геокодування, і власний механізм маршрутизації для обробки та візуалізації маршрутів.

2.3 Клієнт-серверна взаємодія

Клієнт-серверний архітектурний патерн є одним з основних підходів у розробці мережевих та розподілених систем. Він передбачає розділення функціональності між постачальниками послуг (серверами) та їх споживачами (клієнтами), створюючи централізовану структуру, що сприяє ефективному управлінню ресурсами та обробці даних.

У цьому патерні, сервер відповідає за зберігання, обробку та управління даними, а також за надання різних сервісів та ресурсів. Клієнти, у свою чергу, звертаються до сервера для доступу до цих ресурсів, виконуючи запити на отримання даних або виконання певних операцій. Клієнтські додатки можуть бути розроблені на різних платформах та використовувати різноманітні інтерфейси, але вони всі взаємодіють з одним або декількома серверами через мережу.

Однією з ключових переваг клієнт-серверної архітектури є централізоване управління. Сервери можуть ефективно управляти ресурсами, забезпечувати безпеку, резервне копіювання даних та виконувати інші критично важливі задачі.

З іншого боку, клієнти можуть бути легкими та не вимагати значних обчислювальних ресурсів, оскільки основна частина обробки даних відбувається на сервері. Це робить клієнт-серверну архітектуру ідеальною для різних веб-додатків, мобільних застосунків та інших додатків, що вимагають широкого доступу через мережу.

Клієнт-серверна архітектура також сприяє масштабованості та гнучкості системи. Сервери можуть бути налаштовані для обслуговування великої кількості клієнтів, а при необхідності можуть бути додані додаткові сервери або ресурси для підтримки зростаючого навантаження.

Ця взаємодія зображеня на рисунку. 2.4.

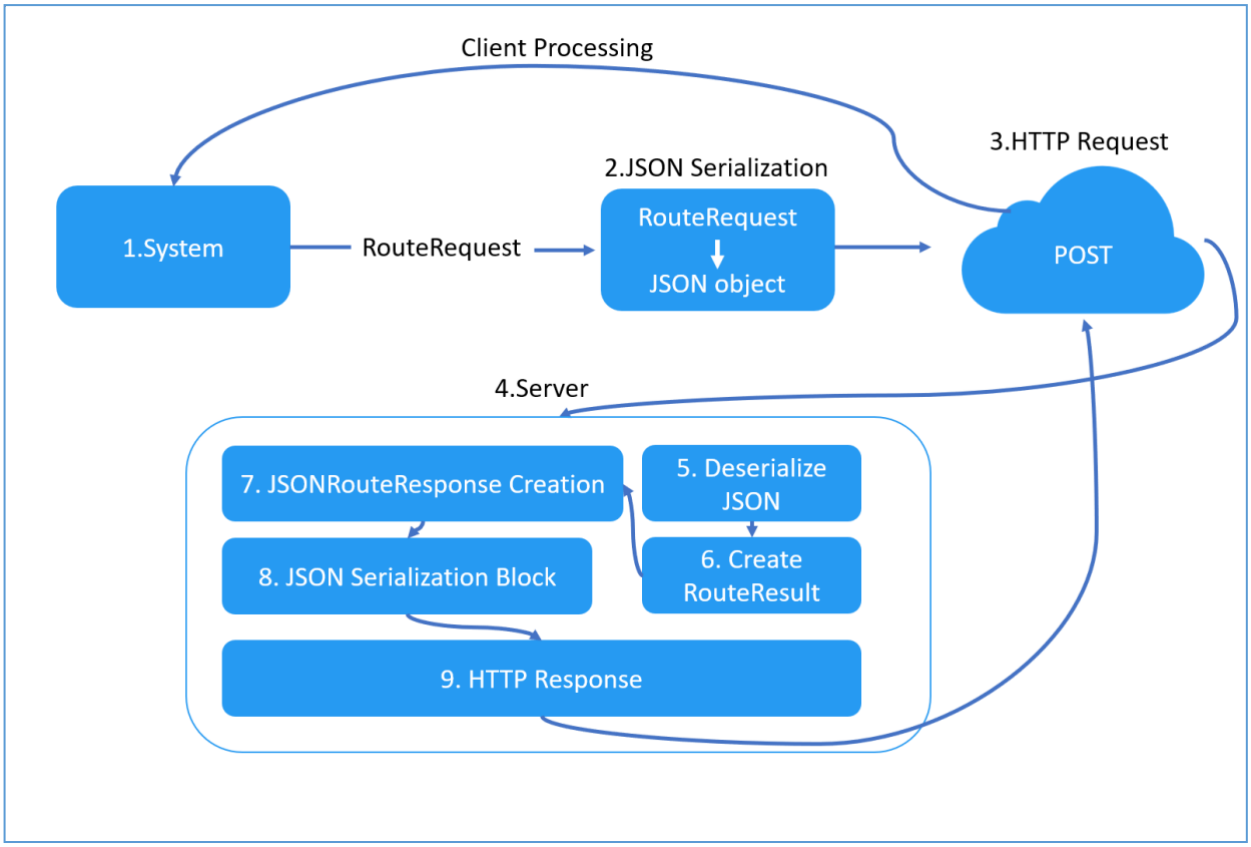

Рисунок 2.4 — Схема клієнт-сервер взаємодії в системі

Детальніше про кожен з кроків:

– Крок 1, користувач, яким може бути веб- або мобільний додаток, створює об’єкт RouteRequest. Цей об’єкт включає всі необхідні параметри, такі як координати, налаштування маршруту та будь-які додаткові опції для розрахунку маршруту;

– Крок 2, потім об’єкт RouteRequest серіалізується в JSON-рядок за допомогою фреймворку Jackson, який переводить властивості Java-об’єкта в JSON-формат. Цей серіалізований рядок JSON включається в тіло HTTP POST-запиту з правильними заголовками, що вказують на тип вмісту;

– Крок 3, сервер обладнаний REST кінцевою точкою, отримує HTTP POST запит. клас контролера зіставлений з кінцевою точкою маршруту, обробляє вхідний запит;

– Крок 4, тіло JSON з клієнтського HTTP-запиту десеріалізується назад в об’єкт RouteRequest на стороні сервера;

– Крок 5, сервер використовує цей об’єкт RouteRequest для обчислення маршруту. GraphHopper, який використовує параметри, вказані в RouteRequest, для обчислення маршруту;

– Крок 6, gісля обчислення маршрутів генерується масив об’єктів RouteResult, що містить деталі кожного обчисленого маршруту. Ці об’єкти RouteResult використовуються для створення об’єктів JSONIndividualRouteResponse, які містять детальну інформацію про маршрут, таку як шляхи та інструкції;

– Крок 7, новий об’єкт JSONRouteResponse створюється з масиву RouteResult і вихідного об’єкта RouteRequest. На цьому кроці сервер також об’єднує рамки з кожного об’єкта RouteResult в одну загальну рамку для всього маршруту за допомогою GeomUtility.generateBoundingFromMultiple;

– Крок 8, об’єкт JSONRouteResponse потім серіалізується в JSON і надсилається назад клієнту в тілі HTTP-відповіді;

– Крок 9, користувач отримує відповідь JSON і десеріалізує її в клієнтську версію об’єкта JSONRouteResponse або безпосередньо обробляє дані

JSON для оновлення інтерфейсу користувача, відображаючи карту маршруту та інші деталі для користувача.

Протягом усієї цієї взаємодії REST API забезпечує чистий протокол зв'язку без стану, який вимагає, щоб кожен запит був самодостатнім. Це означає, що серверу не потрібно підтримувати будь-який стан між запитами, що є перевагою для масштабованості та надійності. Використання JSON як середовища для запитів і відповідей забезпечує легкий і незалежний від мови обмін даними, який можна легко обробляти на обох кінцях системи.

2.4 Висновки

У цьому розділі були проаналізовані та визначенні структури та архітектура системи геопросторової маршрутизації. Це включало вибір інструментів для програмної реалізації, розробку архітектури та встановлення механізму взаємодії між клієнтом та сервером.

Технологічний стек, обраний для проекту, включає JavaScript для фронтенду та Java для бекенду, а також використання додаткових інструментів, таких як Vue.js, Leaflet, Vuetify, Axios, Spring Boot. Цей вибір забезпечує системі потрібну надійність та адаптивність. Використані інструменти сприяють покращенню інтерфейсу користувача та оптимізації обробки серверних запитів.

Архітектура системи побудована на основі клієнт-серверного підходу з використанням REST API, що забезпечує чітку координацію між компонентами, такими як Geo Service та Route Service. Така структура сприяє ефективній маршрутизації та точному обміну даними.

Обґрунтований вибір технологій підкреслює важливість ретельного планування та використання сучасних технологій та методик для створення ефективної, надійної та масштабованої системи геопросторової маршрутизації.

Інтеграція цих технологій створює міцну основу для подальшого розвитку та оптимізації системи, відкриваючи шлях для розширення функціональності та покращення користувацького досвіду.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Ключові ваги та їх вплив на побудову маршруту

Алгоритм Дейкстри, який застосовується у запропонованій програмній системі, є фундаментальним для обчислення найкоротших шляхів у графах, де ваги ребер відіграють ключову роль. Цей алгоритм ідеально підходить для врахування різноманітних факторів, таких як відстань, час, вартість або будь-які інші параметри, що можуть впливати на вибір оптимального маршруту. У контексті програмної системи, реалізація алгоритму Дейкстри включає декілька ключових класів, які відповідають за обчислення ваг ребер та управління пов'язаним з ними функціоналом.

Використання алгоритму Дейкстри у програмній системі дозволяє не тільки ефективно знаходити найкоротші шляхи, але й надає можливість оптимізувати маршрути з урахуванням специфічних потреб користувачів або вимог додатку. Це може бути особливо корисним у додатках для планування маршрутів, логістичних системах, а також у різних застосуваннях, де потрібно враховувати комплексні умови для визначення оптимального шляху.

Таким чином, інтеграція алгоритму Дейкстри у програмну систему забезпечує високу гнучкість та точність у визначенні маршрутів, що є важливим для ефективного та надійного планування шляхів у різних областях застосування.

Клас «AdditionWeighting» є цікавим прикладом використання композиції вагових функцій для формування більш складного і налаштованого маршруту. Центральна ідея AdditionWeighting полягає у сумуванні ваг від різних вагових функцій Weighting, що дозволяє одночасно враховувати кілька критеріїв при визначенні ваги краю. Метод «calcEdgeWeight», це перевизначений метод, який виконує основну функцію вагової функції, зображений на рисунку 3.1.

```
@Override
public double calcEdgeWeight(EdgeIteratorState edgeState, boolean reverse,
long edgeEnterTime) {
    double sumOfWeights = 0;
    for (Weighting weighting : weightings) {
        sumOfWeights += weighting.calcEdgeWeight(edgeState, reverse);
    }
    return superWeighting.calcEdgeWeight(edgeState, reverse,
edgeEnterTime) * sumOfWeights;
}
```

Рисунок 3.1 – Програмний код методу «calcEdgeWeight»

Діаграма класу «AdditionWeighting» зображена на рисунку 3.2

Рисунок 3.2 – Діаграма класу «AdditionWeighting»

Він визначає загальну вагу краю на основі поточного стану краю «EdgeIteratorState». Всередині цього методу, спочатку обчислюється сума ваг від усіх вагових функцій, що містяться у масиві weightings. Кожна вагова функція викликає «calcEdgeWeight» для даного краю, і результати сумуються. Після обчислення суми, це значення множиться на вагу краю, яка розрахована

основною ваговою функцією (superWeighting). Таким чином, кінцева вага краю визначається як комбінація декількох вагових функцій. Використання «AdditionWeighting» дає змогу формувати маршрути, які враховують декілька факторів одночасно, наприклад, швидкість, безпеку та екологічність. Це особливо корисно в складних маршрутизаційних сценаріях, де один критерій не може повністю визначати оптимальний маршрут.

Клас «AvoidHillsWeighting» спеціально розроблений для урахування крутизни пагорбів при плануванні маршруту. «AvoidHillsWeighting» наслідує «FastestWeighting», що означає, що він використовує швидкість як основний фактор, але з додатковими умовами для крутизни. Цей клас залежить від спеціального сховища даних, «HillIndexGraphStorage», яке зберігає інформацію про крутизну дороги.

«CalcEdgeWeight» це основний метод, який обчислює вагу краю з урахуванням крутизни, його вигляд можна побачити на рисунку 3.3.

```
@Override
public double calcEdgeWeight(EdgeIteratorState edgeState, boolean reverse)
{
      if (gsHillIndex != null) {
            boolean revert = edgeState.getBaseNode() <
            edgeState.getAdjNode();
            int hillIndex = gsHillIndex.getEdgeValue(edgeState.getEdge(),
            revert, buffer);

            if (maxSteepness > 0 && hillIndex > maxSteepness)
            return 100;

            return PENALTY_FACTOR[hillIndex];
      }
return 1;
}
```

Рисунок 3.3 – Програмний код методу «calcEdgeWeight»

Використовуючи «HillIndexGraphStorage», метод визначає індекс крутизни для даного краю. Індекс крутизни визначається на основі положення вузлів та кута нахилу дороги. Якщо крутизна перевищує заданий максимальний рівень (maxSteepness), метод повертає дуже високу вагу (у цьому випадку 100), щоб уникнути використання цієї дороги. В інших випадках застосовується штрафний фактор з масиву «PENALTY_FACTOR» залежно від крутизни. Використання

AvoidHillsWeighting дозволяє уникати крутих підйомів на маршруті, що робить маршрут більш комфортним для велосипедистів та пішоходів. Штрафні фактори ефективно збільшують вагу крутих ділянок, спонукаючи алгоритм маршрутизації обирати альтернативні, менш круті шляхи.

Клас «ConstantWeighting» в системі маршрутизації служить для надання однакової ваги кожному краю в мережі доріг. Цей підхід може бути корисним у певних спеціалізованих сценаріях, де характеристики шляху (як-от швидкість, крутизна чи тип дорожнього покриття) не мають значення. «ConstantWeighting» "визначає фіксовані значення для ваги та часу подорожі для кожного краю, незалежно від його фізичних характеристик.

Діаграма класу зображена на рисунку 3.4.

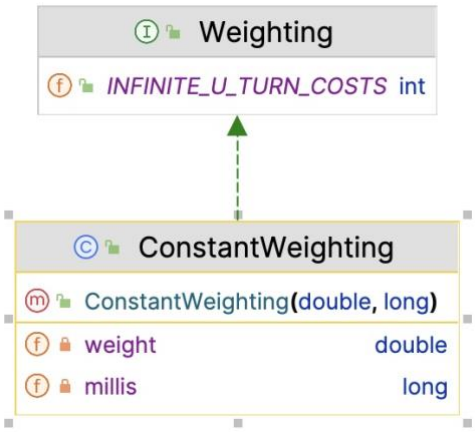

Рисунок 3.4 – Діаграма класу «ConstantWeighting»

Код класу зображено на рисунку 3.5.

```
@Override
public double calcEdgeWeight(EdgeIteratorState edgeIteratorState, boolean
reverse) {
    return weight;
}
@Override
public long calcEdgeMillis(EdgeIteratorState edgeIteratorState, boolean
reverse) {
    return millis;
}
```

Рисунок 3.5 – Програмний код класу «ConstantWeighting»

«CalcEdgeWeight» цей метод повертає постійну вагу (weight), задану для класу. Ця вага застосовується до кожного краю, незалежно від його властивостей. «calcEdgeMillis» аналогічно, цей метод повертає фіксований час подорожі (millis) для кожного краю. Оскільки всі краї мають однакову вагу, алгоритми маршрутизації використовують інші фактори, такі як відстань або рівність шляху, для визначення оптимального маршруту. «ConstantWeighting» може бути корисним у сценаріях, де важливіше досягти певної мети (наприклад, відвідати всі точки на карті) без урахування якості або характеристик шляхів.

Клас «FastestWeighting» (див. рисунок 3.6) розширює базовий клас «FastestWeighting» в системі маршрутизації, вносячи додаткові корективи до ваги маршрутів, зокрема шляхом врахування штрафів за напрямки, що не відповідають очікуванням користувача. «calcEdgeWeight» цей метод розраховує час, необхідний для проходження по краю, використовуючи швидкість, яка розраховується за допомогою «speedCalculator» (див. рисунок 3.7)

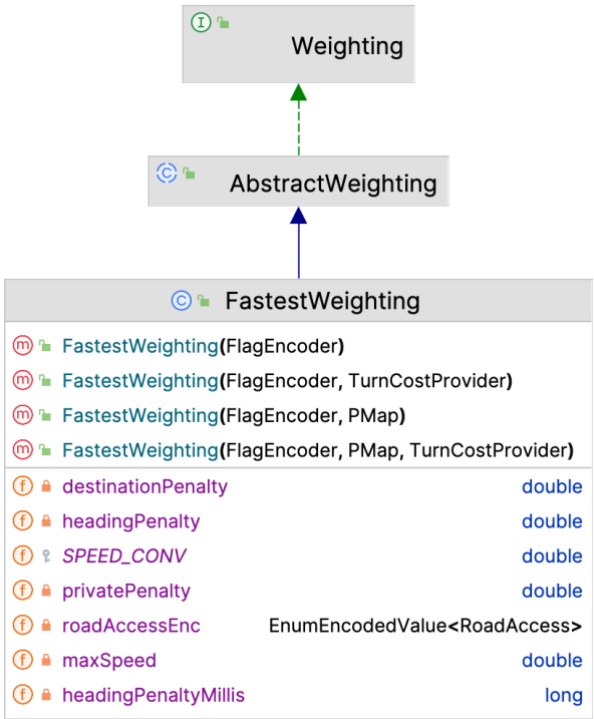

Рисунок 3.6 – Діаграма класу «FastestWeighting»

```
public double calcEdgeWeight(EdgeIteratorState edgeState, boolean reverse,
long edgeEnterTime) {
    double speed = speedCalculator.getSpeed(edgeState, reverse,
edgeEnterTime);
    if (speed == 0)
        return Double.POSITIVE_INFINITY;

    double time = edgeState.getDistance() / speed * SPEED_CONV;
    boolean unfavoredEdge =
edgeState.get(EdgeIteratorState.UNFAVORED_EDGE);
    if (unfavoredEdge)
        time += headingPenalty;
    return time;
}
```

Рисунок 3.7 – Програмний код методу «calcEdgeWeight» у класі «FastestWeighting»

Особливо актуально, коли деякі частини маршруту можуть бути визначені як unfavored, тобто небажані або не відповідають напрямку користувача. У таких випадках, система може використовувати механізм штрафів, наприклад, додавання додаткового часу (headingPenalty) до загального часу маршруту. Це дозволяє системі враховувати не тільки дистанцію та час, але й інші фактори, які можуть впливати на вибір маршруту користувачем.

Компонент «FastestWeighting» у такій системі відіграє ключову роль, оскільки він дозволяє створювати швидкі та ефективні маршрути, враховуючи різні параметри, включаючи індивідуальні уподобання користувачів та актуальні умови доріг. Це може включати, наприклад, уникнення ділянок з високим рівнем трафіку, поганими дорожніми умовами, або навіть маршрутів, які проходять через райони з підвищеною небезпекою.

Використання «FastestWeighting» дозволяє системі балансувати між швидкістю та комфортом маршруту, забезпечуючи користувачам оптимальний вибір. Це особливо важливо у програмах навігації та логістичних системах, де ефективність маршруту може мати значний вплив на загальний час поїздки та витрати на паливо.

Таким чином, інтеграція механізму штрафів та компоненту «FastestWeighting» у систему маршрутизації дозволяє створювати більш точні та адаптовані до потреб користувачів маршрути, забезпечуючи високий рівень задоволення користувачів та ефективності маршрутизації.

Клас «FastestSafeWeighting» призначений для створення швидких, але водночас безпечних маршрутів. Це досягається шляхом модифікації стандартної вагової функції «FastestWeighting» з урахуванням додаткових факторів безпеки. Метод «calcEdgeWeight», перш за все викликає «calcEdgeWeight» з базового класу «FastestWeighting» для отримання стандартної ваги краю. Його вигляд зображено на рисунку 3.8

```
@Override
public double calcEdgeWeight(EdgeIteratorState edgeState, boolean reverse,
long edgeEnterTime) {
    double weight = super.calcEdgeWeight(edgeState, reverse,
edgeEnterTime);
    if (Double.isInfinite(weight))
        return Double.POSITIVE_INFINITY;
    double priority =
getFlagEncoder().getDecimalEncodedValue(FlagEncoderKeys.PRIORITY_KEY).getD
ecimal(reverse, edgeState.getFlags());
    if (priority <= priorityThreshold)
        weight *= 2;
    return weight;
}
```

Рисунок 3.8 – Програмний код методу calcEdgeWeight

Якщо вага краю є нескінченною (що може вказувати на непрохідність), метод повертає Double.POSITIVE_INFINITY. Метод використовує декодер значень (DecimalEncodedValue) з FlagEncoder, щоб отримати пріоритет безпеки краю. Якщо пріоритет краю нижчий за заданий поріг (priorityThreshold), вага краю подвоюється. Це робить менш безпечні краї менш привабливими для маршрутизації. Використання «FastestSafeWeighting» дозволяє створювати маршрути, які є не лише швидкими, але й безпечними, особливо у міських умовах, де безпека може бути критичним фактором.

Клас «GreenWeighting» створений для того, щоб надавати перевагу зеленішим маршрутам, які проходять через більш зелені або природні зони. Це може включати парки, лісові ділянки або інші природні місця, які можуть бути більш приємними та екологічно чистими для проходження.

«HeatStressWeighting» цей клас використовується для зменшення впливу спеки на маршрут. Він використовує індекс теплового стресу та фактор ваги для обчислення загальної ваги краю.

«HgvAccessWeighting» клас використовується для вагового врахування доступності шляхів для важковагового транспорту (HGV). Якщо дорога недоступна для HGV, їй присвоюється нескінченно велика вага.

Клас «LimitedAccessWeighting» спроектований для врахування доріг із обмеженим доступом під час планування маршрутів. Він змінює вагу таких доріг, що дозволяє уникати або зменшувати використання доріг із обмеженим доступом, як-от приватних доріг або доріг із обмеженим призначенням. Метод modifyWeight: змінює вагу краю на основі типу доступу, використовуючи інформацію з EnumEncodedValue<RoadAccess>. Вигляд методу зображено на рисунку 3.9.

```
private double modifyWeight(double weight, EdgeIteratorState edgeState) {
    if (roadAccessEnc != null) {
        RoadAccess access = edgeState.get(roadAccessEnc);
        if (access == RoadAccess.DESTINATION)
            weight *= destinationPenalty;
        else if (access == RoadAccess.PRIVATE)
            weight *= privatePenalty;
    }
    return weight;
}
```

Рисунок 3.9 – Програмний код методу «modifyWeight»

Залежно від того, чи є дорога приватною (RoadAccess.PRIVATE) або із обмеженим призначенням (RoadAccess.DESTINATION), вага краю збільшується на відповідний штрафний фактор.

Метод «calcEdgeWeight» викликає основну вагову функцію, а потім змінює отриману вагу відповідно до обмежень доступу, цей метод зображено на рисунку 3.10.

```
@Override
public double calcEdgeWeight(EdgeIteratorState edgeState, boolean reverse,
long edgeEnterTime) {
    double weight = super.calcEdgeWeight(edgeState, reverse,
edgeEnterTime);
    return modifyWeight(weight, edgeState);
}
```

Рисунок 3.10 – Програмний код методу «calcEdgeWeight» класу «LimitedAccessWeighting»

Діаграма класу «LimitedAccessWeighting» зображена на рисунку 3.11.

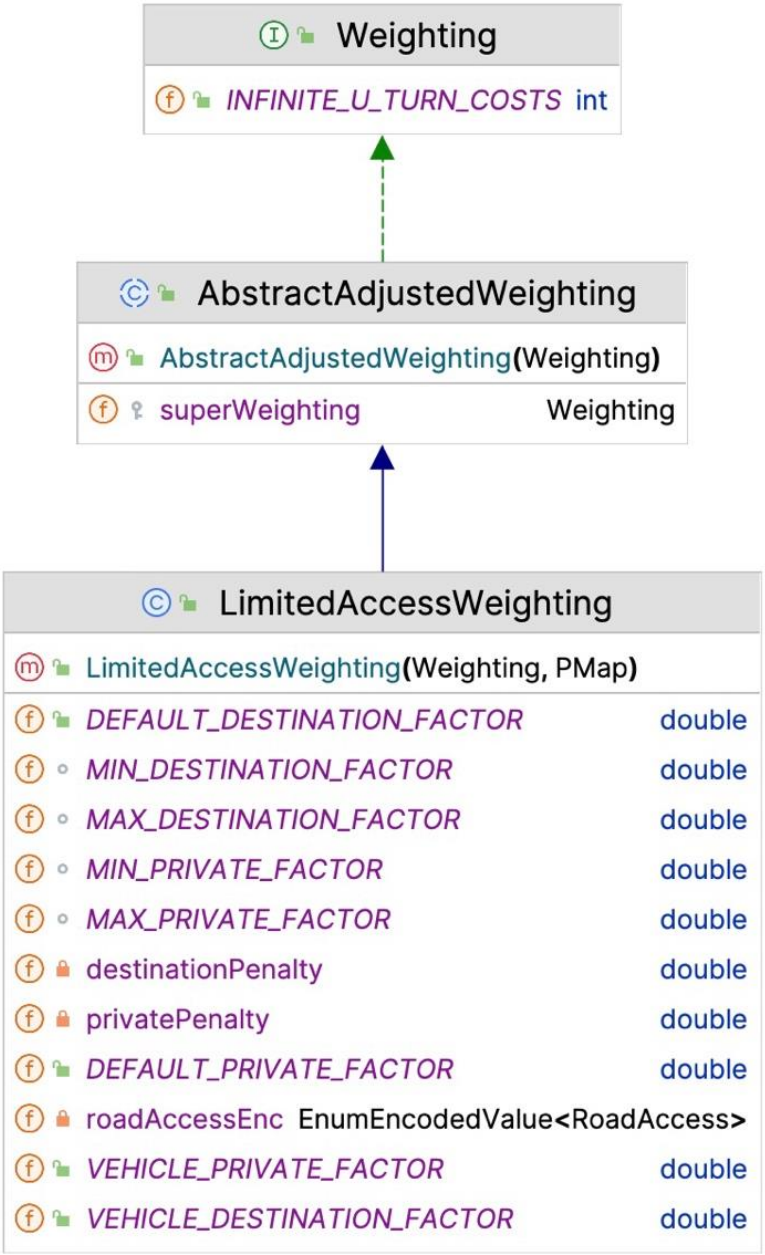

Рисунок 3.11 – Діаграма класу «LimitedAccessWeighting»

Використання «LimitedAccessWeighting» зменшує ймовірність включення доріг із обмеженим доступом у маршрути, що забезпечує більш реалістичний та практичний підхід до планування. Цей клас може бути корисним у різних сценаріях, особливо в міських умовах, де доступ до деяких доріг може бути обмеженим. «LimitedAccessWeighting» є важливим інструментом для створення більш точних і реалістичних маршрутів, що враховують обмеження доступу до доріг, сприяючи більш безпечному та зручному переміщенню у різноманітних середовищах.

Клас «PriorityWeighting» використовує пріоритети, задані для різних типів доріг, для зміни загальної ваги краю. Це дозволяє користувачам визначати пріоритети, наприклад, віддавати перевагу малозавантаженим або більш безпечним дорогам.

Клас «QuietWeighting» спрямований на створення більш тихих маршрутів, використовуючи індекс шуму для зміни загальної ваги краю.

«PreferencePriorityWeighting» схожий на «PriorityWeighting», але з більшим фокусом на персональних перевагах користувача. Цей клас може змінювати вагу шляхів залежно від індивідуальних переваг, таких як перевага певного типу поверхні.

Кожен із цих класів-вагів вносить свій вклад у процес вибору маршруту, дозволяючи користувачам адаптувати свої маршрути згідно з різними критеріями, такими як швидкість, безпека, комфорт або екологічність.

3.2 Реалізація алгоритму побудови маршруту

Алгоритм побудови маршруту є невід’ємною частиною сучасних систем навігації та маршрутизації. Він забезпечує ефективне визначення оптимального шляху між двома або більше точками у географічному просторі. Реалізація цього алгоритму повинна бути швидкою і ефективною, незалежно від розміру графа доріг та складності мережі. Вона повинна ураховувати різні фактори, такі як відстань, швидкість руху, наявність перешкод та інші обмежувальні умови.

Одним з найпопулярніших алгоритмів побудови маршруту є алгоритм Дейкстри, який базується на пошуку найкоротшого шляху в графі. Цей алгоритм забезпечує оптимальність маршруту та може бути застосований до різних видів транспорту і задач маршрутизації. Реалізація алгоритму Дейкстри вимагає ефективної структури даних та оптимізованого коду, щоб забезпечити швидке обчислення маршруту навіть у складних сценаріях. Системою передбачено наступні компоненти для реалізації алгоритму створення маршруту:

Конструктори:

– DijkstraManyToMany(RoutingCHGraph chGraph, Weighting weighting, TraversalMode tMode);

– DijkstraManyToMany(RoutingCHGraphchGraph,IntObjectMap<AveragedMultiTreeSPEntry>existingWeightMap,IntObjectMap<List<AveragedMultiTreeSPEntry>> existingCoreWeightMap, Weighting weighting,TraversalMode tMode);

– Функції та методи:

– initCollections(int size): Ініціалізує колекції для зберігання даних;

– reset(): Очищує дані алгоритму для нового запуску;

– prepare(int[] from, int[] coreExitPoints): Підготовка алгоритму, ініціалізує множину coreExitPoints;

– calcPaths(int[] from, int[] to): Основний метод для розрахунку шляхів;

– addEntriesFromMapToQueue(): Додає елементи з карти ваг до пріоритетної черги;

– runAlgo(): Виконує основний цикл алгоритму пошуку шляху;

– updateTarget(AveragedMultiTreeSPEntry update): Оновлює цільові вузли на основі знайдених шляхів;

– exploreEntry(RoutingCHEdgeIterator iter): Розглядає входи в поточний вузол;

– handleSingleEdgeCase(RoutingCHEdgeIterator iter): Обробляє випадки з одним ребром;

– handleMultiEdgeCase(RoutingCHEdgeIterator iter): Обробляє випадки з декількома ребрами;

– iterateMultiTree(RoutingCHEdgeIterator iter, AveragedMultiTreeSPEntry entry): Ітерує по дереву багатократних джерел;

– exploreEntryDownwards(RoutingCHEdgeIterator iter): Розглядає входи, йдучи вниз по графу;

– iterateMultiTreeDownwards(AveragedMultiTreeSPEntrycurrEdge,RoutingCHEdgeIterator iter, AveragedMultiTreeSPEntry adjEntry): Ітерує по дереву багатократних джерел у напрямку вниз;

– createEmptyEntry(RoutingCHEdgeIterator iter): Створює порожній запис для нового вузла;

– updateEntryInQueue(AveragedMultiTreeSPEntry entry, boolean isNewEntry): Оновлює запис у черзі;

– getEdgeEntry(RoutingCHEdgeIterator iter, List<AveragedMultiTreeSPEntry> entries): Отримує запис для певного ребра;

– createEntriesList(RoutingCHEdgeIterator iter): Створює список записів для вузлів;

– initBestWeightMapEntryList(IntObjectMap<List<AveragedMultiTreeSPEntry>> map, int traversalId): Ініціалізує список найкращих ваг для мапи;

– finishedDownwards(): Перевіряє, чи завершено пошук вниз по графу;

– setTargetGraphExplorer(RoutingCHEdgeIteratortargetGraphExplorer):

– Встановлює експлоратор для цільового графа;

– setTargetMap(IntObjectMap<AveragedMultiTreeSPEntry>targetMap):

– Встановлює карту цільових вузлів;

– setTargetSet(IntHashSet targetSet): Встановлює набір цільових вузлів;

– considerTurnRestrictions(int node): Враховує обмеження на повороти;

– setHasTurnWeighting(boolean hasTurnWeighting): Встановлює вагу поворотів;

– setTreeEntrySize(int entrySize): Встановлює розмір дерева записів;

– setSwap(boolean swap): Встановлює режим обміну (swap);

– getVisitedNodes(): Повертає кількість відвіданих вузлів;

– setVisitedNodes(int numberOfNodes): Встановлює кількість відвіданих вузлів;

– setMaxVisitedNodes(int numberOfNodes): Встановлює максимальну кількість відвіданих вузлів;

– getName(): Повертає назву алгоритму;

– calcPathWeight(RoutingCHEdgeIteratorState iter, SPTEntry currEdge, boolean reverse): Розраховує вагу шляху;

– calcWeight(RoutingCHEdgeIteratorState edgeState, boolean reverse, int prevOrNextEdgeId): Розраховує вагу для певного ребра;

– getTurnWeight(int edgeA, int viaNode, int edgeB, boolean reverse): Отримує вагу повороту;

– calcTime(RoutingCHEdgeIteratorState iter, SPTEntry currEdge, boolean reverse): Розраховує час проходження.

Проаналізуємо детальніше деякі ключові компоненти (функції, методи) які реалізовані в алгоритмі. Метод calcPaths(int[] from, int[] to) в алгоритмі є ключовим, оскільки він ініціює процес пошуку шляхів. Зображений на рисунку 3.12.

```
    public AveragedMultiTreeSPEntry[] calcPaths(int[] from, int[] to) {
        if (from == null || to == null)
            throw new IllegalArgumentException("Input points are null");
        prepare(from, to);
        addEntriesFromMapToQueue();
        outEdgeExplorer = swap ? chGraph.createInEdgeExplorer() :
chGraph.createOutEdgeExplorer();
        this.stoppingCriterion = new
MultiSourceStoppingCriterion(targetSet, targetMap, treeEntrySize);
        runAlgo();
        return new AveragedMultiTreeSPEntry[0];
    }
```

Рисунок 3.12 – Програмний код методу «calcPaths»

Метод приймає два параметри from: масив індексів вузлів, з яких починається пошук шляхів, to: масив індексів вузлів, до яких потрібно знайти шляхи. Перевірка вхідних даних: Спочатку метод перевіряє, чи вхідні масиви from та to не є null. Якщо хоча б один з них null, метод кидає виняток IllegalArgumentException, оскільки для визначення шляхів необхідні обидва

масиви.Використовуючи метод prepare(from, to), алгоритм ініціалізує необхідні структури даних. Це включає визначення точок виходу з ядра графа та ініціалізацію інших внутрішніх структур. Метод addEntriesFromMapToQueue() додає вузли з карти найкращих ваг (bestWeightMap) до пріоритетної черги. Це важливо для визначення початкового порядку розгляду вузлів. Встановлюється об'єкт «stoppingCriterion», який допомагає визначити, коли алгоритм повинен припинити пошук, заснований на заданих цільових вузлах. Викликається метод runAlgo(), який виконує основний цикл алгоритму. У цьому циклі алгоритм використовує пріоритетну чергу для визначення наступного вузла для розгляду та обробляє кожен вузол, враховуючи поточний найкращий знайдений шлях.

Після завершення пошуку, метод повертає масив об'єктів «AveragedMultiTreeSPEntry», який представляє знайдені шляхи. У цьому випадку повертається порожній масив, оскільки фактичні шляхи зберігаються в інших структурах даних алгоритму. Цей метод є вступним кроком для запуску складного процесу пошуку шляхів. Він інтегрує різні компоненти алгоритму для ефективного вирішення задачі маршрутизації багато-до-багатьох, враховуючи всі можливі шляхи від множини початкових точок до множини кінцевих точок.

Метод «runAlgo» є важливою частиною алгоритму, оскільки саме в ньому відбувається основний процес пошуку шляхів (див. рисунок 3.13). Спочатку метод вибирає поточне ребро (запис currEdge) з пріоритетної черги. Якщо черга порожня, алгоритм завершується:

Алгоритм використовує цикл while для ітерації через вузли, які потрібно розглянути. На кожному кроці перевіряється, чи перевищено максимальну кількість відвіданих вузлів. Якщо так, алгоритм завершується. Перевіряється, чи поточний вузол є частиною ядра графа (це робиться за допомогою функції isCoreNode). Якщо вузол є частиною ядра, алгоритм використовує «RoutingCHEdgeIterator» для ітерації по сусіднім ребрам і розглядає кожне з них за допомогою методу exploreEntry. Якщо вузол не є частиною ядра або є однією з точок виходу з ядра, використовується інший ітератор, targetGraphExplorer, для розгляду вниз по графу.

```
   protected void runAlgo() {
        RoutingCHEdgeExplorer explorer = swap ?
chGraph.createInEdgeExplorer() : chGraph.createOutEdgeExplorer();
        currEdge = prioQueue.poll();
        if (currEdge == null)
            return;
        while (!(isMaxVisitedNodesExceeded())) {
            int currNode = currEdge.getAdjNode();
            boolean isCoreNode = isCoreNode(chGraph, currNode, nodeCount,
coreNodeLevel);
            if (isCoreNode) {
                RoutingCHEdgeIterator iter =
explorer.setBaseNode(currNode);
                exploreEntry(iter);
            }
            updateTarget(currEdge);
            if (finishedDownwards() || prioQueue.isEmpty())
                break;
            currEdge = prioQueue.poll();
            if (currEdge == null)
                throw new AssertionError("Empty edge cannot happen");
        }
    }
```

Рисунок 3.13 – Програмний код методу «runAlgo»

Після розгляду сусідніх вузлів, метод «updateTarget» викликається для поточного ребра. Це дозволяє оновити інформацію про цільові вузли на основі нових даних. Викликається метод «finishedDownwards» для перевірки, чи алгоритм повинен завершити пошук. Якщо вся черга спустошена, алгоритм також завершується. Наступне ребро вибирається з пріоритетної черги для подальшого розгляду. Якщо наступне ребро відсутнє, алгоритм кидає виняток, оскільки це стан, який не повинен відбуватися.

Цей метод виконує важливу роль у визначенні та розгляді шляхів у графі. Він ітеративно розглядає вузли та ребра, оновлюючи інформацію про найкращі шляхи, використовуючи структуру даних пріоритетної черги. Це дозволяє алгоритму ефективно вибирати наступні вузли для розгляду та адаптуватися до змін у графі, забезпечуючи знаходження оптимальних шляхів відповідно до заданих критеріїв.

Функція exploreEntry(RoutingCHEdgeIterator iter) в алгоритмі відіграє важливу роль у дослідженні сусідніх вузлів для поточного вузла. «Iter», ітератор «RoutingCHEdgeIterator,» який використовується для перебору сусідніх ребер поточного вузла. Її вигляд зображено на рисунку 3.14

```
private void exploreEntry(RoutingCHEdgeIterator iter) {
    while (iter.next()) {
        if (considerTurnRestrictions(iter.getAdjNode())) {
            handleMultiEdgeCase(iter);
        } else {
            handleSingleEdgeCase(iter);
        }
    }
}
```

Рисунок 3.14 – Програмний код функції «exploreEntry»

Функція використовує «iter» для ітерації по всім сусіднім ребрам поточного вузла. Це включає перебір кожного ребра, що веде від або до поточного вузла. Для кожного ребра в циклі, функція виконує перевірку обмежень повороту (якщо вони застосовні) через метод «considerTurnRestrictions». Залежно від наявності обмежень повороту, функція або обробляє ребро як одиночне (через handleSingleEdgeCase), або як частину багатократного випадку (через handleMultiEdgeCase). У handleSingleEdgeCase, функція розглядає, чи вузол на іншому кінці ребра вже має запис у ваговій карті. Якщо немає запису, створюється новий, і виконується ітерація по дереву багатократних джерел (iterateMultiTree) для оцінки потенційного шляху.

У «handleMultiEdgeCase», функція розглядає випадки, де вузол на іншому кінці ребра вже має кілька записів у ваговій карті. Це може відбуватися, коли вузол є частиною багатьох потенційних шляхів, і кожен шлях потребує окремого розгляду.

Ця функція є ключовою для розуміння, як алгоритм досліджує та вибирає потенційні шляхи у графі. Вона забезпечує гнучкість алгоритму у розгляді різних варіантів шляхів, включаючи врахування обмежень повороту та обробку складних ситуацій з багатократними ребрами. Це дозволяє алгоритму точно визначати оптимальні шляхи, адаптуючись до структури графа та його обмежень.

Функція «updateTarget» (див. рисунок 3.15) відповідає за оновлення інформації про вузли, до яких вже знайдено шляхи, що дозволяє алгоритму оптимізувати вже існуючі шляхи:

```
    private void updateTarget(AveragedMultiTreeSPEntry update) {
        int nodeId = update.getAdjNode();
        if (targetSet.contains(nodeId)) {
            if (!targetMap.containsKey(nodeId)) {
                AveragedMultiTreeSPEntry newTarget = new
AveragedMultiTreeSPEntry(nodeId, EdgeIterator.NO_EDGE,
Double.POSITIVE_INFINITY, false, null, update.getSize());
                newTarget.setSubItemOriginalEdgeIds(EdgeIterator.NO_EDGE);
                targetMap.put(nodeId, newTarget);}
            AveragedMultiTreeSPEntry target = targetMap.get(nodeId);
            boolean updated = false;
            for (int i = 0; i < treeEntrySize; ++i) {
                MultiTreeSPEntryItem targetItem = target.getItem(i);
              double targetWeight = targetItem.getWeight();
                MultiTreeSPEntryItem msptSubItem = update.getItem(i);
                double updateWeight = msptSubItem.getWeight();
            }
            if (updated)
                stoppingCriterion.updateCombinedUnsettled();
        }}
```

Рисунок 3.15 – Програмний код функції «updateTarget»

У першому рядку функції отримується ідентифікатор вузла nodeId, який визначає цільовий вузол для оновлення. Далі, перевіряється, чи міститься цей цільовий вузол у множині «targetSet», яка містить всі цільові вузли, до яких вже знайдено шляхи. Якщо такий вузол вже є в множині, переходимо до оновлення інформації про нього. В першому кроці перевіряється, чи вже існує запис про цей цільовий вузол у «targetMap». Якщо немає, створюється новий об’єкт «newTarget» типу «AveragedMultiTreeSPEntry», який представляє інформацію про цільовий вузол. Цей об’єкт ініціалізується значеннями, які вказують на те, що шлях до цього вузла ще не знайдено. Потім отримується посилання на об’єкт target з targetMap, який представляє інформацію про цей цільовий вузол.

Далі виконується перебір по всіх елементах (i) списку «treeEntrySize», який містить піделементи (Item) для кожного знайденого шляху до цільового вузла. Для кожного із піделементів порівнюється його вага з вагою відповідного елемента у target. Якщо вага піделемента менша, то відбувається оновлення інформації про цей елемент у target. У разі, якщо були здійснені оновлення, виконується оновлення комбінованого списку невизначених вузлів шляху за допомогою методу «updateCombinedUnsettled()» об’єкту «stoppingCriterion».

Таким чином, функція updateTarget визначає процес оновлення інформації про цільовий вузол шляхом порівняння та заміни ваги та інших параметрів піделементів для знайдених шляхів. Це дозволяє оптимізувати вже існуючі шляхи та поліпшити ефективність алгоритму маршрутизації.

Функція «iterateMultiTree» виконує обробку кожного ребра в графі під час пошуку шляху алгоритмом. Ця функція є ключовою для вирішення, чи слід продовжувати рух по певному шляху. Вона зображена на рисунку 3.16.

```
    private boolean iterateMultiTree(RoutingCHEdgeIterator iter,
AveragedMultiTreeSPEntry entry) {
        boolean addToQueue = false;
        visitedNodes++;
        for (int source = 0; source < treeEntrySize; ++source) {
            MultiTreeSPEntryItem currEdgeItem =
this.currEdge.getItem(source);
            double entryWeight = currEdgeItem.getWeight();
            if (entryWeight == Double.POSITIVE_INFINITY ||
!currEdgeItem.isUpdate())
                continue;
            if (stoppingCriterion.isEntryLargerThanAllTargets(source,
entryWeight))
                continue;
            MultiTreeSPEntryItem msptSubItem = entry.getItem(source);
            if (!accept(iter, currEdgeItem.getIncEdge(), swap))
                continue;
            double edgeWeight = calcWeight(iter, swap,
currEdgeItem.getOriginalEdge());
            if (edgeWeight == Double.POSITIVE_INFINITY)
                continue;
            double tmpWeight = edgeWeight + entryWeight;
            if (stoppingCriterion.isEntryLargerThanAllTargets(source,
tmpWeight))
               continue;
        }
        return addToQueue;}
```

Рисунок 3.16 – Програмний код функції «iterateMultiTree»

Перший рядок функції встановлює змінну «addToQueue» в значення false. Ця змінна вказує, чи необхідно додати поточне ребро до черги для подальшого оброблення. Далі, збільшується лічильник відвіданих вузлів. У наступному циклі перебираються всі елементи (source) списку «treeEntrySize», який містить піделементи для кожного шляху у дереві найкоротших шляхів. Для кожного елемента отримується поточна вага (entryWeight). Якщо вага рівна Double.POSITIVE_INFINITY або піделемент не оновлювався, цикл переходить до наступної ітерації. Потім виконується перевірка за допомогою методу

stoppingCriterion.isEntryLargerThanAllTargets, яка перевіряє, чи є вага піделемента більшою за всі цільові значення для даного шляху. Якщо це так, цикл переходить до наступної ітерації. Далі отримується піделемент з об’єкта entry для поточного шляху. Викликається метод accept з параметрами iter, currEdgeItem.getIncEdge() та swap. Цей метод визначає, чи слід приймати ребро як частину шляху. Якщо результат методу є false, цикл переходить до наступної ітерації.

Обчислюється вага ребра за допомогою методу «calcWeight» з параметрами iter, swap та currEdgeItem.getOriginalEdge(). Якщо вага рівна Double.POSITIVE_INFINITY, цикл переходить до наступної ітерації. Обчислюється тимчасова вага tmpWeight, яка є сумою ваги ребра та ваги поточного шляху. Проводиться перевірка за допомогою методу stoppingCriterion.isEntryLargerThanAllTargets, чи є тимчасова вага більшою за всі цільові значення для даного шляху. Якщо це так, цикл переходить до наступної ітерації. Якщо вага піделемента «msptSubItem» більша за тимчасову вагу, оновлюються значення піделемента з вагою, ребром, оригінальним ребром, вхідним ребром, батьком та позначкою оновлення. Змінна «addToQueue» встановлюється в true, оскільки поточне ребро слід додати до черги для подальшого оброблення. Після чого «addToQueue» повертається.

3.3 Формування та заповнення мапи

У системі маршрутизації процес формування та наповнення карти починається з моменту, коли сервер отримує запит на маршрут від клієнта. Цей процес включає в себе збір, обробку та відображення маршрутних даних, що вимагає точності та зручності для користувача. Збір даних може включати геопросторові дані, інформацію про дорожні умови, трафік, обмеження швидкості та інші важливі аспекти. Ці дані можуть надходити з різних джерел, включаючи відкриті бази даних, сенсори, GPS-трекери та користувацькі відгуки.

Після збору даних відбувається їх обробка, де дані перетворюються у формат, придатний для використання системою маршрутизації. Це може

включати геокодування, стандартизацію даних, видалення помилок та невідповідностей, а також інтеграцію даних з різних джерел у єдину базу даних. Останнім кроком є відображення маршрутних даних на карті, що включає візуалізацію маршрутів, дорожніх знаків, трафіку та інших важливих елементів. Відображення має бути зрозумілим та інтуїтивно зручним для користувача, забезпечуючи легкий доступ до інформації про маршрут.

Коли сервер отримує запит на маршрут від клієнта, система використовує зібрані та оброблені дані для розрахунку оптимального маршруту. Цей маршрут потім відображається на карті та надсилається користувачу. Важливо, щоб система була здатна швидко реагувати на запити та адаптуватися до змін у дорожніх умовах, щоб забезпечити актуальність та точність маршрутних рекомендацій.

Таким чином, ефективна система маршрутизації залежить від точного збору, обробки та відображення маршрутних даних, а також від швидкої та точної взаємодії з користувачем. Це забезпечує користувачам надійні та зручні інструменти для планування та навігації, що є ключовим для успішної реалізації програмного забезпечення для маршрутизації.

Цей запит інкапсулюється в об'єкт «RouteRequest»(див. рисунок 3.17), який містить основні деталі, такі як:

– «coordinates» (List<List<Double>>): Це поле представляє список координат (широта, довгота) для маршруту. Кожна координата є парою парних чисел, необхідних для визначення початкової, кінцевої та проміжних точок маршруту;

– «routePreference» (APIEnums.RoutePreference): Визначає перевагу для маршруту, наприклад, найшвидший, найкоротший тощо. Цей параметр впливає на те, як розраховується та оптимізується маршрут;

– «units» (APIEnums.Units): Визначає систему одиниць для маршруту, наприклад, кілометри або милі. Це важливо для представлення відстані та швидкості у вибраних користувачем одиницях;

– «alternativeRoutes» (RouteRequestAlternativeRoutes): Обробляє запити на альтернативні маршрути, надаючи користувачам декілька варіантів маршрутів;

– «maximumSpeed» (double): Встановлює максимальну швидкість, яку слід враховувати при розрахунку маршруту, що впливає на розрахунковий час у дорозі.

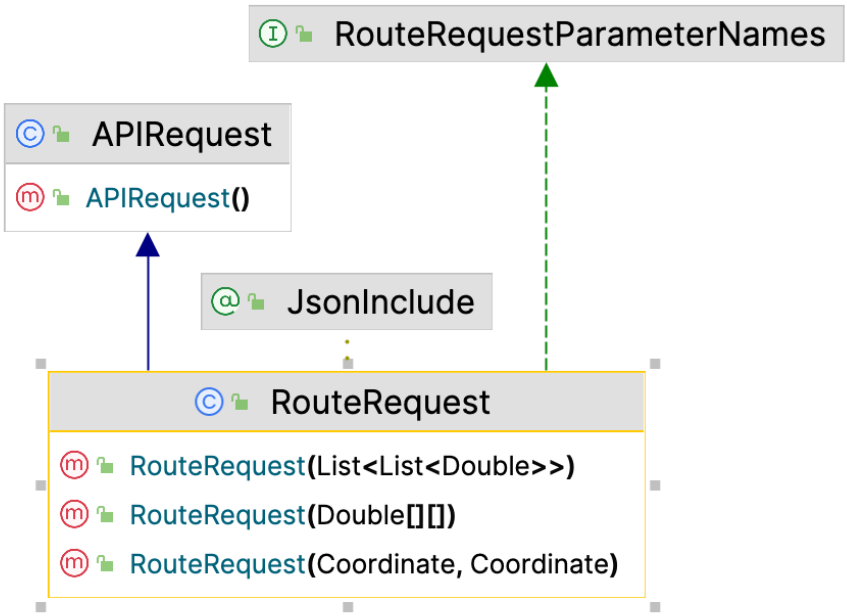

Рисунок 3.17 – Діаграма класу «RouteRequest»

Після отримання запиту на маршрут, «RoutingAPI» сервера активно приступає до його обробки. Цей процес включає в себе визначення профілю маршрутизації, що враховує такі фактори, як умови дороги та інші важливі параметри, а також встановлення формату, у якому буде надана відповідь. В цьому контексті, метод «getJsonRoute» в «RoutingAPI» відіграє ключову роль, оскільки він відповідає за генерацію та форматування відповіді.

Основна суть системи полягає в її здатності точно генерувати маршрути, виходячи з запитів клієнтів.

Завдяки функції routingService.generateRouteFromRequest(request), система здійснює розрахунок маршруту, ураховуючи всі задані користувачем параметри,

такі як пункт відправлення, пункт призначення, бажаний час подорожі, та інші специфічні вимоги.

Після того, як маршрут успішно згенеровано, система формує відповідь у форматі «JSONRouteResponse». Ця відповідь містить усі необхідні дані для візуалізації маршруту на карті, включаючи точки повороту, відстань, оцінений час подорожі та інші важливі інструкції. Ця інформація є критично важливою для користувача, оскільки дозволяє з легкістю візуалізувати та слідувати маршруту.

Таким чином, система маршрутизації забезпечує не тільки точне прокладання маршрутів, але й зручний інтерфейс для користувачів, що дозволяє їм ефективно планувати свої подорожі. Це досягається завдяки використанню передових технологій та алгоритмів, які забезпечують високу точність та надійність системи.

Деталі кожного сегмента маршруту зберігаються в класі IndividualRouteResponse, його зображено на рисунку 3.17

```
public class IndividualRouteResponse {
    protected Coordinate[] routeCoordinates;
    protected boolean includeElevation = false;
    protected boolean isPtRequest = false;
    public IndividualRouteResponse(RouteResult result, RouteRequest
request) {
        if (result.getGeometry() != null)
            this.routeCoordinates = result.getGeometry();
        if (request.hasUseElevation())
            includeElevation = request.getUseElevation();
        isPtRequest = request.isPtRequest();
    }
}
```

Рисунок 3.17 – Програмний код класу «IndividualRouteResponse»

Діаграму класу «IndividualRouteResponse» зображено на рисунку 3.18

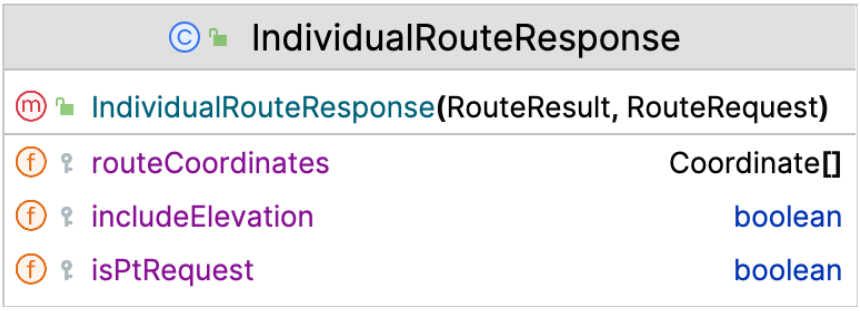

Рисунок 3.18 – Діаграма класу IndividualRouteResponse

routeCoordinates (Coordinate[]) масив об'єктів типу «Coordinate», що представляють географічні точки, які формують маршрут. Ці координати необхідні для відображення маршруту на карті. includeElevation (boolean) прапорець, який вказує, чи слід включати дані про висоту у відповідь про маршрут. Це може бути важливо для маршрутів, що пролягають через різноманітний рельєф місцевості. isPtRequest (boolean) булеве значення, яке вказує, чи запит стосується громадського транспорту. Це впливає на інтерпретацію та представлення маршруту.

Конструктор IndividualRouteResponse(RouteResult result, RouteRequest request). Якщо об'єкт результату (екземпляр RouteResult) містить дані геометрії (перевіряється за допомогою result.getGeometry() != null), ця геометрія присвоюється полю routeCoordinates. Ця геометрія по суті представляє фізичний шлях маршруту. Таким чином, клас «IndividualRouteResponse» має вирішальне значення для обробки та інкапсуляції детальної інформації про кожен сегмент маршруту, включаючи його координати. Цей клас гарантує, що система маршрутизації надасть вичерпні та адаптовані відповіді на кожен запит про маршрут.

3.4 Висновки

У цьому розділі магістерської кваліфікаційної роботи представлено глибокий аналіз програмної реалізації системи маршрутизації, фокусуючись на ключових аспектах алгоритму Дейкстри та його впровадженні у систему.

Описуються різноманітні вагові класи, що використовуються в системі для модифікації маршрутів. Це включає в себе врахування таких факторів, як крутизна доріг, максимальна дозволена швидкість, обмеження доступу та інші, що впливають на визначення ваг ребер. Ці ваги вносять суттєві корективи у процес планування маршрутів, спрямовуючи систему на вибір найбільш оптимальних і безпечних шляхів.

Основна увага приділяється реалізації алгоритму побудови маршруту. Обговорюються ключові методи та їх роль у ефективному визначенні маршрутів у різноманітних географічних і дорожніх умовах. Наголошується на здатності алгоритму адаптуватися до складних умов маршрутизації, забезпечуючи високу точність та ефективність обчислень.

Окремий акцент робиться на процесі формування та заповнення карти. Відображається повний цикл обробки запитів на маршрут – від ініціації запиту користувачем до фінального відображення маршруту на карті. Підкреслюється роль системи у точному відображенні географічних даних та адаптації маршрутів до індивідуальних уподобань користувачів.

Загалом, цей розділ демонструє глибину та складність програмної реалізації системи маршрутизації, відображаючи здатність системи забезпечувати ефективні, безпечні та налаштовані маршрути на основі різноманітних параметрів і обмежень.

4 ТЕСТУВАННЯ СИСТЕМИ

4.1 Вибір варіантів тестування

Існують різні способи тестування програмного забезпечення, кожен з яких має свій унікальний підхід та переваги. Модульне тестування передбачає тестування окремих модулів або компонентів програмного забезпечення ізольовано. Воно зосереджене на перевірці функціональності невеликих, незалежних частин коду. Юніт-тести зазвичай пишуться розробниками і часто виконуються в процесі розробки. Цей тип тестування допомагає виявити помилки і переконатися, що кожен компонент програмного забезпечення функціонує правильно [28]. Інтеграційне тестування спрямоване на перевірку взаємодії між різними компонентами або модулями програмного забезпечення. Воно гарантує, що ці компоненти безперебійно працюють разом і що дані між ними правильно передаються. Інтеграційне тестування необхідне для виявлення проблем, які можуть виникнути при інтеграції декількох частин програмної системи.

Системне тестування оцінює всю програмну систему в цілому. Воно перевіряє, чи всі компоненти програмного забезпечення правильно функціонують разом і відповідають заданим вимогам. Системне тестування може включати як функціональне, так і нефункціональне тестування для оцінки загальної продуктивності, надійності та зручності використання програмного забезпечення. Приймальне тестування проводиться для того, щоб визначити, чи відповідає програмне забезпечення вимогам та очікуванням замовника. Зазвичай воно виконується кінцевими користувачами або зацікавленими сторонами і підтверджує, що програмне забезпечення готове до розгортання. Приймальне тестування гарантує, що програмне забезпечення відповідає своєму призначенню і відповідає потребам користувачів [30]. Тестування продуктивності фокусується на оцінці швидкості, швидкості реагування, масштабованості та стабільності програмного забезпечення за різних умов навантаження. Воно вимірює показники продуктивності системи, такі як час

відгуку, пропускна здатність і використання ресурсів, щоб виявити будь-які вузькі місця або проблеми з продуктивністю.

Тестування безпеки призначене для виявлення вразливостей і потенційних ризиків безпеки в програмному забезпеченні. Воно допомагає переконатися, що конфіденційні дані захищені, контроль доступу реалізований правильно, а програмне забезпечення стійке до загроз безпеки. Тестування безпеки може включати такі методи, як тестування на проникнення, сканування вразливостей та аналіз коду. Тестування чорного ящика фокусується на зовнішній поведінці програмного забезпечення без урахування його внутрішньої структури або деталей реалізації. Тестувальники проводять тестування чорного ящика, імітуючи реальні сценарії та вхідні дані, щоб спостерігати за реакцією системи. Цей підхід гарантує, що програмне забезпечення функціонує правильно з точки зору користувача, і допомагає виявити потенційні функціональні дефекти.

Тестування білого ящика заглиблюється у внутрішню структуру, код та архітектуру програмного забезпечення. Тестувальники з доступом до внутрішніх компонентів системи можуть ретельно вивчити логіку коду, алгоритми та деталі реалізації. Такий підхід дозволяє виявити помилки кодування, вузькі місця в продуктивності, вразливості безпеки та будь-які інші проблеми, які можуть виникнути через внутрішню роботу програмного забезпечення.

Кожна методологія тестування відіграє важливу роль у забезпеченні якості та надійності програмного забезпечення. У той час як модульне та інтеграційне тестування фокусуються на окремих компонентах, системне та приймальне тестування перевіряють загальну функціональність та відповідність вимогам користувача. Тестування продуктивності забезпечує оптимальну роботу системи, а тестування безпеки підвищує стійкість програмного забезпечення до потенційних загроз. Тестування чорного ящика перевіряє зовнішню поведінку, тоді як тестування білого ящика досліджує внутрішню структуру та якість коду[31]. Важливо вибрати і поєднати відповідні методології тестування, виходячи з конкретних потреб і цілей програмного проекту. Для тестування

розробленої системи було обрано системне програмування, яке включає в себе відповідність системи поставленим очікуванням та вимогам.

4.2 Тестування розробленої системи

Тест-кейси відіграють критичну роль у процесі тестування програмного забезпечення, оскільки вони дозволяють систематично перевіряти функціональність та відповідність програми встановленим вимогам. Кожен тест-кейс зазвичай містить набір кроків, умов виконання, вхідних даних, очікуваних результатів та фактичних результатів. Це дозволяє тестувальникам точно визначити, чи працює програмне забезпечення так, як передбачалося, та ідентифікувати будь-які відхилення від заданих специфікацій.

Розробка ефективних тест-кейсів вимагає глибокого розуміння функціональних та нефункціональних вимог програмного забезпечення. Тест-кейси повинні бути достатньо детальними, щоб покривати різні сценарії використання, включаючи крайові випадки та помилкові умови. Це допомагає забезпечити, що програмне забезпечення є надійним та стійким до помилок.

Крім того, тест-кейси важливі для процесу регресійного тестування, де вони використовуються для перевірки, що нові зміни або оновлення не порушили існуючу функціональність. Регулярне виконання тест-кейсів під час розробки допомагає виявити та виправити помилки на ранніх стадіях, що може значно знизити вартість та складність їх виправлення у майбутньому.

Вони допомагають командам розробників та тестувальників забезпечити, що програмне забезпечення відповідає всім визначеним стандартам якості та вимогам користувачів. Ефективно сплановані та виконані тест-кейси можуть значно підвищити впевненість у якості та надійності програмного продукту перед його випуском.

У підсумку, тест-кейси є невід'ємною частиною процесу тестування, яка допомагає забезпечити, що програмне забезпечення функціонує належним чином, відповідає вимогам та готове до використання кінцевими користувачами.

Приклад тест кейсів наведено у таблиці 4.1

Таблиця 4.1 – Приклад тест кейсів

№	**Функція**	**Дія користувача**	**Очікувана реакція системи**
1	Доступ до системи	Відкрити веб-браузер і ввести URL-адресу системи	Система завантажується, відображається інтерфейс карти та планування маршруту
2	Відображення базової карти	Не потрібно дій	На карті відображаються міста, містечка та регіони з чіткими назвами
3	Навігація та масштабування	Використання елементів керування масштабом	Карта збільшується або зменшується відповідно
4	Пошук місця	Ввести назву місця або координати	Карта центрується на вказаному місці
5	Пошук та вибір об’єктів	Вибір місця на карті	Відображається вікно з інформацією про місце
6	Планування маршруту	Вибір двох точок на карті	Побудова маршруту з відображенням деталей
7	Додаткові параметри маршруту	Вибір між рекомендованим та найкоротшим маршрутом	Оновлення маршруту з новими даними

Продовження таблиці 4.1 – Приклад тест кейсів

№	Функція	Дія користувача	Очікувана реакція системи
8	Уникнення об’єктів	Вибір об’єктів для уникнення	Врахування обраних об’єктів у побудові маршруту
9	Налаштування швидкості	Налаштування максимальної та мінімальної швидкості	Оновлення маршруту згідно з параметрами швидкості
10	Альтернативні маршрути	Створення альтернативних маршрутів з налаштуваннями	Відображення кількох альтернативних маршрутів

Проведемо системне тестування на відповідність очікуваного результату поводження системи на дії користувача. Коли користувач зайде на посилання системи, повинна відображатись мапа (див. рисунок 4.1), де чітко та візуально зрозуміло мають з’явитись назви ключових об’єктів на ній. Тестування цього етапу можна вважати пройденим, адже очікуваний результат відповідає дійсності.

Це підтверджує, що система здатна надавати користувачам точну та актуальну інформацію, що є критично важливим для забезпечення ефективної навігації та планування маршрутів. Візуальна чіткість та точність географічних даних на мапі є ключовими факторами для забезпечення зручності користувачів та їхньої довіри до системи.

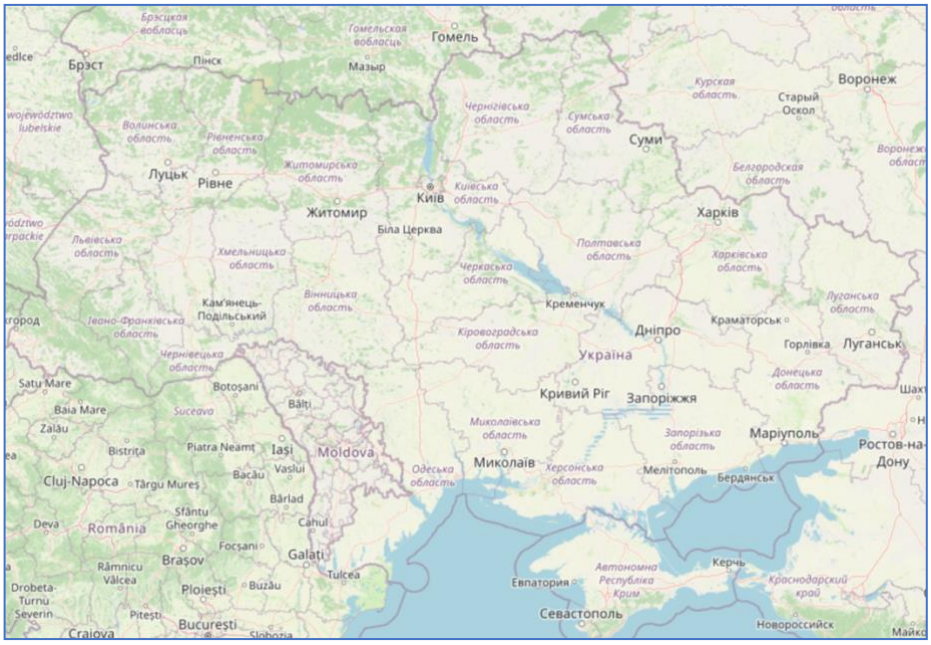

Рисунок 4.1 — Відображення мапи

Також на мапі крім об’єктів повинен відображатись функціонал (див. рисунок 4.2) який відповідає за збільшення\зменшення масштабу карти та введення необхідного місця або координат.

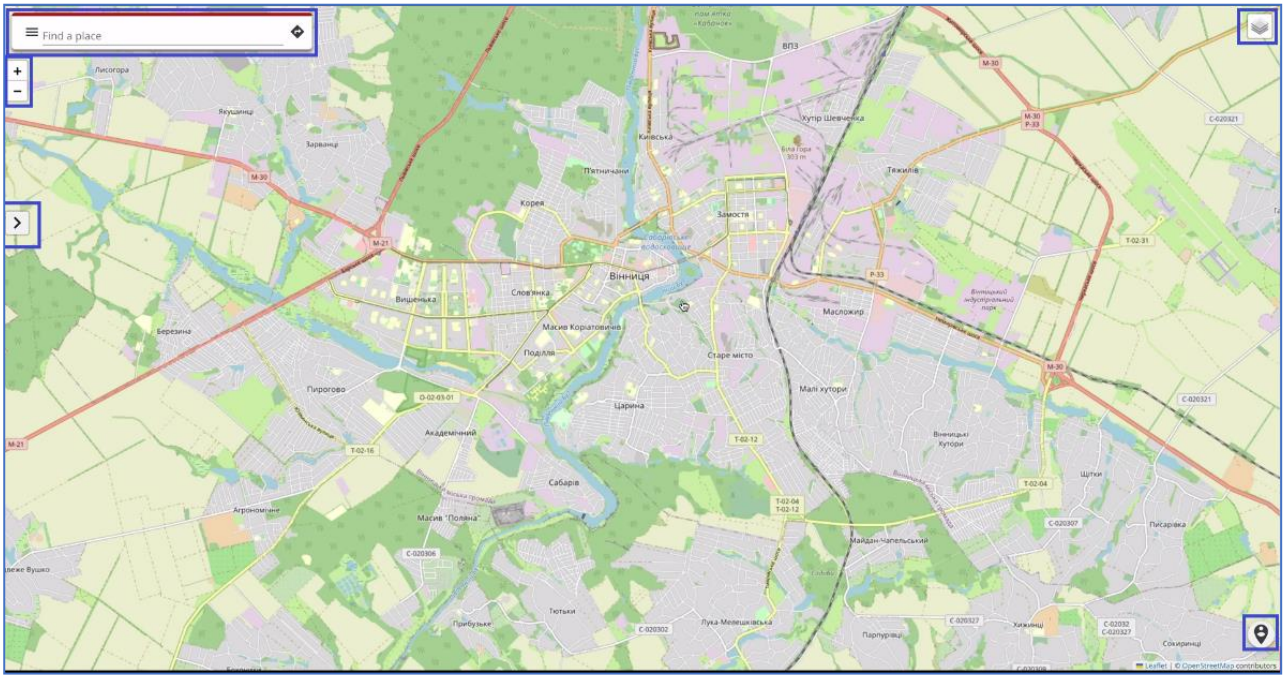

Рисунок 4.2 — Інтерфейс системи

На зображенні синім кольором виділені основні для цього етапу функціональні можливості системи. Поле для вводу координат або назви місця, зміна шарів відображення, кнопки зміни масштабу, поточне місце користувача, та відкриття повної форми. Все відображається правильно у заздалегідь визначених місцях, збільшення та зменшення масштабу також відповідає очікуваному результату. З чого можна зробити висновок про успішне проходження цього етапу тестування.

Під час того як користувач буде вводити щось у поле для координат або назви об’єкту, система повинна пропонувати йому ці місця з контекстного списку. Контекстний список зображено на рисунку 4.3.

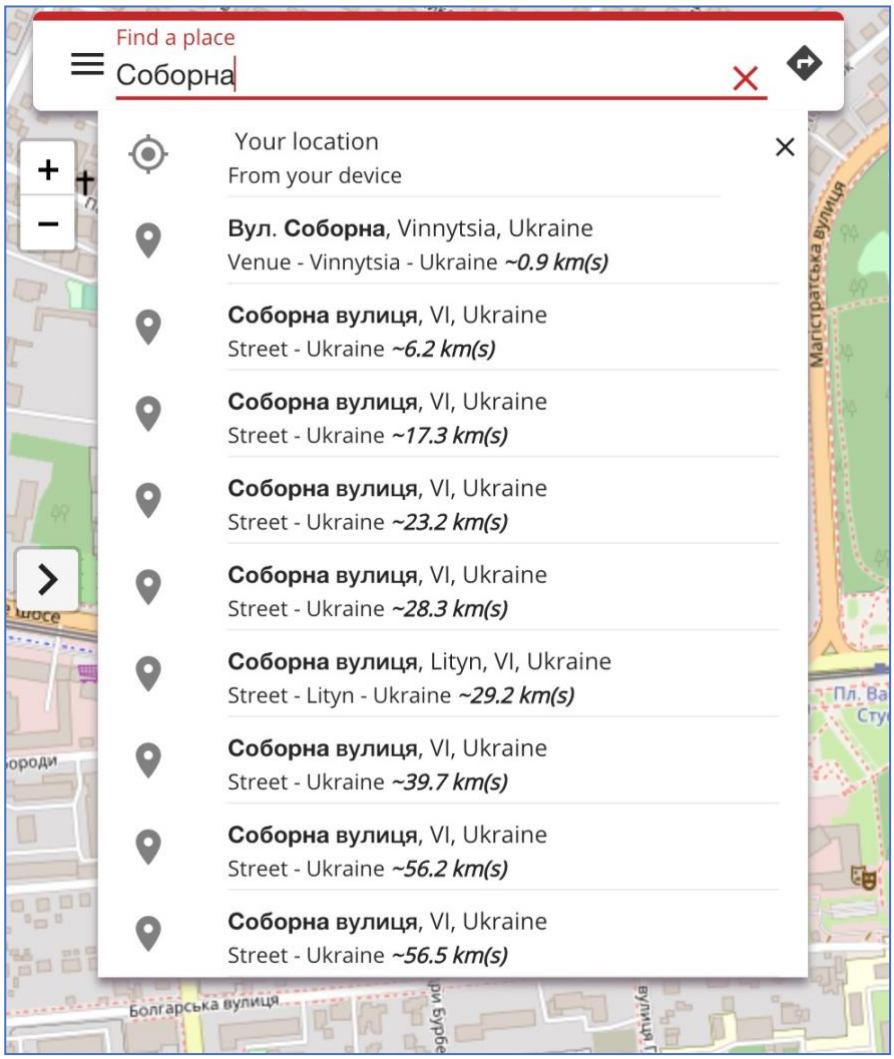

Рисунок 4.3 — Контекстний список системи

Цей етап тестування показує що система підбирає варіанти які суміжні до того що хоче отримати користувач, тобто все працює правильно. Якщо ж користувач вибере якесь місце на карті, то має з’явитись вікно з інформацією про це місце та певні дії з цією інформацією. Вигляд вікна зображено на рисунку. 4.4.

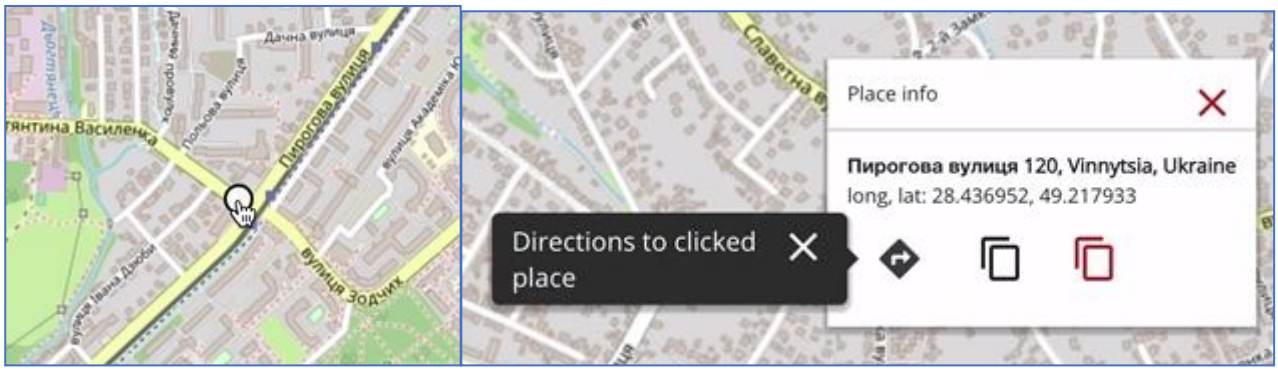

Рисунок 4.4 — Додаткова інформація про місце на карті

Як видно на рисунку, функціонал додавання елементу на карту, тобто точки в місці яке обрав користувач працює. Вікно з додатковою інформацією про місце відображається, в цьому вікні користувач може побачити адресу цього місця. Прокласти маршрут до нього від свого поточного місця знаходження, скопіювати ці значення, або передати їх до інтерфейсу програми. За рахунок того що всі елементи відображаються та працюють належним чином, цей етап тестування також можна вважати пройденим.

При виборі двох точок користувачем, система повинна надавати можливість побудувати маршрут між ними і візуалізувати його на мапі, які покажуть початкову та кінцеву точки. При цьому на мапі також повинен відображатися приблизний час, необхідний для подорожі цим маршрутом, і загальна відстань між цими точками. У головному інтерфейсі користувача мають бути доступні деталі маршруту, такі як назви початкової та кінцевої точок, а також кнопки для додавання ще однієї точки до маршруту та очищення всіх полів для нового вибору точок (див. рисунок 4.5). Це забезпечить зручність та інформативність користування системою для планування маршрутів.

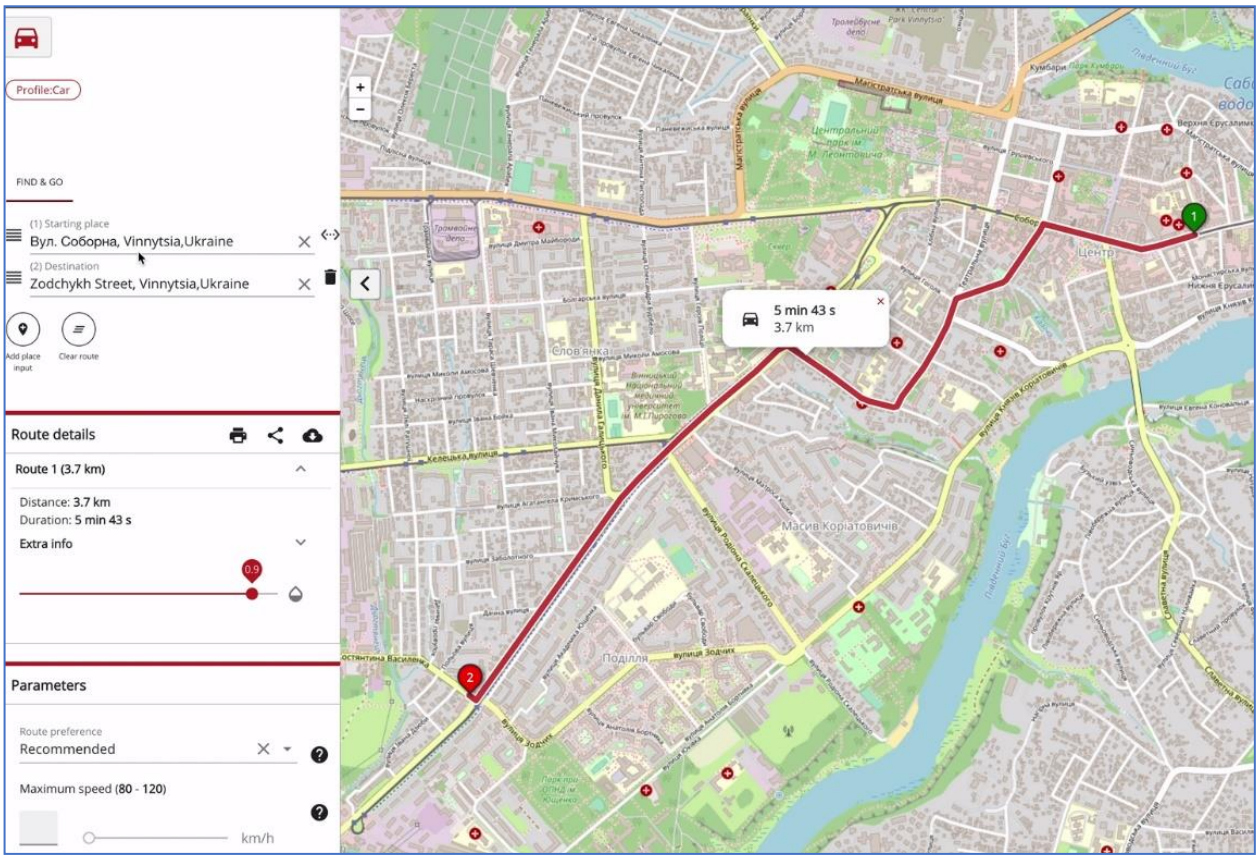

Рисунок 4.5 — Додаткова інформація про місце на карті

На мапі точки показуються як 1 (початкова) та 2 (кінцева) що також підкреслює зручність та орієнтацію користувача. На мапі все відображається добре, всі елементи помітні. Маршрут відображається між тими очками які обрав користувач, тобто система працює правильно. Тестування цього функціоналу можна вважати успішним.

Також користувач може обрати додаткові параметри які будуть впливати на побудову маршруту. Для початку користувач може обрати або рекомендований системою маршрут, або найкоротший. Вибір маршруту зображено на рисунку. 4.6.

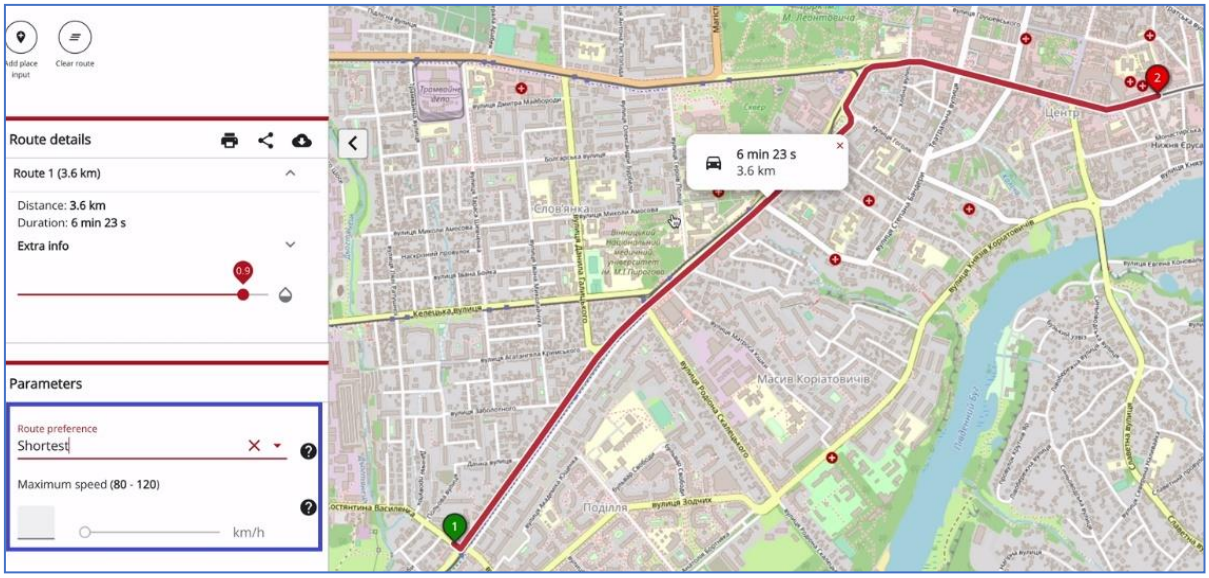

Рисунок 4.6 — Вибір найкоротшого маршруту

Коли користувач обрав найкоротший маршрут система перебудовує маршрут з урахуванням цього та відображає новий маршрут з новими даними часу та дистанції. Цей етап тестування система також пройшла успішно, новий маршрут виводиться інформація для нього оновлюється також.

В системі реалізований функціонал уникання певних об’єктів під час побудови маршруту. Вигляд вікна зображено на рисунку 4.7.

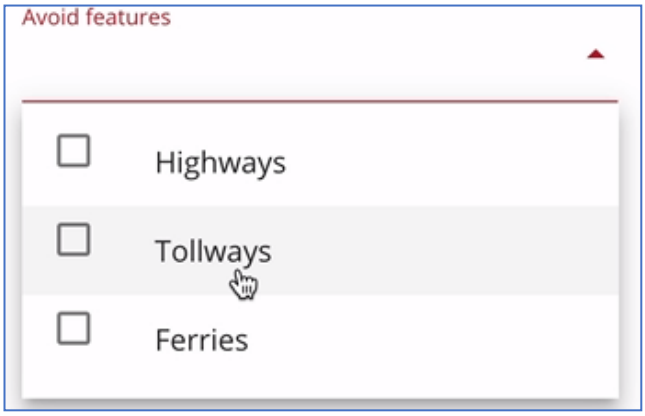

Рисунок 4.7 — Список об’єктів уникання під час побудови маршруту

Такими об’єктами є шосе, платні дороги та пароми. Система виводить це список користувачу. Та враховує ці параметри під час створення маршруту. Тестування успішно пройдено.

Також в системі реалізовано побудову маршруту враховуючи максимальну мінімальну швидкість яку користувач може регулювати від 20 до 120 км/год. Це зображено на рисунку 4.8.

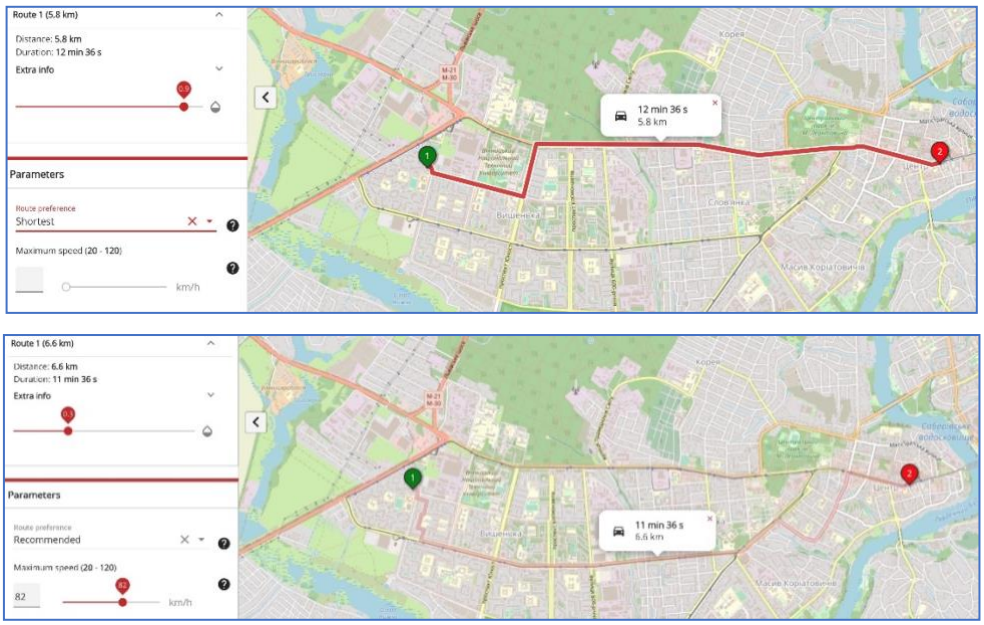

Рисунок 4.8 — Зміна маршруту з врахуванням параметру швидкості

Як і під час зміни рекомендованого або найшвидшого маршруту, так і при зміні параметрів швидкості система має змінювати маршрут та його дані. На рисунку продемонстровано що система змінює поточний маршрут, та виводить нові дані. Тестування можна вважати успішним.

В системі також реалізований функціонал створення альтернативного маршруту. Користувач може обрати ще декілька альтернативних маршрутів до свого поточного (див. рисунок 4.9). Налаштувавши для цього деякі параметри. Параметр який відповідає за те наскільки у % альтернативний маршрут може бути схожим на основний, та параметр часу\складності наскільки альтернативний маршрут може бути складним відповідно і довшим в порівнянні

з основним, теж у % співвідношенні. Система повинна відображати всі ці елементи інтерфейсу щоб користувач міг їх налаштовувати під себе. Також система має відображати зміни, тобто новий маршрут складений за налаштуваннями користувача.

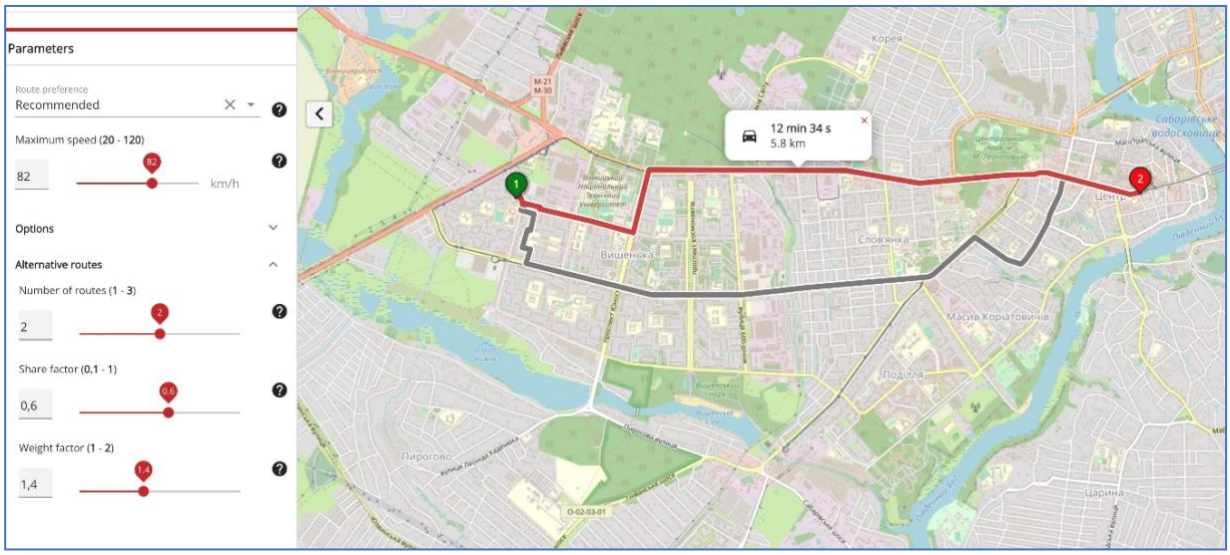

Рисунок 4.9 — Зміна маршруту з врахуванням параметру швидкості

Як можна побачити на рисунку система правильно показує альтернативний маршрут, роблячи його сірим кольором а основний червоного що додає певної зручності. Користувач може перемикатись між цими маршрутами, роблячи основним будь який з них. Так як система виконує всі необхідні дії цього етапу то тестування можна вважати успішним.

Тестування основного функціоналу пройдено успішно. На всіх етапах система видавала очікуваний результат, а всі дані відображалися коректно та зрозуміло. Значною перевагою цієї системи є інтерфейс та алгоритм побудови маршруту, які реагують на налаштування користувача та змінюють маршрут в залежності від них. Це дозволяє користувачам налаштувати систему під свої потреби та отримувати оптимальний маршрут для своїх конкретних вимог.

4.3 Інструкція користувача та технічні вимоги системи

Основною технічною вимогою для використання системи є наявність підключеного до Інтернету пристрою з сучасним веб-браузером. Це означає, що користувачі можуть отримати доступ до системи з будь-якого підтримуваного пристрою, такого як стаціонарний комп’ютер, ноутбук, планшет або смартфон.

Завдяки цій універсальності, користувачі можуть легко отримати доступ до системи з будь-якого місця та в будь-який час, за умови наявності інтернет-підключення. Це забезпечує зручність та гнучкість для користувачів, оскільки вони можуть використовувати систему на своєму улюбленому пристрої зручності.

Для користувачів було створено таблицю використання (див. таблицю 4.2), де кожна дія користувача має відповідну реакцію з боку системи. Це є чудовим рішенням, яке сприяє зрозумінню та легкому взаємодії з системою. Ця таблиця допомагає користувачам легко зорієнтуватися в системі та швидко зрозуміти, як правильно виконати необхідні дії. Вона забезпечує їм певний рівень впевненості та контролю над взаємодією з системою. Крім того, таблиця використання є зрозумілим та структурованим рішенням. Вона організована за діями та функціями, що полегшує пошук необхідної інформації для конкретних завдань.

Таблиця 4.2 — Інструкція використання системи

Функція	**Дія користувача**	**Реакція системи**
Доступ до системи	Відкрийте веб-браузер і введіть URL-адресу системи в адресному рядку.	Система завантажується, користувач отримує доступ до інтерфейсу карти та планування маршруту.
Відображення базової карти	Не застосовується	На карті відображаються міста, містечка та регіони з чіткими назвами географічних об’єктів
Навігація та збільшення/зменшення	Використовуйте елементи керування масштабом.	Карта збільшується або зменшується відповідно.

Продовження таблиці 4.2 — Інструкція використання системи

Функція	**Дія користувача**	**Реакція системи**
Введіть назву місця або координати в відповідному полі.	Карта центрується на вказаному місці.	
Переключайтесь між різними шарами карти (наприклад, топографічний, супутниковий).	Вигляд карти змінюється в залежності від обраного шару.	
Натискайте на кнопки масштабування.	Карта збільшується або зменшується згідно з вказівками.	
Відображення вашого поточного місцезнаходження, якщо воно доступне.	Ваше місцезнаходження позначається на карті.	
Відкрийте повний вид карти.	Карта розширюється до повноекранного режиму.	
Пошук та вибір об’єктів	Введіть назву місця або координати в поле пошуку.	Система пропонує варіанти зі списку контексту
Виберіть місце на карті.	Вікно з інформацією про це місце та пов’язаними діями відображається	
Планування маршруту	Виберіть дві точки на карті.	Система побудує маршрут, показуючи початкову і кінцеву точки, час подорожі та загальну відстань
Перегляньте деталі маршруту, включаючи назви точок.	Деталі маршруту відображаються в інтерфейсі користувача.	
Додайте додаткові точки до маршруту або очистіть всі поля для нового вибору точок.	Ви можете додавати додаткові точки або скидати маршрут для нового вибору.	
Додаткові параметри маршруту	Виберіть між рекомендованим системою маршрутом або найкоротшим маршрутом.	Система перераховує і відображає обраний маршрут з оновленими даними
Налаштуйте максимальну та мінімальну швидкість для розрахунку маршруту (від 20 до 120 км/год).	Система оновлює маршрут відповідно до вибраних параметрів швидкості	
Альтернативні маршрути	Створюйте альтернативні маршрути та налаштовуйте параметри (схожість, час, складність)	Система відображає кілька альтернативних маршрутів на основі ваших вподобань

4.4 Висновки

Четвертий розділ дослідження присвячений одному з ключових аспектів розробки програмного продукту – тестуванню. Тестування відіграє важливу роль у процесі розробки, оскільки воно дозволяє перевірити функціональність системи, її надійність та відповідність встановленим вимогам.

У цьому розділі розглядаються обрані методи тестування, які були застосовані для перевірки розробленої системи. Цей етап має велике значення для забезпечення високої якості продукту та ідентифікації потенційних недоліків або помилок до їх впровадження у реальні умови. Тестування включає аналіз різних аспектів системи, від основного функціоналу до конкретних сценаріїв використання.

Під час тестування на різних етапах розробки система продемонструвала високу якість, що свідчить про її потенціал як надійного програмного продукту для маршрутизації та навігації. Тестування не тільки підкреслило сильні сторони системи, але й допомогло виявити та виправити можливі слабкі місця.

Також, у розділі представлено інструкцію з користування системою. Цей елемент є важливим для забезпечення зручності та інтуїтивної взаємодії користувачів з системою. Інструкція містить настанови по використанню ключових функцій, а також рекомендації та поради, які допоможуть користувачам максимально ефективно використовувати систему для задоволення своїх потреб.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Для визначення можливостей та перспектив комерціалізації наукової розробки за темою «Дослідження методів оптимізації траєкторії руху транспортних засобів у програмній системі позиціювання в режимі реального часу» необхідне здійснення оцінювання її комерційного потенціалу.

Для оцінки комерційного потенціалу доцільним є здійснення комерційно-технологічного аудиту, що полягає в опитуванні групи експертів щодо списку критеріїв.

При оцінюванні аспектів комерційного потенціалу наукової розробки були залучені експерти з кафедри програмного забезпечення Вінницького національного технічного університету: експерт 1 (Бобко О.Б., студент групи 1ПІ-22М, займає посаду провідного інженера програміста у компанії ТОВ «ЕПАМ ДІДЖИТАЛ»), експерт 2 (Нестерук В.О., студент групи 1ПІ-22М, займає посаду старшого інженера програміста у компанії ТОВ «ЕПАМ ДІДЖИТАЛ»), експерт 3 (Решетнік О.О., займає посаду провідного інженера-програміста у компанії ТОВ «ЕПАМ ДІДЖИТАЛ»).

У таблиці 5.1 надано результати опитування експертів щодо комерційного потенціалу наукової розробки.

Критеріями оцінювання є: технічна здійсненність концепції, ринкові переваги (наявність аналогів), ринкові переваги (ціна продукту), ринкові переваги (технічні властивості), ринкові переваги (експлуатаційні витрати), ринкові перспективи (розмір ринку), ринкові перспективи (конкуренція), практична здійсненність (наявність фахівців), практична здійсненність (наявність фінансів), практична здійсненність (потреба нових матеріалів), практична здійсненність (термін реалізації), практична здійсненність (розробка документів) [31].

Кожен параметр оцінений експертами за п’ятибальною шкалою – від 0 до 5.

Таблиця 5.1 – Результати оцінювання комерційного потенціалу наукової розробки

Критерії	**Експерт**		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	4	5	4
2. Ринкові переваги (наявність аналогів)	3	3	4
3. Ринкові переваги (ціна продукту)	3	3	3
4. Ринкові переваги (технічні властивості)	4	4	3
5. Ринкові переваги (експлуатаційні витрати)	4	4	4
6. Ринкові перспективи (розмір ринку)	4	4	4
7. Ринкові перспективи (конкуренція)	3	3	3
8. Практична здійсненність (наявність фахівців)	4	4	3
9. Практична здійсненність (наявність фінансів)	4	3	4
10. Практична здійсненність (необхідність нових матеріалів)	4	4	4
11. Практична здійсненність (термін реалізації)	5	4	4
12. Практична здійсненність (розробка документів)	3	3	3
Сума балів	45	44	43
Середньоарифметична сума балів $СБ_с$	$СБ_с = \frac{\sum_1^3 СБ_i}{3} = 44$		

Результат оцінювання відповідає проміжку 41…48 та означає [31], що для наукової розробки за темою «Методи оптимізації траєкторії руху транспортних засобів у програмній системі позиціювання в режимі реального часу» рівень комерційного потенціалу високий.

До аналогів науково-технічної розробки належать: Google Maps platform, Waze, Route4Me.

Аналогом для порівняння було обрано Google Maps.

Основними недоліками аналога є: відсутність функції побудови оптимального маршруту з урахуванням заторів та ремонтів доріг, обмежені можливості планування поїздок групою користувачів, відсутність інформації про вартість проїзду різними маршрутами. Перевагами нової розробки відносно аналогу є: автоматичне врахування заторів та ремонтів при побудові оптимального маршруту, можливість спільного планування поїздок групою користувачів, надання інформації про вартість проїзду різними маршрутами з урахуванням виду транспорту.

У таблиці 5.2 наведено порівняння технічних параметрів аналога та науково-технічної розробки.

Параметр «функціональність» означає кількість наявних у розробки функцій з переліку (двовіконна візуалізація зображень, обчислення відстані між перешкодами, збереження маршрутів у бібліотеку, можливість прокладення шляху, вводу точок). Коефіцієнт вагомості параметра становить 0.4.

Параметр «простота використання» означає кількість властивостей з переліку (відсутність необхідності знання програмних мов, простота інтерфейсу, відсутність необхідності встановлення додаткових програм, браузерів). Коефіцієнт вагомості параметра становить 0.3.

Параметр «кількість мовних версій інтерфейсу» означає число мов, на яких доступний контент у засобі. Коефіцієнт вагомості параметра – 0.1.

Параметр «оптимізація руху» означає, на основі яких у даних можна показати кращий маршрут. Коефіцієнт вагомості параметра – 0.2.

Таблиця 5.2 – Основні технічні показники аналога і нового програмного продукту

Параметр	**Аналог (балів)**	**Нова Розробка (балів)**	**Індекс зміни значення параметра**	**Коефіцієнт вагомості**
Функціональність	3	4	1.33	0.4
Простота використання	1	3	3	0.3
Кількість мовних версій інтерфейсу	10	3	0.3	0.1
Кількість вбудованих моделей відбиття	2	3	1.5	0.2

Груповий параметричний індекс за технічними показниками розраховується за формулою [31]

$$I_{TP} = \sum_{i=1}^{n} a_i * q_i \quad (5.1)$$

де q_i - одиничний параметричний показник, a_i вагомість параметричного показника.

Розраховуємо значення групового індекса за технічними показниками

$$I_{TP} = 0.4 * 1.33 + 0.3 * 3 + 0.1 * 0.3 + 0.2 * 1.5 = 1.73$$

За технічними показниками розробка переважає аналог у 1.73 разів.

Отже, наукова розробка характеризується високим комерційним потенціалом за рахунок розширення функціональних можливостей (можливість прокладення доступнішого маршруту, обчислення відстані між точками) та підвищення простоти використання (не вимагається знання мов програмування, встановлення сторонніх програм).

Нова розробка може бути використана водіями, перевізниками, та службами доставки національного рівня. Реалізовані у програмі алгоритми можуть бути використані для покращення руху транспортнх засобів та

уникнення заторів. Технічна готовність розробки становить 70%, наявний готовий промисловий зразок.

5.2 Розрахунок витрат на здійснення науково-дослідної роботи

Розрахунок витрат для науково-дослідної роботи включає здійснення обчислень по статтях калькуляції [31].

Витрати на основну заробітну плату розраховуються за формулою

$$З_о = \sum_{i=1}^{k} \frac{M}{T_P} * t \quad (5.2)$$

де М – місячний посадовий оклад розробника,T_p - кількість робочих днів у місяці (21), t – кількість днів роботи дослідника, k – кількість посад. Основна заробітна плата розробників розрахована у таблиці 5.3.

Таблиця 5.3 – Розрахунки основної заробітної плати розробників

Працівник	**Оклад, грн.**	**Оплата за робочий день, грн.**	**Число днів роботи**	**Витрати на оплату праці, грн.**
Науковий керівник	35000	1167	105	122535
Інженер-програміст	30000	1000	85	85000
Всього				207535

Витрати на додаткову заробітну плату дослідників розраховуються за формулою

$$З_{дод} = З_о * \frac{10\%}{100} \quad (5.3)$$

Додаткова заробітна плата дослідників становить

$$З_{дод} = 207535 * 0.1 = 20753 \text{ грн.}$$

Відрахування на соціальні заходи обчислюються за формулою

$$З_н = (З_о + З_{дод}) * 22\% / 100 \quad (5.4)$$

Відрахування на соціальні заходи становлять

$$З_н = (207535 + 20753) * 0.22 = 50223 \text{ грн.}$$

Витрати на програмне забезпечення, що використовується при дослідженні науково-технічної розробки, розраховуються за формулою

$$В_{прг} = \sum_{i=1}^{k} Ц_{ірпг} * С_{прг.і} * K_i \quad (5.5)$$

де $Ц_{ірпг}$– ціна придбання одиниці ПЗ (грн.), $С_{прг.і}$ – кількість придбаних одиниць, K_i – коефіцієнт врахування інсталяції, k – кількість найменувань ПЗ

У таблиці 5.5 розраховано витрати на програмне забезпечення

Таблиця 5.5 – Витрати на програмне забезпечення

Назва	**Кількість, шт**	**Ціна за одиницю, грн**	**Вартість, грн**
MS Office 2021	1	4950	5445
Microsoft Visual Studio 2021	1	12545	13799
Геокодек	1	20000	22000
Хмарні сервіси Google Cloud	1	1500	1650
Всього			42894

Амортизаційні відрахування для обладнання розраховуються через прямолінійний метод згідно з формулою

$$A_{\text{обл}} = \frac{\text{Ц}_\text{б}}{\text{Т}_\text{в}} * \frac{t_{\text{вик}}}{12} \quad (5.6)$$

де $\text{Ц}_\text{б}$– балансова вартість обладнання, програм, приміщень (грн), $t_{\text{вик}}$– термін використання обладнання, програм, приміщень (місяців), $T_\text{в}$ – термін корисного використання обладнання, програм, приміщень (років).

Обчислення амортизаційних відрахувань представлені у таблиці 5.6.

Таблиця 5.6 – Амортизаційні відрахування по кожному виду обладнання

Найменування	**Балансова вартість, грн**	**Строк корисного використання, років**	**Термін використання обладнання, місяців**	**Амортизаційні відрахування, грн**
Ноутбук ASUS	44799	5	3	2240
Принтер Canon	26999	5	3	1350
MS Office 2021	4950	10	3	247
Microsoft Visual Studio 2021	12545	10	3	627
Геокодек	20000	20	3	1000
Хмарні сервіси Google Cloud	1500	1	3	75
Всього				5539

До статті «Паливо та енергія для науково-виробничих цілей» відносяться витрати на всі види палива й енергії, що безпосередньо використовуються з технологічною метою на проведення досліджень.

$$В_е = \sum_{i=1}^{n} \frac{W_{yt} * t_i * Ц_е * К_{впі}}{n_i} \quad (5.7)$$

де W_{yt}- встановлена потужність обладнання кВт;

t_i- тривалість роботи обладнання, год;

$Ц_е$- вартість 1 кВт/год електроенергії, грн;

$K_{впі}$- коефіцієнт, що враховує використання потужності, $K_{впі} < 1$;

n_i – ККД обладнання, $n_i < 1$;

Для написання магістерської роботи використовується ноутбук та принтер.

Потужність ноутбука – 0.5 кВТ/год, принтера – 0.025 кВТ/год. Розрахуємо вартість електроенергії.

$$В_е = \frac{0.5*1520*7,5*0,5}{0.8} + \frac{0.025*10*7.5*0,8}{0.6} = 3562,5 + 2.5 = 3565 \text{ (грн)}$$

Витрати на матеріали розраховано у таблиці 5.4.

Таблиця 5.4 – Розрахунок витрат на матеріали

Найменування матеріалу	**Кількість, шт.**	**Ціна за штуку, грн.**	**Сума, грн.**
Флеш-пам’ять	1	4300	4730
Картидж	1	1200	1320

Продовження Таблиці 5.4 – Розрахунок витрат на матеріали

Найменування матеріалу	**Кількість, шт.**	**Ціна за штуку, грн.**	**Сума, грн.**
Папір офісний А4 80 г / м2 500 аркушів	1	329	361.9
Набір олівців чорнографітних 12 шт.	1	85	93.5
Всього			6505.4

Витрати, які не ввійшли до попередніх статей, враховуються в інших витратах. Інші витрати становлять 50% ($Н_{ів}$) від основної заробітної плати дослідників, обчислюються за формулою.

$$І_в = (З_о + З_р) * \frac{Н_{ів}}{100\%} \quad (5.8)$$

де $Н_{ів}$- норма нарахування за статтею «Інші витрати»

Інші витрати становлять

$$І_в = 207535 * 0.5 = 103767 \text{ (грн.)}$$

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 бан-ків; витрати, пов’язані з освоєнням виробництва продукції; витрати на на-уково-технічну інформацію та рекламу та ін.

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

$$В_{нзв} = (З_о + З_р) * \frac{Н_{нзв}}{100\%} \quad (5.9)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати».

Інших категорій витрат не передбачено.

Розрахуємо загальновиробничі витрати

$$В_{нзв} = 207535 * 1.5 = 311302.5 \text{ (грн)}$$

Сумарні витрати на проведення науково-дослідної роботи обчислюються за формулою [31]

$$В_{заг} = З_о + З_{дод} + З_н + M + В_{прг} + А_{обл} + І_в + В_{нзв} + В_е \quad (5.10)$$

Отже, сумарні витрати на проведення науково-дослідної роботи

$$В_{заг} = 207535 + 20753 + 50223 + 6505.4 + 42894 + 5539 + 103767 + 311302.5 + 3565 = 752083.9 \text{ (грн).}$$

Загальні витрати на завершення науково-технічної роботи обчислюються

$$ЗВ = \frac{В_{заг}}{n} \quad (5.11)$$

де n – коефіцієнт виконання наукової роботи (0.7).

Загальні витрати на завершення розробки становлять

$$ЗВ = \frac{752083.9}{0.7} = 1074405.5 \text{ (грн)}$$

5.3 Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором

Чистий прибуток є прибутком, що залишається після врахування витрат на виробництво, амортизації, сплати податків. Річне збільшення чистого прибутку від реалізації розробки обчислюється за формулою [31].

$$\Delta П_i = (\pm \Delta Ц_о * N * Ц_о * \Delta N) * \lambda * p * (1 - \frac{\vartheta}{100}) \quad (5.12)$$

Де N – основний показник, ΔN – зміна N після впровадження розробки у розглянутому році, $Ц_о$ – основний якісний показник (ціна реалізації після впровадження розробки), $\pm\Delta Ц_о$ - зміна якісного показника від впровадження розробки).

У даному випадку N є кількістю користувачів, ΔN - зміною кількості користувачів, $\pm\Delta Ц_о$ – зміна вартості після впровадження результатів розробки (взято 500 грн), $Ц_о$ – вартість програми після впровадження науково-технічної розробки, приймемо як 1000 грн + $\pm\Delta Ц_о$.

Прогнозується, що протягом 3 років отримуються позитивні результати від впровадження розробки.

Передбачається, що ΔN у першому році становить 400, у другому 850, у третьому 1050. N до реалізації становить 3000.

Обчислимо зміну чистого прибутку (грн) у кожному з цих років згідно з виразами:

$$\Delta П_1 = (500 * 3000 + (1000 + 500) * 400) * 0.8333 * 0.5 * (1 - 0.18) = 717471.3 \text{ (грн)}$$

$$\Delta П_2 = (500 * 3000 + (1000 + 500) * 850) * 0.8333 * 0.5 * (1 - 0.18) = 948087 \text{ (грн)}$$

$$\Delta П_3 = (500 * 3000 + (1000 + 500) * 1050) * 0.8333 * 0.5 * (1 - 0.18) = 1050583 \text{ (грн)}$$

Приведена вартість збільшення всіх чистих прибутків від впровадження розробки обчислюється за формулою

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t} \quad (5.13)$$

де T – роки, протягом яких очікується отримання позитивних результатів від впровадження та комерціалізації розробки, Δ – ставка дисконтування (0.15),

t – період від впровадження розробки до отримання інвестором додаткових чистих прибутків у році.

$$ПП = 717471/(1 + 0.15) + 948087/(1 + 0.15) * 2 + 1050583/(1 + 0.15) * 3 = 6049485.4 \text{ (грн)}$$

Початкові інвестиції для вкладення інвестором обчислюються за формулою

$$PV = k_{розр} * ЗВ \quad (5.14)$$

де $K_{розр}$k розр. - коефіцієнт витрат на впровадження розробки.

Обчислимо значення початкових інвстицій для вкладання інвестором

$$PV = 2 * 1074405.5 = 2148811 \text{ (грн)}$$

Абсолютний економічний ефект від впровадження розробки є чистим приведеним доходом, обчислюється за формулою

$$Е_{абс} = ПП - PV \quad (5.15)$$

В результаті абсолютний економічний ефект від впровадження розробки

$$Е_{абс} = 6049485.4 - 2148811 = 3900674.4 \text{ (грн)}$$

Отже, можлива доцільність впровадження розробки.

Внутрішня економічна дохідність допомагає остаточно визначити, чи доцільно впроваджувати розробку, розраховується за формулою

$$Е_в = \sqrt[Т_ж]{1 + \frac{Е_{абс}}{PV}} - 1 \quad (5.16)$$

де $Т_ж$– життєвий цикл розробки.

Розрахуємо $Е_в$

$$Е_в = \sqrt[3]{1 + \frac{3900674.4}{2144110}} - 1 = 0.41 = 41\%$$

Мінімальна внутрішня економічна дохідність інвестицій τ мін показує, чи буде інвестор зацікавлений в інвестиціях, обчислюється за формулою

$$\tau_{\text{мін}} = d + f \tag{5.17}$$

де d – середньозважена ставка за депозитними операціями в комерційних банках (обрано 0.09), f – ризикованість вкладення інвестицій (обрано 0.1).

$$\tau_{\text{мін}} = 0.09 + 0.1 = 0.19 = 19\%$$

Внутрішня економічна дохідність більше ніж мінімальна внутрішня економічна дохідність, тому інвестори повинні бути зацікавлені у фінансуванні розробки.

Період окупності обчислюється за формулою

$$\text{Т}_{\text{ок}} = \frac{1}{\text{Е}_{\text{в}}} \tag{5.18}$$

Отримуємо значення 1 / 0.41 = 2.4 роки. Отриманий період окупності менший за 3 роки, тому наукова розробка є комерційно привабливою.

5.4 Висновки

Отже, у даному розділі досліджено економічні аспекти науково-технічної розробки за темою «Дослідження методів оптимізації траєкторії руху транспортних засобів у програмній системі позиціювання в режимі реального часу». Здійснено оцінювання комерційного потенціалу науково-технічної розробки шляхом систематизації результатів опитування експертів. Обчислено витрати за статтями: витрати на основну заробітну плату розробників, на додаткову заробітну плату дослідників, на соціальні заходи, на комплектуючі, на програмне забезпечення, амортизаційні відрахування для обладнання, витрати, які не ввійшли до попередніх статей. Обчислено загальні витрати на завершення розробки. Розраховано річне збільшення чистого прибутку від реалізації для трьох років, на основі розрахунків знайдено приведену вартість збільшення всіх

чистих прибутків. Здійснено порівняння внутрішньої економічної дохідності та мінімальної внутрішньої економічної дохідності для прогнозування інтересу інвестора у фінансуванні. Розраховано період окупності розробки.

В результаті аналізу опитування експертів показано високий комерційний потенціал науково-технічної розробки. Період окупності дорівнює 2.4 роки. Отже, показано, що розробка є комерційно привабливою. Це підкреслює значення проекту не тільки з науково-технічної, але й з економічної точки зору. Розробка пропонує інноваційні рішення, які можуть бути вигідно використані в різних галузях, включаючи транспорт, логістику, управління міським рухом та інші сфери, де важливо ефективно оптимізувати маршрути в реальному часі.

Крім того, розробка може сприяти підвищенню безпеки дорожнього руху, зниженню витрат на паливо та часу в дорозі, що є важливими факторами для бізнесу та кінцевих споживачів. Таким чином, інвестиції в цю розробку не тільки обіцяють фінансову вигоду, але й сприяють соціальному та екологічному благополуччю, що робить її привабливою для широкого спектру потенційних інвесторів.

ВИСНОВКИ

У магістерській кваліфікаційній роботі досліджено методи оптимізації траєкторій руху транспортних засобів у програмній системі позиціонування в реальному часу, досягнуто поставленої мети та виконано всі заплановані задачі.

Детально проаналізовано існуючі методи оптимізації траєекторії руху транспортних засобів у режимі реального часу, де було виявлено ключові аспекти, пов'язані з ефективною навігацією, проаналізовано плюси і мінуси існуючих алгоритмів. По ряду зазначених показників особлива увага була приділена алгоритму Дейкстри та потенціалу його вдосконалення для систем позиціонування в реальному часі.

Модифіковано метод двонаправленого алгоритму Дейкстри для оптимізації траєкторій руху транспортних засобів та метод адаптивної мультимодальної оптимізації траєкторій, що впливає на вагу ребер графів.

Далі було зосереджено увагу на структурі та архітектурі системи геопросторової маршрутизації. Вибір технологічного стеку, включаючи JavaScript, Java, Vue.js, Leaflet, Spring Boot, забезпечив системі потрібну надійність та адаптивність. Клієнт-серверна архітектура з використанням REST API сприяла чіткій координації між компонентами системи.

Розроблено програмну систему позиціонування в реальному часу, модифіковано методи оптимізації траєкторій руху транспортних засобів. Розглянуто вагові класи для модифікації маршрутів, що включають в себе різні фактори, такі як крутизна доріг та максимальна дозволена швидкість.

Тестування розробленої системи підтвердило її високу якість та відповідність встановленим вимогам. Також було представлено інструкцію з користування системою, що забезпечує зручність та інтуїтивну взаємодію користувачів з системою.

Проведено оцінку економічного аспекту розробки. Аналіз комерційного потенціалу та оцінка витрат показали, що проект є комерційно привабливим з періодом окупності у 2.4 роки.

Таким чином, інвестиції в цю розробку не тільки обіцяють фінансову вигоду, але й виконують соціальну та екологічну місію, що робить її привабливою для широкого спектру потенційних інвесторів.

Загалом, результати магістерської кваліфікаційної роботи демонструють успішне виконання поставлених задач, включаючи розробку та тестування ефективних методів оптимізації траєкторій, а також підтверджують високий комерційний потенціал розробки. Розробка відкриває нові перспективи для підвищення ефективності та безпеки транспортних систем, забезпечуючи значний внесок у сферу геопросторової навігації та маршрутизації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Гончар С.І., Рейда О.М. Вдосконалений алгоритм дейкстри // Сучасні комп’ютерні системи та мережі в управлінні – 2023 : матеріали VI Всеукраїнської науково-практичної інтернет-конференції молодих вчених та студентів «Сучасні інформаційні системи та технології»
Херсон, 30.11.23 р. [Електронний ресурс]. URL: http://kntu.net.ua/ukr/Pro-universitet2/Novini-universitetu/VI-Vseukrayins-ka-naukovo-praktichna-konferenciya-studentiv-aspirantiv-ta-molodih-vchenih-z-avtomatichnogo-upravlinnya
2. Гончар С.І., Рейда О.М. Оптимізація траєкторії руху транспортних засобів: алгоритм А* // Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії – 2023 : матеріали Всеукраїнської науково-технічної конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ) Вінниця, 20.03.23 р. - 23.03.23 р. [Електронний ресурс]. URL: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2023
3. Top 10 Benefits of Route Optimization You Must Know. [Електроний ресурс] URL: https://www.upperinc.com/blog/benefits-of-route-optimization/.
4. A* Search Algorithm. [Електроний ресурс] URL: codegym.cc/groups/posts/a-search-algorithm-in-java.
5. What is A* Search Algorithm?. [Електроний ресурс] URL: https://www.mygreatlearning.com/blog/a-search-algorithm-in-artificial-intelligence/.
6. Алгоритм Дейкстри [Електроний ресурс] URL: https://ua5.org/algorithm/1970-algorytm-dejkstry.html.
7. Bellman Ford’s Algorithm [Електроний ресурс] URL: https://www.programiz.com/dsa/bellman-ford-algorithm.
8. Griffiths D J, Mehdi Qasim H, Wang Tingkai and Gough E Norman. A GENETIC ALGORITHM FOR PATH PLANNING. School of Computing and Information Technology, University Of Wolverhampton, Wolverhampton.

9. Floyd Warshall Algorithm [Електроний ресурс] URL: https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/.

10. AI Route Optimization & Route Planning Guide: How AI Routing Can Transform Route Optimization [Електроний ресурс] URL: fareye.com/resources/blogs/ai-route-optimization.

11. Comparative Analysis of Optimal Path Search Algorithms [Електроний ресурс] URL:researchgate.net/publication/347596106_Comparative_Analysis_of_Optimal_Path_Search_Algorithms.

12. Yizhen Huang, Qingming Yi, Min Shi An Improved Dijkstra Shortest Path Algorithm College of Information Science and Technology, Jinan University.

13. Reasons Why JavaScript is the World's Most Popular Programming Language [Електроний ресурс] URL: https://www.trienpont.com/javascript/;

14. Коли та чому Java використовується для розробки? [Електроний ресурс] URL: https://wezom.com.ua/ua/blog/kogda-i-pochemu-java-ispolzuetsja-dlja-razrabotki-prilozhenij.

15. The Progressive JavaScript Framework [Електроний ресурс] URL: https://vuejs.org/.

16. an open-source JavaScript library for mobile-friendly interactive maps [Електроний ресурс] URL: https://leafletjs.com/.

17. Vue Component Framework [Електроний ресурс] URL: https://vuetifyjs.com/en/.

18. The JavaScript API to consume openrouteservice(s) painlessly! [Електроний ресурс] URL: https://github.com/GIScience/openrouteservice-js.

19. Заснований на Promise HTTP клієнт для браузеру та Nodejs [Електроний ресурс] URL: https://axios-http.com/uk/docs/intro.

20. Spring Boot [Електроний ресурс] URL: https://spring.io/projects/spring-boot.

21. springdoc-openapi v2.2.0 [Електроний ресурс] URL: https://springdoc.org/.

22. Swagger Parser [Електроний ресурс] URL: https://github.com/swagger-api/swagger-parser.

23. Parsing YAML with SnakeYAML [Електроний ресурс] URL: https://www.baeldung.com/java-snake-yaml.

24. GeoTools [Електроний ресурс] URL: https://www.osgeo.org/.

25. Jackson Project Home [Електроний ресурс] URL: https://github.com/FasterXML/jackson.

26. Ukraine (with Crimea) [Електроний ресурс] URL: https://download.geofabrik.de/europe/ukraine.html.

27. OpenStreetMap Data in Layered GIS Format [Електроний ресурс] URL: https://download.geofabrik.de/osm-data-in-gis-formats-free.pdf.

28. Why REST API for data transfer? [Електроний ресурс] URL: https://medium.com/@deep2017.arjun/why-rest-api-for-data-transfer-f66efc31a079.

29. The different types of software testing? [Електроний ресурс] URL: https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing.

30. Difference between System Testing and Acceptance Testing [Електроний ресурс] URL: geeksforgeeks.org/difference-between-system-testing-and-acceptance-testing/.

31. Differences between Black Box Testing and White Box Testing [Електроний ресурс] URL: browserstack.com/guide/black-box-testing-and-white-box-testing.

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

ДОДАТКИ

ДОДАТОК А. Технічне завдання

98

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

ЗАТВЕРДЖУЮ
д.т.н., професор Романюк О.Н.
"19" 09 2023 р.

Технічне завдання
на магістерську кваліфікаційну роботу «Дослідження методів оптимізації траєкторії руху транспортних засобів у програмній системі позиціювання в режимі реального часу» за спеціальністю
121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:
к.т.н., доц. кафедри ПЗ Рейда О.М.
"19" 09 2023 р.

Виконав:
студент гр.1ПІ-22м Гончар С.І.
"19" 09 2023 р.

Вінниця – 2023 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Дослідження методів оптимізації траєкторії руху транспортних засобів у програмній системі позиціювання в режимі реального часу».

Галузь застосування системи маршрутизації.

2. Підстава для розробки.

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ №247 від 18 вересня 2023 року ректора по ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки.

Метою даної роботи є підвищення рівня оптимізації траєкторії руху транспортних засобів у режимі реального часу

Призначення роботи – покращення методів оптимізації траєкторії руху транспортних засобів у режимі реального часу.

3 Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись МКР.

1. What is A* Search Algorithm?. [Електроний ресурс] URL: https://www.mygreatlearning.com/blog/a-search-algorithm-in-artificial-intelligence/.
2. Алгоритм Дейкстри [Електроний ресурс] URL: https://ua5.org/algorithm/1970-algorytm-dejkstry.html.
3. Bellman Ford’s Algorithm [Електроний ресурс] URL: https://www.programiz.com/dsa/bellman-ford-algorithm.

4. Griffiths D J, Mehdi Qasim H, Wang Tingkai and Gough E Norman. A GENETIC ALGORITHM FOR PATH PLANNING. School of Computing and Information Technology, University Of Wolverhampton, Wolverhampton.
5. Floyd Warshall Algorithm [Електроний ресурс] URL: https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/.
6. Comparative Analysis of Optimal Path Search Algorithms [Електроний ресурс] URL:researchgate.net/publication/347596106_Comparative_Analysis_of_Optimal_Path_Search_Algorithms.

4. Технічні вимоги

Середовище розробки – Visual Studio; мови розробки - Java, JavaScript; операційна система – Windows 10, macOS; веб-браузер – edge, chrome або safari; розподілена система управління версіями – Git; віртуальна машина Java.

5. Конструктивні вимоги.

Програмна система повинна бути безпечна, надійна, продуктивна і зручна у використанні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

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

– пояснювальна записка до МКР;

– технічне завдання;

– лістинги програми.

7. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

8. Стадії та етапи розробки:

№ з/п	Стан питання та постановка задач дослідження	Строк виконання етапів роботи
1	Аналіз моделей освітлення та вибір напрямків досліджень	20.09.2023 – 30.09.2023
2	Розробка структури системи	30.09.2023 – 15.10.2023
3	Програмна реалізація системи	15.10.2023 – 15.11.2023
4	Тестування системи	15.11.2023 – 20.11.2023
5	Економічна частина	20.11.2023 – 1.12.2023

9. Порядок контролю та прийняття.

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком

ДОДАТОК Б. Протокол перевірки роботи

103

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: Дослідження методів оптимізації траєкторії руху транспортних засобів у програмній системі позиціювання в режимі реального часу

Тип роботи: магістерська кваліфікаційна робота

Підрозділ : кафедра програмного забезпечення, ФІТКІ, 1ПІ – 22м

Науковий керівник: ктн., доц. кафедри ПЗ Рейда О.М.

Unicheck	
Оригінальність	**96.0**
Схожість	4.0

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

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

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

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Опис прийнятого рішення: **допустити до захисту**

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Гончар С.І.

Керівник роботи _____ Рейда О.М.

ДОДАТОК С. Лістинг програми

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.1.1</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>

  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mkr.sh</groupId>
  <artifactId>route-service</artifactId>
  <version>8.0-SNAPSHOT</version>
  <packaging>pom</packaging>
  <name>route-service</name>

  <modules>
    <module>route-engine</module>
    <module>route-api</module>
  </modules>

  <properties>
    <java.version>17</java.version>
    <maven.compiler.source>${java.version}</maven.compiler.source>
    <maven.compiler.target>${java.version}</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.timestamp>${maven.build.timestamp}</project.timestamp>
    <maven.build.timestamp.format>yyyy-MM-dd'T'HH:mm:ss'Z'</maven.build.timestamp.format>
    <springdoc-openapi-starter-webmvc-ui.version>2.1.0</springdoc-openapi-starter-webmvc-ui.version>
    <swagger-parser.version>2.1.16</swagger-parser.version>
    <snakeyaml.version>2.0</snakeyaml.version>
    <geotools.version>29.1</geotools.version>
    <slf4j.version>2.0.7</slf4j.version>
    <log4j.version>2.20.0</log4j.version>
    <fasterxml.jackson.version>2.14.2</fasterxml.jackson.version>
    <commons-io.version>2.11.0</commons-io.version>
    <typesafe-config.version>1.4.1</typesafe-config.version>
    <org-json.version>20231013</org-json.version>
    <postgresql.version>42.6.0</postgresql.version>
    <progressbar.version>0.9.5</progressbar.version>
    <jqwik.version>1.6.5</jqwik.version>
  </properties>

  <repositories>
    <repository>
      <id>heigit-nexus-public</id>
      <name>HeiGIT maven repositories</name>
      <url>https://repo.heigit.org/repository/maven-public/</url>
    </repository>
    <repository>
      <id>osgeo</id>
      <name>OSGeo Release Repository</name>
      <url>https://repo.osgeo.org/repository/release/</url>
      <snapshots>
```

```
        <enabled>false</enabled>
      </snapshots>
      <releases>
        <enabled>true</enabled>
      </releases>
    </repository>
</repositories>

<dependencyManagement>
  <dependencies>
    <!-- As long as swagger-parser uses snakeyaml <2 it cannot be used. Too much vulnerabilities. -->
    <!-- The package is updated quite often. -->
    <dependency>
      <groupId>io.swagger.parser.v3</groupId>
      <artifactId>swagger-parser</artifactId>
      <version>${swagger-parser.version}</version>
    </dependency>

    <dependency>
      <groupId>org.springdoc</groupId>
      <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
      <version>${springdoc-openapi-starter-webmvc-ui.version}</version>
    </dependency>

    <dependency>
      <groupId>org.apache.logging.log4j</groupId>
      <artifactId>log4j</artifactId>
      <type>pom</type>
      <version>${log4j.version}</version>
    </dependency>

    <dependency>
      <groupId>org.apache.logging.log4j</groupId>
      <artifactId>log4j-1.2-api</artifactId>
      <version>${log4j.version}</version>
    </dependency>

    <dependency>
      <groupId>org.apache.logging.log4j</groupId>
      <artifactId>log4j-api</artifactId>
      <version>${log4j.version}</version>
    </dependency>

    <dependency>
      <groupId>org.apache.logging.log4j</groupId>
      <artifactId>log4j-core</artifactId>
      <version>${log4j.version}</version>
    </dependency>

    <dependency>
      <groupId>commons-io</groupId>
      <artifactId>commons-io</artifactId>
      <version>${commons-io.version}</version>
    </dependency>
```

```
<dependency>
   <groupId>com.fasterxml.jackson.datatype</groupId>
   <artifactId>jackson-datatype-jsr310</artifactId>
   <version>${fasterxml.jackson.version}</version>
</dependency>

<dependency>
   <groupId>com.fasterxml.jackson.core</groupId>
   <artifactId>jackson-databind</artifactId>
   <version>${fasterxml.jackson.version}</version>
</dependency>

<dependency>
   <groupId>com.fasterxml.jackson.core</groupId>
   <artifactId>jackson-core</artifactId>
   <version>${fasterxml.jackson.version}</version>
</dependency>

<dependency>
   <groupId>com.fasterxml.jackson.core</groupId>
   <artifactId>jackson-annotations</artifactId>
   <version>${fasterxml.jackson.version}</version>
</dependency>

<dependency>
   <groupId>com.typesafe</groupId>
   <artifactId>config</artifactId>
   <version>${typesafe-config.version}</version>
</dependency>

<dependency>
   <groupId>me.tongfei</groupId>
   <artifactId>progressbar</artifactId>
   <version>${progressbar.version}</version>
</dependency>

<dependency>
   <groupId>net.jqwik</groupId>
   <artifactId>jqwik</artifactId>
   <version>${jqwik.version}</version>
</dependency>

<dependency>
   <groupId>org.geotools</groupId>
   <artifactId>gt-main</artifactId>
   <version>${geotools.version}</version>
</dependency>

<dependency>
   <groupId>org.geotools</groupId>
   <artifactId>gt-metadata</artifactId>
   <version>${geotools.version}</version>
</dependency>

<dependency>
```

```
            <groupId>org.geotools</groupId>
            <artifactId>gt-epsg-hsql</artifactId>
            <version>${geotools.version}</version>
        </dependency>

        <dependency>
            <groupId>org.geotools</groupId>
            <artifactId>gt-geojson</artifactId>
            <version>${geotools.version}</version>
        </dependency>

        <dependency>
            <groupId>org.geotools</groupId>
            <artifactId>gt-swing</artifactId>
            <version>${geotools.version}</version>
        </dependency>

        <dependency>
            <groupId>org.geotools</groupId>
            <artifactId>gt-shapefile</artifactId>
            <version>${geotools.version}</version>
        </dependency>

        <dependency>
            <groupId>org.json</groupId>
            <artifactId>json</artifactId>
            <version>${org-json.version}</version>
        </dependency>

        <!--                https://mvnrepository.com/artifact/org.postgresql/postgresql                -->
        <dependency>
            <groupId>org.postgresql</groupId>
            <artifactId>postgresql</artifactId>
            <version>${postgresql.version}</version>
        </dependency>

        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>${slf4j.version}</version>
        </dependency>

        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-reload4j</artifactId>
            <version>${slf4j.version}</version>
        </dependency>

    </dependencies>
</dependencyManagement>

<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
```

```
    <artifactId>maven-compiler-plugin</artifactId>
</plugin>

<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
    <configuration>
        <skip>true</skip>
    </configuration>
</plugin>

<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <configuration>
        <!--don't run API tests by default only through profile "apitests" -->
        <excludes>
            <exclude>**/apitests/**</exclude>
        </excludes>
        <!--suppress UnresolvedMavenProperty -->
        <argLine>-Duser.language=en -Duser.region=US -Dillegal-access=permit ${surefireArgLine}</argLine>
    </configuration>
</plugin>

<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <executions>
        <!-- prepare agent for measuring unit tests -->
        <execution>
            <id>prepare-unit-tests</id>
            <goals>
                <goal>prepare-agent</goal>
            </goals>
            <configuration>
                <propertyName>surefireArgLine</propertyName>
            </configuration>
        </execution>
    </executions>
</plugin>

<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-enforcer-plugin</artifactId>
    <executions>
        <execution>
            <id>enforce-maven</id>
            <goals>
                <goal>enforce</goal>
            </goals>
            <configuration>
                <rules>
                    <requireMavenVersion>
                        <version>3.6</version>
                    </requireMavenVersion>
```

```
                </rules>
              </configuration>
            </execution>
          </executions>
        </plugin>
      </plugins>
      <pluginManagement>
        <plugins>
          <!-- clean lifecycle, see https://maven.apache.org/ref/current/maven-core/lifecycles.html#clean_Lifecycle -->
          <plugin>
            <artifactId>maven-clean-plugin</artifactId>
            <version>3.1.0</version>
          </plugin>
          <plugin>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.10.1</version>
          </plugin>
          <plugin>
            <artifactId>maven-deploy-plugin</artifactId>
            <version>2.8.2</version>
          </plugin>
          <plugin>
            <artifactId>maven-enforcer-plugin</artifactId>
            <version>3.2.1</version>
          </plugin>
          <plugin>
            <artifactId>maven-gpg-plugin</artifactId>
            <version>3.0.1</version>
          </plugin>
          <plugin>
            <artifactId>maven-install-plugin</artifactId>
            <version>2.5.2</version>
          </plugin>
          <plugin>
            <artifactId>maven-jar-plugin</artifactId>
            <version>3.0.2</version>
          </plugin>
          <!-- default lifecycle, jar packaging: see https://maven.apache.org/ref/current/maven-core/default-
bindings.html#Plugin_bindings_for_jar_packaging                                                                -->
          <plugin>
            <artifactId>maven-resources-plugin</artifactId>
            <version>3.0.2</version>
          </plugin>
          <!-- site lifecycle, see https://maven.apache.org/ref/current/maven-core/lifecycles.html#site_Lifecycle -->
          <plugin>
            <artifactId>maven-site-plugin</artifactId>
            <version>3.7.1</version>
          </plugin>
          <plugin>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>2.22.2</version>
          </plugin>
          <plugin>
            <artifactId>maven-war-plugin</artifactId>
            <version>3.3.2</version>
```

```
        </plugin>
        <plugin>
          <groupId>org.jacoco</groupId>
          <artifactId>jacoco-maven-plugin</artifactId>
          <version>0.8.10</version>
        </plugin>
      </plugins>
    </pluginManagement>
  </build>
</project>
```

```
package com.mkr.sh.api.controllers;

import com.fasterxml.jackson.databind.exc.InvalidDefinitionException;
import com.fasterxml.jackson.databind.exc.InvalidFormatException;
import com.fasterxml.jackson.databind.exc.MismatchedInputException;
import com.fasterxml.jackson.databind.exc.UnrecognizedPropertyException;
import com.mkr.sh.exceptions.*;
import io.swagger.v3.oas.annotations.Parameter;
import jakarta.servlet.http.HttpServletResponse;
import com.mkr.sh.api.EndpointsProperties;
import com.mkr.sh.api.SystemMessageProperties;
import com.mkr.sh.api.errors.CommonResponseEntityExceptionHandler;
import com.mkr.sh.api.requests.routing.RouteRequest;
import com.mkr.sh.api.responses.routing.geojson.GeoJSONRouteResponse;
import com.mkr.sh.api.responses.routing.gpx.GPXRouteResponse;
import com.mkr.sh.api.responses.routing.json.JSONRouteResponse;
import com.mkr.sh.api.services.RoutingService;
import com.mkr.sh.api.util.AppConfigMigration;
import com.mkr.sh.routing.APIEnums;
import com.mkr.sh.routing.RouteResult;
import com.mkr.sh.routing.RoutingErrorCodes;
import org.locationtech.jts.geom.Coordinate;
import org.springframework.core.convert.ConversionFailedException;
import org.springframework.http.ResponseEntity;
import org.springframework.http.converter.HttpMessageConversionException;
import org.springframework.http.converter.HttpMessageNotReadableException;
import
org.springframework.web.bind.MissingServletRequestParameterException;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/v2/directions")
public class RoutingAPI {
    static final CommonResponseEntityExceptionHandler errorHandler = new
CommonResponseEntityExceptionHandler(RoutingErrorCodes.BASE);

    private final EndpointsProperties endpointsProperties;
    private final SystemMessageProperties systemMessageProperties;
    private final RoutingService routingService;

    public RoutingAPI(EndpointsProperties endpointsProperties,
SystemMessageProperties systemMessageProperties, RoutingService
routingService) {
        this.endpointsProperties =
```

```
AppConfigMigration.overrideEndpointsProperties(endpointsProperties);
        this.systemMessageProperties = systemMessageProperties;
        this.routingService = routingService;
    }


    @GetMapping
    public void getGetMapping() throws MissingParameterException {
        throw new
MissingParameterException(RoutingErrorCodes.MISSING_PARAMETER, "profile");
    }

    @PostMapping
    public String getPostMapping(@RequestBody RouteRequest request) throws
MissingParameterException {
        throw new
MissingParameterException(RoutingErrorCodes.MISSING_PARAMETER, "profile");
    }


    @PostMapping(value = "/{profile}/*")
    public void getInvalidResponseType() throws StatusCodeException {
        throw new
StatusCodeException(HttpServletResponse.SC_NOT_ACCEPTABLE,
RoutingErrorCodes.UNSUPPORTED_EXPORT_FORMAT, "This response format is not
supported");
    }

    @GetMapping(value = "/{profile}", produces =
{"application/geo+json;charset=UTF-8"})
    public GeoJSONRouteResponse
getSimpleGeoJsonRoute(@Parameter(description = "Specifies the route
profile.", required = true, example = "driving-car") @PathVariable
APIEnums.Profile profile,

@Parameter(description = "Start coordinate of the route in
`longitude,latitude` format.", required = true, example =
"8.681495,49.41461") @RequestParam Coordinate start,

@Parameter(description = "Destination coordinate of the route in
`longitude,latitude` format.", required = true, example =
"8.687872,49.420318") @RequestParam Coordinate end) throws
StatusCodeException {
        RouteRequest request = new RouteRequest(start, end);
        request.setProfile(profile);

        RouteResult[] result =
routingService.generateRouteFromRequest(request);

        return new GeoJSONRouteResponse(result, request,
systemMessageProperties, endpointsProperties);
    }

    @PostMapping(value = "/{profile}")
    public JSONRouteResponse getDefault(@Parameter(description =
"Specifies the route profile.", required = true, example = "driving-car")
@PathVariable APIEnums.Profile profile,
                                        @Parameter(description = "The
request payload", required = true) @RequestBody RouteRequest request)
throws StatusCodeException {
```

```
        return getJsonRoute(profile, request);
    }

    @PostMapping(value = "/{profile}/geojson", produces =
"application/geo+json;charset=UTF-8")
    public GeoJSONRouteResponse getGeoJsonRoute(
            @Parameter(description = "Specifies the route profile.",
required = true, example = "driving-car") @PathVariable APIEnums.Profile
profile,
            @Parameter(description = "The request payload", required =
true) @RequestBody RouteRequest request) throws StatusCodeException {
        request.setProfile(profile);
        request.setResponseType(APIEnums.RouteResponseType.GEOJSON);

        RouteResult[] result =
routingService.generateRouteFromRequest(request);

        return new GeoJSONRouteResponse(result, request,
systemMessageProperties, endpointsProperties);
    }

    @ExceptionHandler(MissingServletRequestParameterException.class)
    public ResponseEntity<Object> handleMissingParams(final
MissingServletRequestParameterException e) {
        return errorHandler.handleStatusCodeException(new
MissingParameterException(RoutingErrorCodes.MISSING_PARAMETER,
e.getParameterName()));
    }

    @ExceptionHandler({HttpMessageNotReadableException.class,
ConversionFailedException.class, HttpMessageConversionException.class,
Exception.class})
    public ResponseEntity<Object> handleReadingBodyException(final
Exception e) {
        final Throwable cause = e.getCause();
        if (cause instanceof UnrecognizedPropertyException exception) {
            return errorHandler.handleUnknownParameterException(new
UnknownParameterException(RoutingErrorCodes.UNKNOWN_PARAMETER,
exception.getPropertyName()));
        } else if (cause instanceof InvalidFormatException exception) {
            return errorHandler.handleStatusCodeException(new
ParameterValueException(RoutingErrorCodes.INVALID_PARAMETER_FORMAT, "" +
exception.getValue()));
        } else if (cause instanceof ConversionFailedException exception) {
            return errorHandler.handleStatusCodeException(new
ParameterValueException(RoutingErrorCodes.INVALID_PARAMETER_VALUE, "" +
exception.getValue()));
        } else if (cause instanceof InvalidDefinitionException exception)
{
            return errorHandler.handleStatusCodeException(new
ParameterValueException(RoutingErrorCodes.INVALID_PARAMETER_VALUE,
exception.getPath().get(0).getFieldName()));
        } else if (cause instanceof MismatchedInputException exception) {
            return errorHandler.handleStatusCodeException(new
ParameterValueException(RoutingErrorCodes.INVALID_PARAMETER_FORMAT,
exception.getPath().get(0).getFieldName()));
        } else {
            // Check if we are missing the body as a whole
            if (e.getLocalizedMessage().startsWith("Required request body
is missing")) {
```

```
                return errorHandler.handleStatusCodeException(new
EmptyElementException(RoutingErrorCodes.MISSING_PARAMETER, "Request body
could not be read"));
            }
            return errorHandler.handleGenericException(e);
        }
    }

    @ExceptionHandler(StatusCodeException.class)
    public ResponseEntity<Object> handleException(final
StatusCodeException e) {
        return errorHandler.handleStatusCodeException(e);
    }
}
```

```
package com.mkr.sh.routing;

import com.graphhopper.GHRequest;
import com.graphhopper.GHResponse;
import com.graphhopper.GraphHopper;
import com.graphhopper.config.CHProfile;
import com.graphhopper.config.LMProfile;
import com.graphhopper.config.Profile;
import com.graphhopper.gtfs.*;
import com.graphhopper.routing.util.AccessFilter;
import com.graphhopper.routing.util.EdgeFilter;
import com.graphhopper.routing.util.EncodingManager;
import com.graphhopper.routing.util.FlagEncoder;
import com.graphhopper.routing.weighting.Weighting;
import com.graphhopper.storage.*;
import com.graphhopper.storage.index.LocationIndex;
import com.graphhopper.util.*;
import com.graphhopper.util.shapes.BBox;
import com.graphhopper.util.shapes.GHPoint;
import com.mkr.sh.common.Pair;
import com.mkr.sh.exceptions.IncompatibleParameterException;
import com.mkr.sh.exceptions.InternalServerException;
import com.mkr.sh.exceptions.MaxVisitedNodesExceededException;
import com.mkr.sh.exceptions.PointNotFoundException;
import com.mkr.sh.export.ExportRequest;
import com.mkr.sh.export.ExportResult;
import com.mkr.sh.export.ExportWarning;
import com.mkr.sh.isochrones.*;
import com.mkr.sh.isochrones.statistics.StatisticsProvider;
import com.mkr.sh.isochrones.statistics.StatisticsProviderConfiguration;
import com.mkr.sh.isochrones.statistics.StatisticsProviderFactory;
import com.mkr.sh.matrix.*;
import com.mkr.sh.matrix.algorithms.core.CoreMatrixAlgorithm;
import com.mkr.sh.matrix.algorithms.dijkstra.DijkstraMatrixAlgorithm;
import com.mkr.sh.matrix.algorithms.rphast.RPHASTMatrixAlgorithm;
import com.mkr.sh.routing.graphhopper.extensions.*;
import
com.mkr.sh.routing.graphhopper.extensions.flagencoders.FlagEncoderNames;
import com.mkr.sh.routing.pathprocessors.ORSPathProcessorFactory;
import com.mkr.sh.isochrones.*;
import com.mkr.sh.matrix.*;
import com.mkr.sh.routing.graphhopper.extensions.util.ORSParameters;
import com.typesafe.config.Config;
import org.apache.log4j.Logger;
import com.mkr.sh.config.EngineConfig;
import com.mkr.sh.routing.configuration.RouteProfileConfiguration;
import com.mkr.sh.routing.graphhopper.extensions.*;
import
com.mkr.sh.routing.graphhopper.extensions.storages.GraphStorageUtils;
import
com.mkr.sh.routing.graphhopper.extensions.storages.OsmIdGraphStorage;
import
com.mkr.sh.routing.graphhopper.extensions.storages.WheelchairAttributesGra
phStorage;
import
com.mkr.sh.routing.graphhopper.extensions.storages.builders.BordersGraphSt
orageBuilder;
import
com.mkr.sh.routing.graphhopper.extensions.storages.builders.GraphStorageBu
```

```
ilder;
import com.mkr.sh.routing.parameters.ProfileParameters;
import com.mkr.sh.util.DebugUtility;
import com.mkr.sh.util.ProfileTools;
import com.mkr.sh.util.StringUtility;
import com.mkr.sh.util.TimeUtility;
import org.locationtech.jts.geom.Coordinate;

import java.io.File;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.text.SimpleDateFormat;
import java.time.*;
import java.util.*;


public class RoutingProfile {
    private static final Logger LOGGER =
Logger.getLogger(RoutingProfile.class);
    private static final Object lockObj = new Object();
    private static int profileIdentifier = 0;
    private final Integer[] mRoutePrefs;
    private final RouteProfileConfiguration config;
    private final ORSGraphHopper mGraphHopper;
    private Integer mUseCounter;
    private String astarApproximation;
    private Double astarEpsilon;

    public RoutingProfile(EngineConfig engineConfig,
RouteProfileConfiguration rpc, RoutingProfileLoadContext loadCntx) throws
Exception {
        mRoutePrefs = rpc.getProfilesTypes();
        mUseCounter = 0;

        mGraphHopper = initGraphHopper(engineConfig, rpc, loadCntx);

        config = rpc;

        Config optsExecute = config.getExecutionOpts();
        if (optsExecute != null) {
            if (optsExecute.hasPath("methods.astar.approximation"))
                astarApproximation =
optsExecute.getString("methods.astar.approximation");
            if (optsExecute.hasPath("methods.astar.epsilon"))
                astarEpsilon =
Double.parseDouble(optsExecute.getString("methods.astar.epsilon"));
        }
    }

    public static ORSGraphHopper initGraphHopper(EngineConfig
engineConfig, RouteProfileConfiguration config, RoutingProfileLoadContext
loadCntx) throws Exception {
        String osmFile = engineConfig.getSourceFile();
        ORSGraphHopperConfig args = createGHSettings(osmFile, config);

        int profileId;
        synchronized (lockObj) {
            profileIdentifier++;
            profileId = profileIdentifier;
```

```
        }

        long startTime = System.currentTimeMillis();

        if (LOGGER.isInfoEnabled()) {
            LOGGER.info("[%d] Profiles: '%s', location:
'%s'.".formatted(profileId, config.getProfiles(), config.getGraphPath()));
        }

        GraphProcessContext gpc = new GraphProcessContext(config);

gpc.setGetElevationFromPreprocessedData(engineConfig.isElevationPreprocess
ed());

        ORSGraphHopper gh = new ORSGraphHopper(gpc);

        ORSDefaultFlagEncoderFactory flagEncoderFactory = new
ORSDefaultFlagEncoderFactory();
        gh.setFlagEncoderFactory(flagEncoderFactory);

        ORSPathProcessorFactory pathProcessorFactory = new
ORSPathProcessorFactory();
        gh.setPathProcessorFactory(pathProcessorFactory);

        gh.init(args);




        if (loadCntx.getElevationProvider() != null) {
            if (args.has("graph.elevation.provider")) {
                gh.setElevationProvider(loadCntx.getElevationProvider());
            }
        } else {
            loadCntx.setElevationProvider(gh.getElevationProvider());
        }
        gh.setGraphStorageFactory(new
ORSGraphStorageFactory(gpc.getStorageBuilders()));

        gh.importOrLoad();

        for (GraphStorageBuilder builder : gpc.getStorageBuilders()) {
            if
(builder.getName().equals(BordersGraphStorageBuilder.BUILDER_NAME)) {

pathProcessorFactory.setCountryBordersReader(((BordersGraphStorageBuilder)
builder).getCbReader());
            }
        }

        if (LOGGER.isInfoEnabled()) {
            GraphHopperStorage ghStorage = gh.getGraphHopperStorage();
            LOGGER.info("[%d] Edges: %s - Nodes: %s.".formatted(profileId,
ghStorage.getEdges(), ghStorage.getNodes()));
            LOGGER.info("[%d] Total time: %s.".formatted(profileId,
TimeUtility.getElapsedTime(startTime, true)));
            LOGGER.info("[%d] Finished at: %s.".formatted(profileId, new
SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date())));
        }
```

```
        File file = new File(osmFile);
        Path pathTimestamp = Paths.get(config.getGraphPath(),
"stamp.txt");
        File file2 = pathTimestamp.toFile();
        if (!file2.exists())
            Files.write(pathTimestamp,
Long.toString(file.length()).getBytes());

        return gh;
    }

    private static ORSGraphHopperConfig createGHSettings(String
sourceFile, RouteProfileConfiguration config) {
        ORSGraphHopperConfig ghConfig = new ORSGraphHopperConfig();
        ghConfig.putObject("graph.dataaccess", "RAM_STORE");
        ghConfig.putObject("datareader.file", sourceFile);
        ghConfig.putObject("graph.location", config.getGraphPath());
        ghConfig.putObject("graph.bytes_for_flags",
config.getEncoderFlagsSize());

        if (!config.getInstructions())
            ghConfig.putObject("instructions", false);
        if (config.getElevationProvider() != null &&
config.getElevationCachePath() != null) {
            ghConfig.putObject("graph.elevation.provider",
StringUtility.trimQuotes(config.getElevationProvider()));
            ghConfig.putObject("graph.elevation.cache_dir",
StringUtility.trimQuotes(config.getElevationCachePath()));
            ghConfig.putObject("graph.elevation.dataaccess",
StringUtility.trimQuotes(config.getElevationDataAccess()));
            ghConfig.putObject("graph.elevation.clear",
config.getElevationCacheClear());
            if (config.getInterpolateBridgesAndTunnels())
                ghConfig.putObject("graph.encoded_values",
"road_environment");
            if (config.getElevationSmoothing())
                ghConfig.putObject("graph.elevation.smoothing", true);
        }

        boolean prepareCH = false;
        boolean prepareLM = false;
        boolean prepareCore = false;
        boolean prepareFI = false;

        Integer[] profilesTypes = config.getProfilesTypes();
        Map<String, Profile> profiles = new LinkedHashMap<>();




        if (profilesTypes.length != 1)
            throw new IllegalStateException("Expected single profile in
config");

        String vehicle =
RoutingProfileType.getEncoderName(profilesTypes[0]);

        boolean hasTurnCosts = config.isTurnCostEnabled();
```

```
        String[] weightings = {ProfileTools.VAL_FASTEST,
ProfileTools.VAL_SHORTEST, ProfileTools.VAL_RECOMMENDED};
        for (String weighting : weightings) {
            if (hasTurnCosts) {
                String profileName = ProfileTools.makeProfileName(vehicle,
weighting, true);
                profiles.put(profileName, new
Profile(profileName).setVehicle(vehicle).setWeighting(weighting).setTurnCo
sts(true));
            }
            String profileName = ProfileTools.makeProfileName(vehicle,
weighting, false);
            profiles.put(profileName, new
Profile(profileName).setVehicle(vehicle).setWeighting(weighting).setTurnCo
sts(false));
        }

        ghConfig.putObject(ProfileTools.KEY_PREPARE_CORE_WEIGHTINGS,
"no");

        if (config.getIsochronePreparationOpts() != null) {
            Config fastisochroneOpts =
config.getIsochronePreparationOpts();
            prepareFI = true;
            if (fastisochroneOpts.hasPath(ProfileTools.KEY_ENABLED) ||
fastisochroneOpts.getBoolean(ProfileTools.KEY_ENABLED)) {
                prepareFI =
fastisochroneOpts.getBoolean(ProfileTools.KEY_ENABLED);
                if (!prepareFI)

ghConfig.putObject(ProfileTools.KEY_PREPARE_FASTISOCHRONE_WEIGHTINGS,
"no");
                else

ghConfig.putObject(ORSParameters.FastIsochrone.PROFILE,
config.getProfiles());
            }

            if (prepareFI) {

                if (fastisochroneOpts.hasPath(ProfileTools.KEY_THREADS))
                    ghConfig.putObject("prepare.fastisochrone.threads",
fastisochroneOpts.getInt(ProfileTools.KEY_THREADS));
                if
(fastisochroneOpts.hasPath(ProfileTools.KEY_MAXCELLNODES))

ghConfig.putObject("prepare.fastisochrone.maxcellnodes",
StringUtility.trimQuotes(fastisochroneOpts.getString(ProfileTools.KEY_MAXC
ELLNODES)));
                if
(fastisochroneOpts.hasPath(ProfileTools.KEY_WEIGHTINGS)) {
                    List<Profile> fastisochronesProfiles = new
ArrayList<>();
                    String fastisochronesWeightingsString =
StringUtility.trimQuotes(fastisochroneOpts.getString(ProfileTools.KEY_WEIG
HTINGS));
                    for (String weighting :
fastisochronesWeightingsString.split(",")) {
```

```
                        String configStr = "";
                        weighting = weighting.trim();
                        if (weighting.contains("|")) {
                            configStr = weighting;
                            weighting = weighting.split("\\|")[0];
                        }
                        PMap configMap = new PMap(configStr);
                        boolean considerTurnRestrictions =
configMap.getBool("edge_based", hasTurnCosts);

                        String profileName =
ProfileTools.makeProfileName(vehicle, weighting,
considerTurnRestrictions);
                        Profile profile = new
Profile(profileName).setVehicle(vehicle).setWeighting(weighting).setTurnCo
sts(considerTurnRestrictions);
                        profiles.put(profileName, profile);
                        fastisochronesProfiles.add(profile);
                    }

ghConfig.setFastisochroneProfiles(fastisochronesProfiles);
                }
            }
        }

        if (config.getPreparationOpts() != null) {
            Config opts = config.getPreparationOpts();
            if (opts.hasPath("min_network_size"))
                ghConfig.putObject("prepare.min_network_size",
opts.getInt("min_network_size"));
            if (opts.hasPath("min_one_way_network_size"))
                ghConfig.putObject("prepare.min_one_way_network_size",
opts.getInt("min_one_way_network_size"));

            if (opts.hasPath("methods")) {
                if (opts.hasPath(ProfileTools.KEY_METHODS_CH)) {
                    prepareCH = true;
                    Config chOpts =
opts.getConfig(ProfileTools.KEY_METHODS_CH);

                    if (chOpts.hasPath(ProfileTools.KEY_ENABLED) ||
chOpts.getBoolean(ProfileTools.KEY_ENABLED)) {
                        prepareCH =
chOpts.getBoolean(ProfileTools.KEY_ENABLED);
                    }

                    if (prepareCH) {
                        if (chOpts.hasPath(ProfileTools.KEY_THREADS))
                            ghConfig.putObject("prepare.ch.threads",
chOpts.getInt(ProfileTools.KEY_THREADS));
                        if (chOpts.hasPath(ProfileTools.KEY_WEIGHTINGS)) {
                            List<CHProfile> chProfiles = new
ArrayList<>();
                            String chWeightingsString =
StringUtility.trimQuotes(chOpts.getString(ProfileTools.KEY_WEIGHTINGS));
                            for (String weighting :
chWeightingsString.split(","))
                                chProfiles.add(new
CHProfile(ProfileTools.makeProfileName(vehicle, weighting, false)));
                            ghConfig.setCHProfiles(chProfiles);
```

```
                        }
                    }
                }

                if (opts.hasPath(ProfileTools.KEY_METHODS_LM)) {
                    prepareLM = true;
                    Config lmOpts =
opts.getConfig(ProfileTools.KEY_METHODS_LM);

                    if (lmOpts.hasPath(ProfileTools.KEY_ENABLED) ||
lmOpts.getBoolean(ProfileTools.KEY_ENABLED)) {
                        prepareLM =
lmOpts.getBoolean(ProfileTools.KEY_ENABLED);
                    }

                    if (prepareLM) {
                        if (lmOpts.hasPath(ProfileTools.KEY_THREADS))
                            ghConfig.putObject("prepare.lm.threads",
lmOpts.getInt(ProfileTools.KEY_THREADS));
                        if (lmOpts.hasPath(ProfileTools.KEY_WEIGHTINGS)) {
                            List<LMProfile> lmProfiles = new
ArrayList<>();
                            String lmWeightingsString =
StringUtility.trimQuotes(lmOpts.getString(ProfileTools.KEY_WEIGHTINGS));
                            for (String weighting :
lmWeightingsString.split(","))
                                lmProfiles.add(new
LMProfile(ProfileTools.makeProfileName(vehicle, weighting,
hasTurnCosts)));
                            ghConfig.setLMProfiles(lmProfiles);
                        }
                        if (lmOpts.hasPath(ProfileTools.KEY_LANDMARKS))
                            ghConfig.putObject("prepare.lm.landmarks",
lmOpts.getInt(ProfileTools.KEY_LANDMARKS));
                    }
                }

                if (opts.hasPath(ProfileTools.KEY_METHODS_CORE)) {
                    prepareCore = true;
                    Config coreOpts =
opts.getConfig(ProfileTools.KEY_METHODS_CORE);

                    if (coreOpts.hasPath(ProfileTools.KEY_ENABLED) ||
coreOpts.getBoolean(ProfileTools.KEY_ENABLED)) {
                        prepareCore =
coreOpts.getBoolean(ProfileTools.KEY_ENABLED);
                        if (!prepareCore)

ghConfig.putObject(ProfileTools.KEY_PREPARE_CORE_WEIGHTINGS, "no");
                    }

                    if (prepareCore) {
                        if (coreOpts.hasPath(ProfileTools.KEY_THREADS)) {
                            String[] threads =
coreOpts.getString(ProfileTools.KEY_THREADS).split(",");
                            int threadsCH = Integer.parseInt(threads[0]);
                            int threadsLM = threads.length > 1 ?
Integer.parseInt(threads[1]) : threadsCH;
                            ghConfig.putObject("prepare.core.threads",
threadsCH);
```

```
                            ghConfig.putObject("prepare.corelm.threads",
threadsLM);
                        }
                        if (coreOpts.hasPath(ProfileTools.KEY_WEIGHTINGS))
{
                            List<CHProfile> coreProfiles = new
ArrayList<>();
                            List<LMProfile> coreLMProfiles = new
ArrayList<>();
                            String coreWeightingsString =
StringUtility.trimQuotes(coreOpts.getString(ProfileTools.KEY_WEIGHTINGS));
                            for (String weighting :
coreWeightingsString.split(",")) {
                                String configStr = "";
                                if (weighting.contains("|")) {
                                    configStr = weighting;
                                    weighting = weighting.split("\\|")[0];
                                }
                                PMap configMap = new PMap(configStr);
                                boolean considerTurnRestrictions =
configMap.getBool("edge_based", hasTurnCosts);

                                String profileName =
ProfileTools.makeProfileName(vehicle, weighting,
considerTurnRestrictions);
                                profiles.put(profileName, new
Profile(profileName).setVehicle(vehicle).setWeighting(weighting).setTurnCo
sts(considerTurnRestrictions));
                                coreProfiles.add(new
CHProfile(profileName));
                                coreLMProfiles.add(new
LMProfile(profileName));
                            }
                            ghConfig.setCoreProfiles(coreProfiles);
                            ghConfig.setCoreLMProfiles(coreLMProfiles);
                        }
                        if (coreOpts.hasPath(ProfileTools.KEY_LMSETS))
                            ghConfig.putObject("prepare.corelm.lmsets",
StringUtility.trimQuotes(coreOpts.getString(ProfileTools.KEY_LMSETS)));
                        if (coreOpts.hasPath(ProfileTools.KEY_LANDMARKS))
                            ghConfig.putObject("prepare.corelm.landmarks",
coreOpts.getInt(ProfileTools.KEY_LANDMARKS));
                    }
                }
            }
        }

        if (config.getExecutionOpts() != null) {
            Config opts = config.getExecutionOpts();
            if (opts.hasPath(ProfileTools.KEY_METHODS_CORE)) {
                Config coreOpts =
opts.getConfig(ProfileTools.KEY_METHODS_CORE);
                if (coreOpts.hasPath(ProfileTools.KEY_DISABLING_ALLOWED))
                    ghConfig.putObject("routing.core.disabling_allowed",
coreOpts.getBoolean(ProfileTools.KEY_DISABLING_ALLOWED));

                if (coreOpts.hasPath(ProfileTools.KEY_ACTIVE_LANDMARKS))
                    ghConfig.putObject("routing.corelm.active_landmarks",
coreOpts.getInt(ProfileTools.KEY_ACTIVE_LANDMARKS));
            }
```

```
            if (opts.hasPath(ProfileTools.KEY_METHODS_LM)) {
                Config lmOpts =
opts.getConfig(ProfileTools.KEY_METHODS_LM);
                if (lmOpts.hasPath(ProfileTools.KEY_ACTIVE_LANDMARKS))
                    ghConfig.putObject("routing.lm.active_landmarks",
lmOpts.getInt(ProfileTools.KEY_ACTIVE_LANDMARKS));
            }
        }

        if (config.getOptimize() && !prepareCH)
            ghConfig.putObject("graph.do_sort", true);

        if (!config.getGtfsFile().isEmpty())
            ghConfig.putObject("gtfs.file", config.getGtfsFile());

        String flagEncoder = vehicle;
        if (!Helper.isEmpty(config.getEncoderOptions()))
            flagEncoder += "|" + config.getEncoderOptions();

        ghConfig.putObject("graph.flag_encoders",
flagEncoder.toLowerCase());
        ghConfig.putObject("index.high_resolution",
config.getLocationIndexResolution());
        ghConfig.putObject("index.max_region_search",
config.getLocationIndexSearchIterations());
        ghConfig.setProfiles(new ArrayList<>(profiles.values()));

        return ghConfig;
    }

    private static boolean supportWeightingMethod(int profileType) {
        return RoutingProfileType.isDriving(profileType) ||
RoutingProfileType.isCycling(profileType) ||
RoutingProfileType.isPedestrian(profileType);
    }

    private boolean hasCHProfile(String profileName) {
        boolean hasCHProfile = false;
        for (CHProfile chProfile :
getGraphhopper().getCHPreparationHandler().getCHProfiles()) {
            if (profileName.equals(chProfile.getProfile()))
                hasCHProfile = true;
        }
        return hasCHProfile;
    }

    private boolean hasCoreProfile(String profileName) {
        boolean hasCoreProfile = false;
        for (CHProfile chProfile :
getGraphhopper().getCorePreparationHandler().getCHProfiles()) {
            if (profileName.equals(chProfile.getProfile()))
                hasCoreProfile = true;
        }
        return hasCoreProfile;
    }

    public long getMemoryUsage() {
        return mGraphHopper.getMemoryUsage();
    }
```

```
    public ORSGraphHopper getGraphhopper() {
        return mGraphHopper;
    }

    public BBox getBounds() {
        return mGraphHopper.getGraphHopperStorage().getBounds();
    }

    public StorableProperties getGraphProperties() {
        return mGraphHopper.getGraphHopperStorage().getProperties();
    }

    public RouteProfileConfiguration getConfiguration() {
        return config;
    }

    public Integer[] getPreferences() {
        return mRoutePrefs;
    }

    public boolean hasCarPreferences() {
        for (Integer mRoutePref : mRoutePrefs) {
            if (RoutingProfileType.isDriving(mRoutePref))
                return true;
        }
        return false;
    }

    public boolean isCHEnabled() {
        return mGraphHopper != null &&
mGraphHopper.getCHPreparationHandler().isEnabled();
    }

    public void close() {
        mGraphHopper.close();
    }

    private synchronized boolean isGHUsed() {
        return mUseCounter > 0;
    }

    private synchronized void beginUseGH() {
        mUseCounter++;
    }

    private synchronized void endUseGH() {
        mUseCounter--;
    }


    public IsochroneMap buildIsochrone(IsochroneSearchParameters
parameters, String[] attributes) throws Exception {

        String[] tempAttributes;
        if
(Arrays.toString(attributes).contains(ProfileTools.KEY_TOTAL_POP.toLowerCa
se()) &&
!(Arrays.toString(attributes).contains(ProfileTools.KEY_TOTAL_AREA_KM.toLo
werCase()))) {
            tempAttributes = new String[attributes.length + 1];
```

```
            int i = 0;
            while (i < attributes.length) {
                String attribute = attributes[i];
                tempAttributes[i] = attribute;
                i++;
            }
            tempAttributes[i] = ProfileTools.KEY_TOTAL_AREA_KM;
        } else if
((Arrays.toString(attributes).contains(ProfileTools.KEY_TOTAL_AREA_KM.toLo
werCase())) &&
(!Arrays.toString(attributes).contains(ProfileTools.KEY_TOTAL_POP.toLowerC
ase()))) {
            tempAttributes = new String[attributes.length + 1];
            int i = 0;
            while (i < attributes.length) {
                String attribute = attributes[i];
                tempAttributes[i] = attribute;
                i++;
            }
            tempAttributes[i] = ProfileTools.KEY_TOTAL_POP;
        } else {
            tempAttributes = attributes;
        }


        IsochroneMap result;

        beginUseGH();

        try {
            RouteSearchContext searchCntx =
createSearchContext(parameters.getRouteParameters());

            IsochroneMapBuilderFactory isochroneMapBuilderFactory = new
IsochroneMapBuilderFactory(searchCntx);
            result = isochroneMapBuilderFactory.buildMap(parameters);

            endUseGH();
        } catch (Exception ex) {
            endUseGH();
            if (DebugUtility.isDebug()) {
                LOGGER.error(ex);
            }
            throw new
InternalServerException(IsochronesErrorCodes.UNKNOWN, "Unable to build an
isochrone map.");
        }

        if (tempAttributes != null && result.getIsochronesCount() > 0) {
            try {
                Map<StatisticsProviderConfiguration, List<String>>
mapProviderToAttrs = new HashMap<>();
                for (String attr : tempAttributes) {
                    StatisticsProviderConfiguration provConfig =
parameters.getStatsProviders().get(attr);

                    if (provConfig != null) {
                        if (mapProviderToAttrs.containsKey(provConfig)) {
                            List<String> attrList =
mapProviderToAttrs.get(provConfig);
```

```
                            attrList.add(attr);
                        } else {
                            List<String> attrList = new ArrayList<>();
                            attrList.add(attr);
                            mapProviderToAttrs.put(provConfig, attrList);
                        }
                    }
                }

                for (Map.Entry<StatisticsProviderConfiguration,
List<String>> entry : mapProviderToAttrs.entrySet()) {
                    StatisticsProviderConfiguration provConfig =
entry.getKey();
                    StatisticsProvider provider =
StatisticsProviderFactory.getProvider(provConfig.getName(),
provConfig.getParameters());
                    String[] provAttrs =
provConfig.getMappedProperties(entry.getValue());

                    for (Isochrone isochrone : result.getIsochrones()) {
                        double[] attrValues =
provider.getStatistics(isochrone, provAttrs);
                        isochrone.setAttributes(entry.getValue(),
attrValues, provConfig.getAttribution());
                    }
                }

            } catch (Exception ex) {
                if (DebugUtility.isDebug()) {
                    LOGGER.error(ex);
                }
                throw new
InternalServerException(IsochronesErrorCodes.UNKNOWN, "Unable to compute
isochrone attributes.");
            }
        }

        return result;
    }


    public MatrixResult computeMatrix(MatrixRequest req) throws Exception
{
        GraphHopper gh = getGraphhopper();
        String encoderName =
RoutingProfileType.getEncoderName(req.getProfileType());
        FlagEncoder flagEncoder =
gh.getEncodingManager().getEncoder(encoderName);
        PMap hintsMap = new PMap();
        int weightingMethod = req.getWeightingMethod() ==
WeightingMethod.UNKNOWN ? WeightingMethod.RECOMMENDED :
req.getWeightingMethod();
        ProfileTools.setWeightingMethod(hintsMap, weightingMethod,
req.getProfileType(), false);
        ProfileTools.setWeighting(hintsMap, weightingMethod,
req.getProfileType(), false);
        String CHProfileName = ProfileTools.makeProfileName(encoderName,
hintsMap.getString("weighting", ""), false);
        String CoreProfileName = ProfileTools.makeProfileName(encoderName,
hintsMap.getString("weighting", ""), true);
```

```
        try {

            if (!req.getFlexibleMode() &&
gh.getCHPreparationHandler().isEnabled() && hasCHProfile(CHProfileName)) {
                return computeRPHASTMatrix(req, gh, flagEncoder,
CHProfileName);
            }

            else if (req.getSearchParameters().getDynamicSpeeds() &&
mGraphHopper.isCoreAvailable(CoreProfileName)) {
                return computeCoreMatrix(req, gh, flagEncoder, hintsMap,
CoreProfileName);
            }

            else {

                return computeDijkstraMatrix(req, gh, flagEncoder,
hintsMap, CHProfileName);
            }
        } catch (PointNotFoundException e) {
            throw e;
        } catch (MaxVisitedNodesExceededException e) {
            throw new
InternalServerException(MatrixErrorCodes.MAX_VISITED_NODES_EXCEEDED,
"Unable to compute a distance/duration matrix: " + e.getMessage());
        } catch (Exception ex) {
            throw new InternalServerException(MatrixErrorCodes.UNKNOWN,
"Unable to compute a distance/duration matrix: " + ex.getMessage());
        }
    }
```

ДОДАТОК Г. Ілюстративна частина

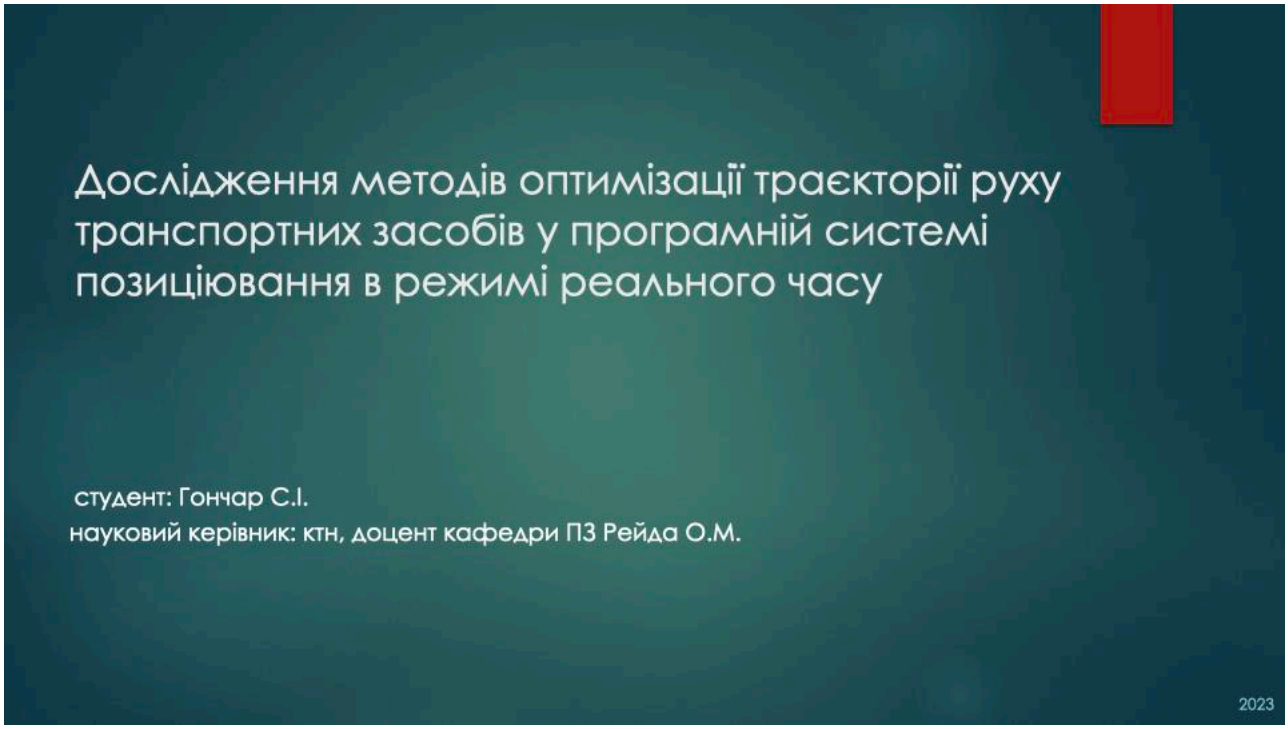

Рисунок Г.1 – Титульний слайд

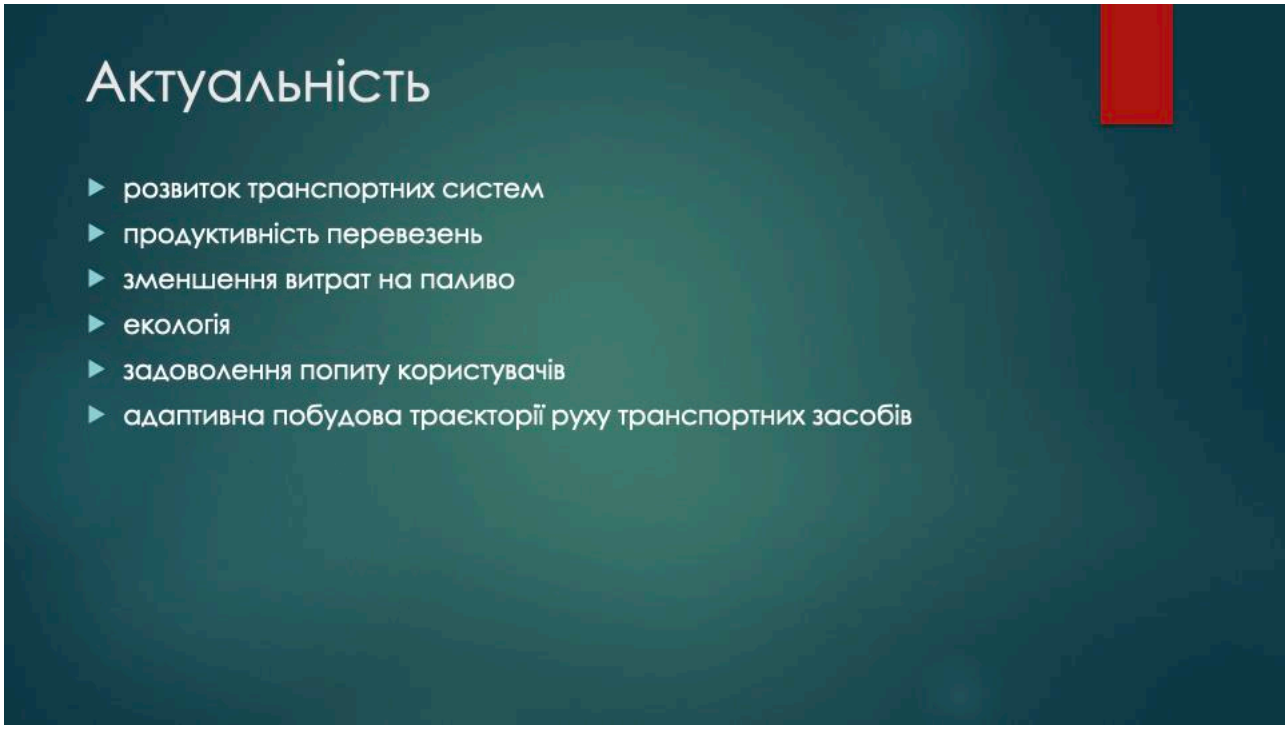

Рисунок Г.2 – Слайд присвячений актуальності дослідження

Рисунок Г.3 – Слайд «Мета та завдання дослідження»

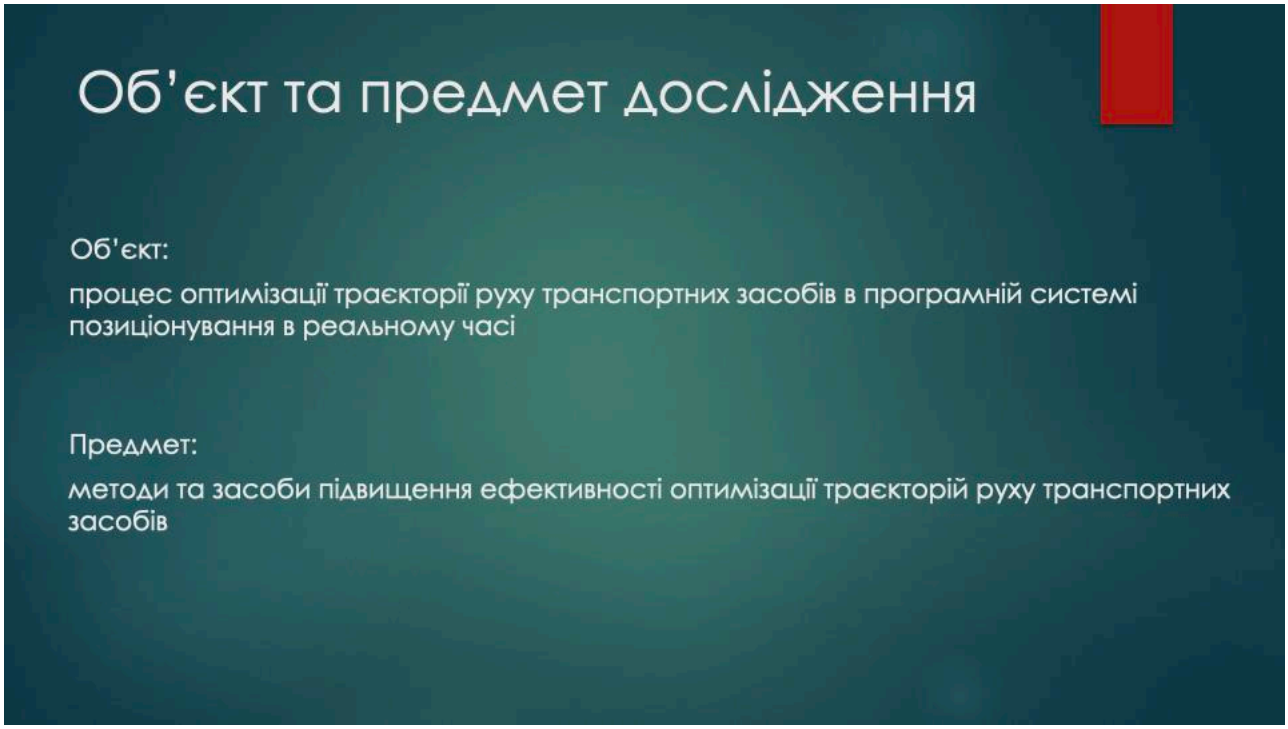

Рисунок Г.4 – Слайд «Об’єкт та предмет дослідження»

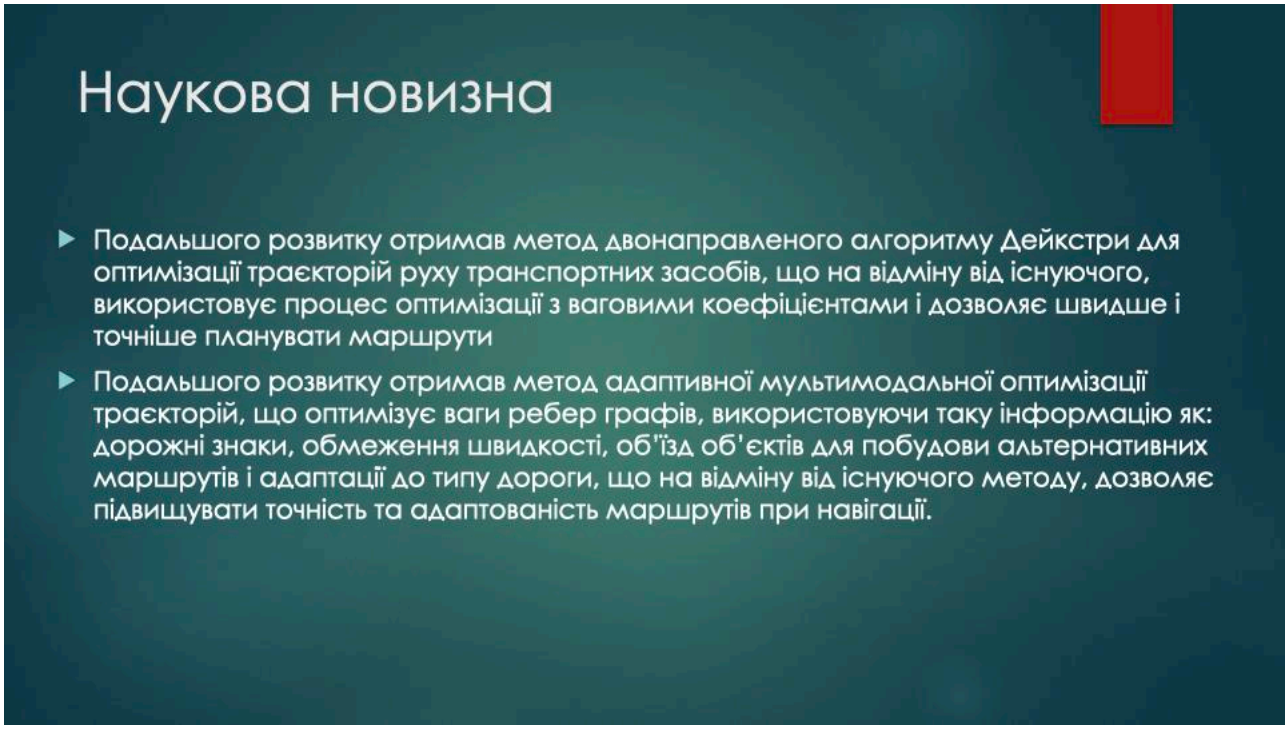

Рисунок Г.5 – Слайд «Наукова новизна»

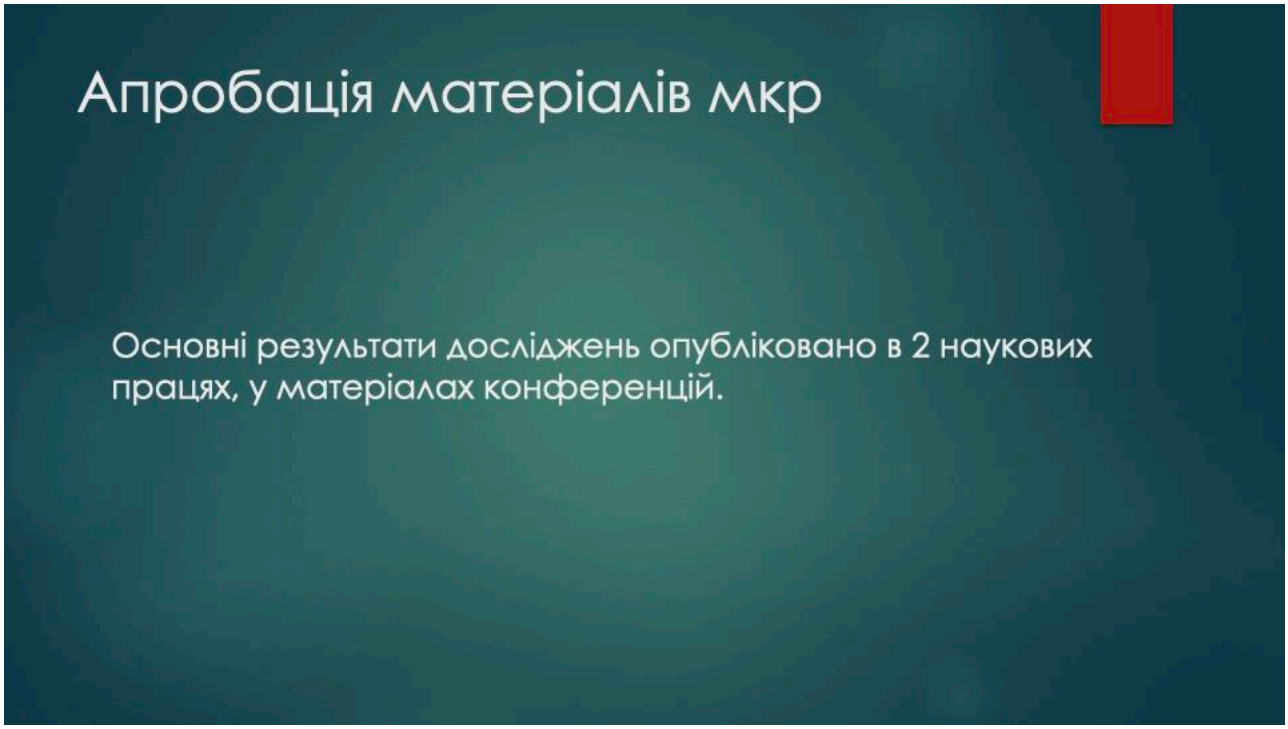

Рисунок Г.6 – Слайд «Апробація матеріалів мкр»

Розгляд аналогів

Критерій / Алгоритм	«A*»	Дейкстри	Беллмана Форда	Генетичні	Флойда Уоршалла	ШІ
Швидкість виконання	8	9	6	5	3	5
Точність	9	9	9	9	9	9
Складність Реалізації	7	9	5	3	4	2
Робота в динамічному середовищі	9	7	7	9	1	8
Підтримка великих мереж	9	9	9	9	9	9
Використання додаткових даних	6	6	2	9	2	9
Сумарний коефіцієнт	48	49	38	44	28	42

Рисунок Г.7 – Слайд «Об’єкт та предмет дослідження»

Метод Дейкстри
A -> C

- вузол A - знаходження ребра з min вагою до B, F, E
- A -> F
- вузол F - знаходження ребра з min вагою до E, B, D
- A -> F -> B
- вузол B - знаходження ребра з min вагою до C
- Оптимальний шлях знайдено A -> F -> B -> C

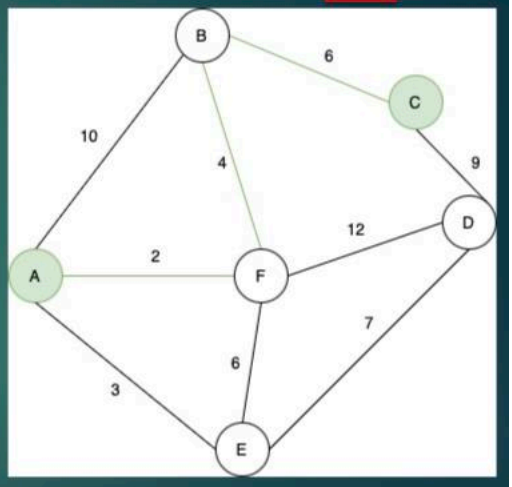

Рисунок Г.8 – Слайд «Метод Дейкстри»

Рисунок Г.9 – Слайд «Днонаправлений метод Дейкстри»

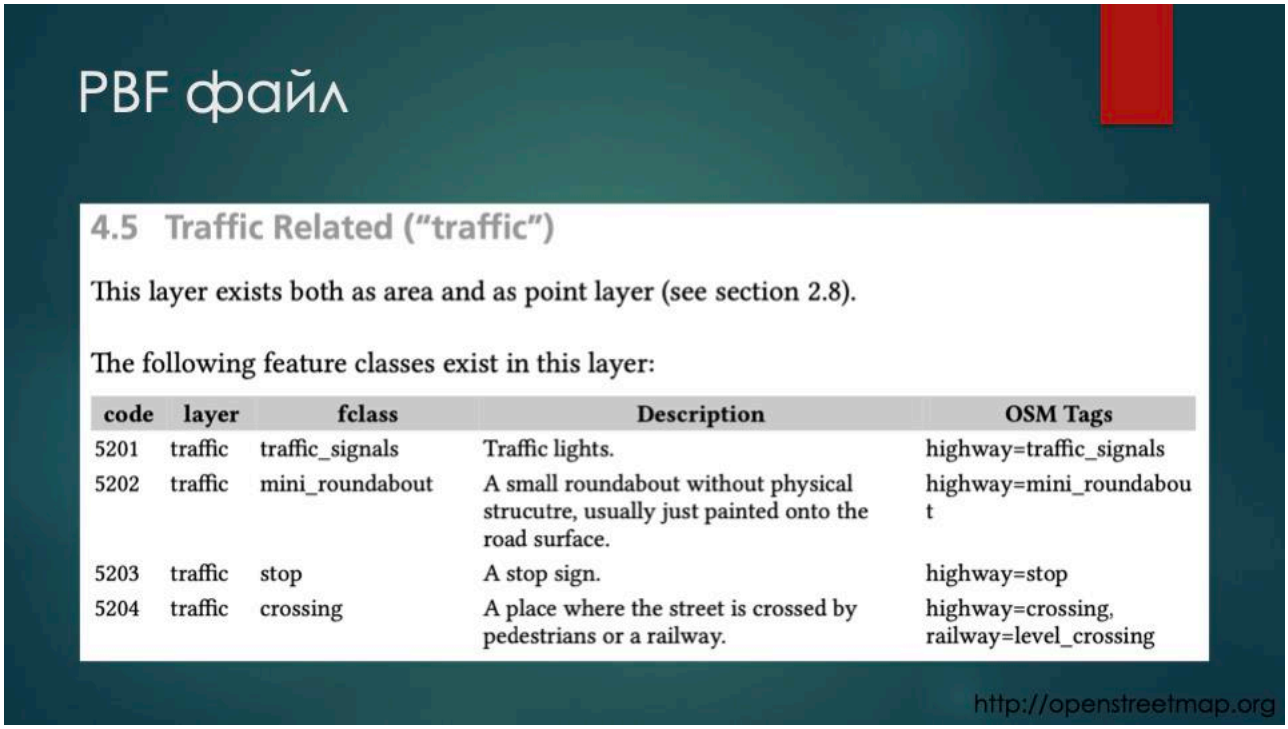

Рисунок Г.10 – Слайд «PBF файл»

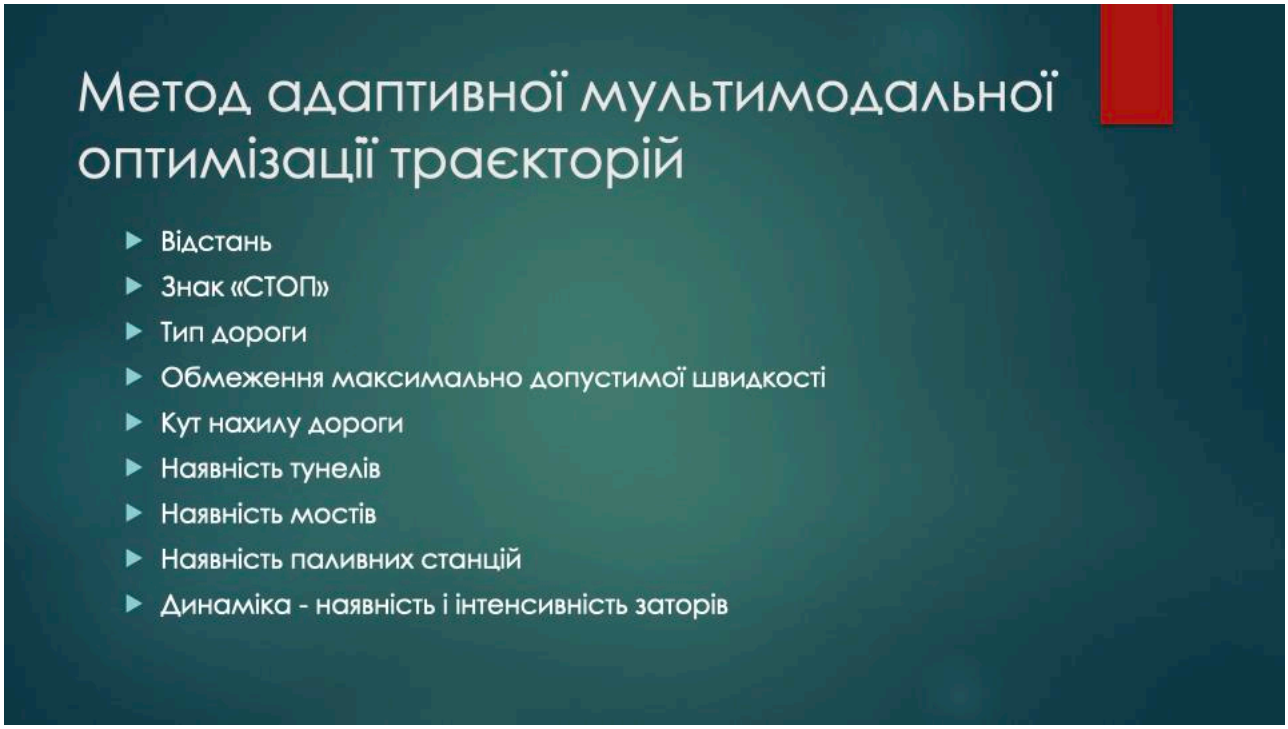

Рисунок Г.11 – Слайд «Метод адаптивної мультимодальної оптимізації траєкторій»

Рисунок Г.12 – Слайд «Контекстна діаграма системи»

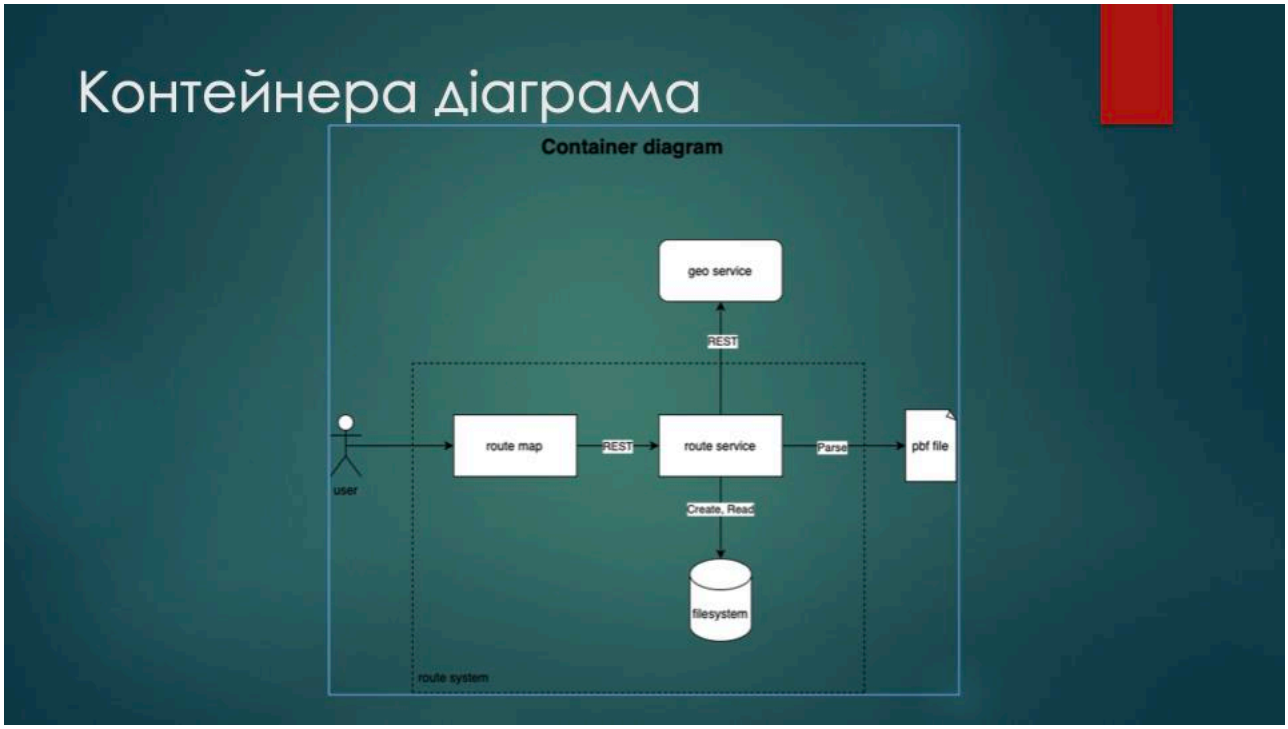

Рисунок Г.13 – Слайд «Контейнерна діаграма»

Рисунок Г.14 – Слайд «Компонента діаграма»

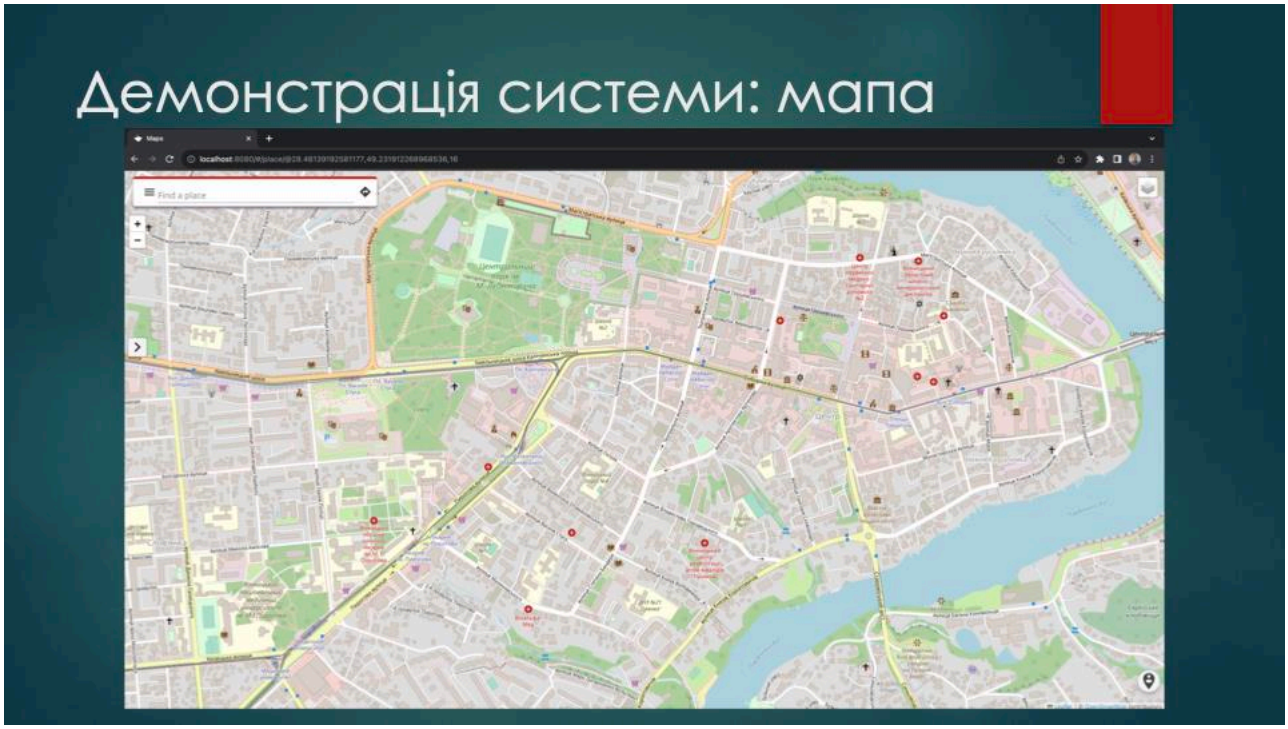

Рисунок Г.15 – Слайд «Демонстрація системи: мапа»

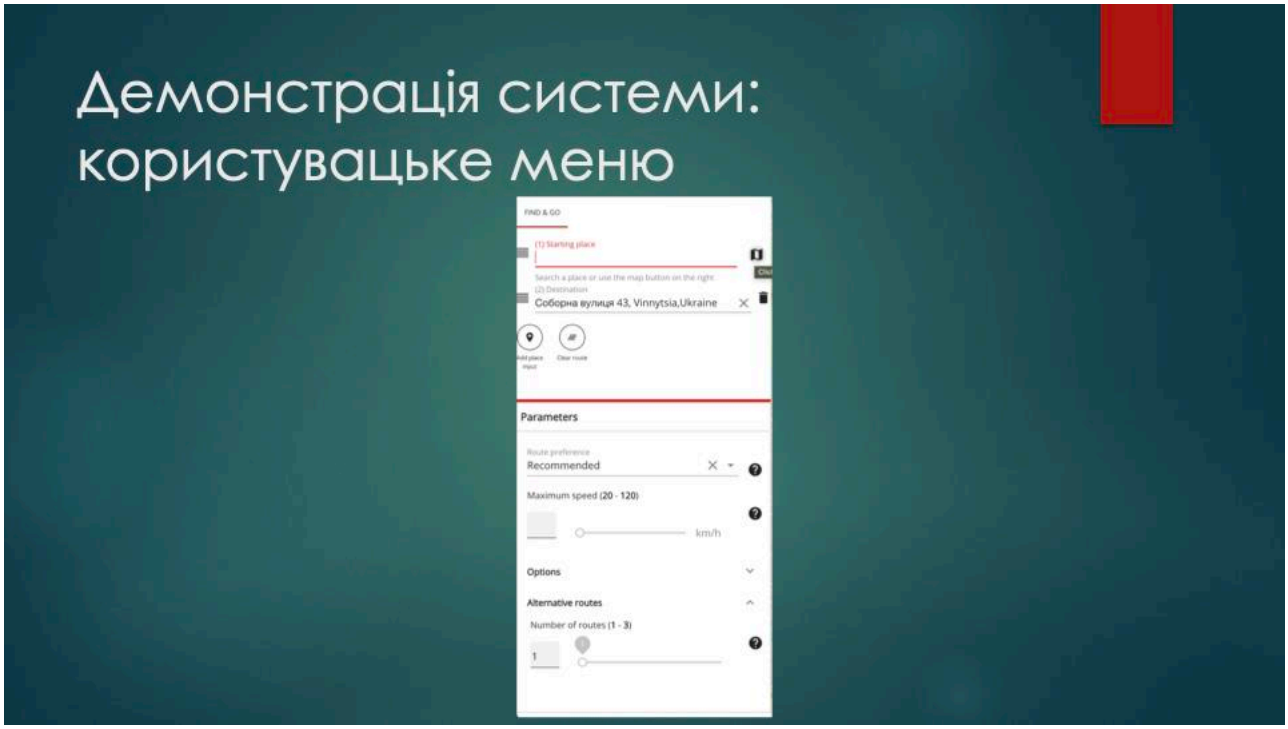

Рисунок Г.16 – Слайд «Демонстрація системи: користувацьке меню»

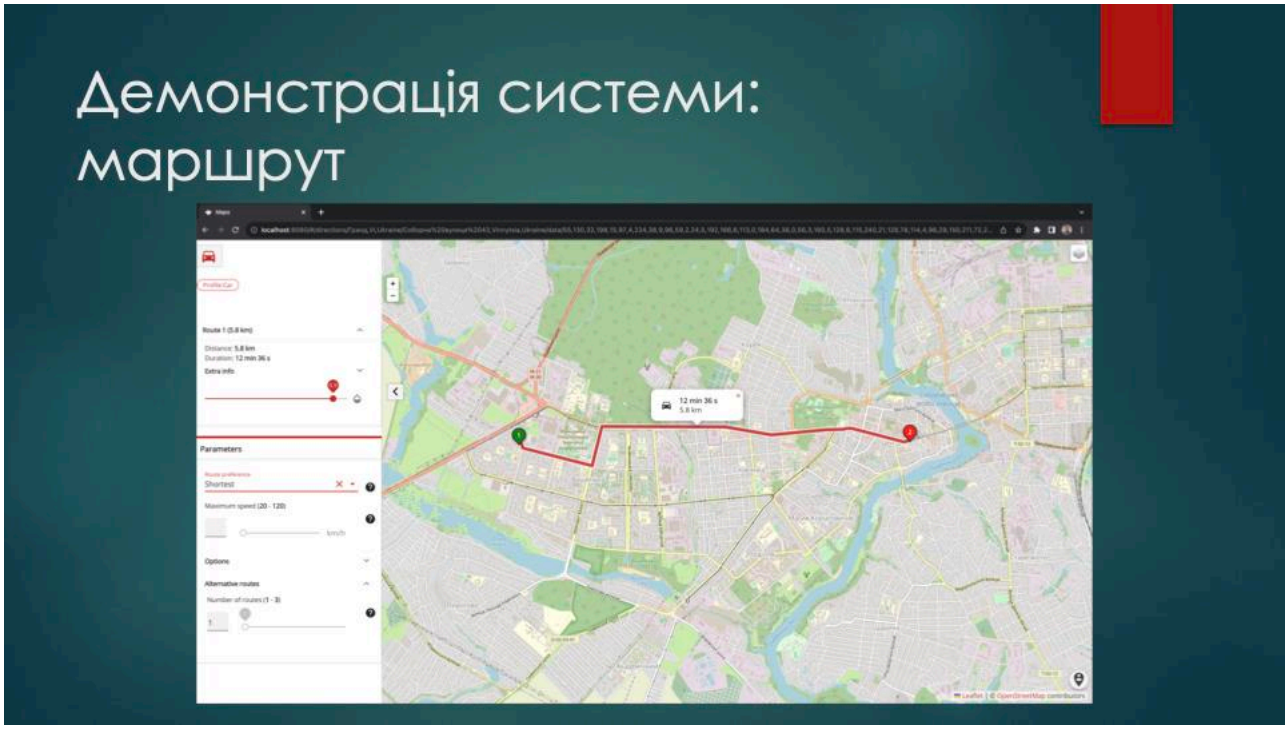

Рисунок Г.17 – Слайд «Маршрут»

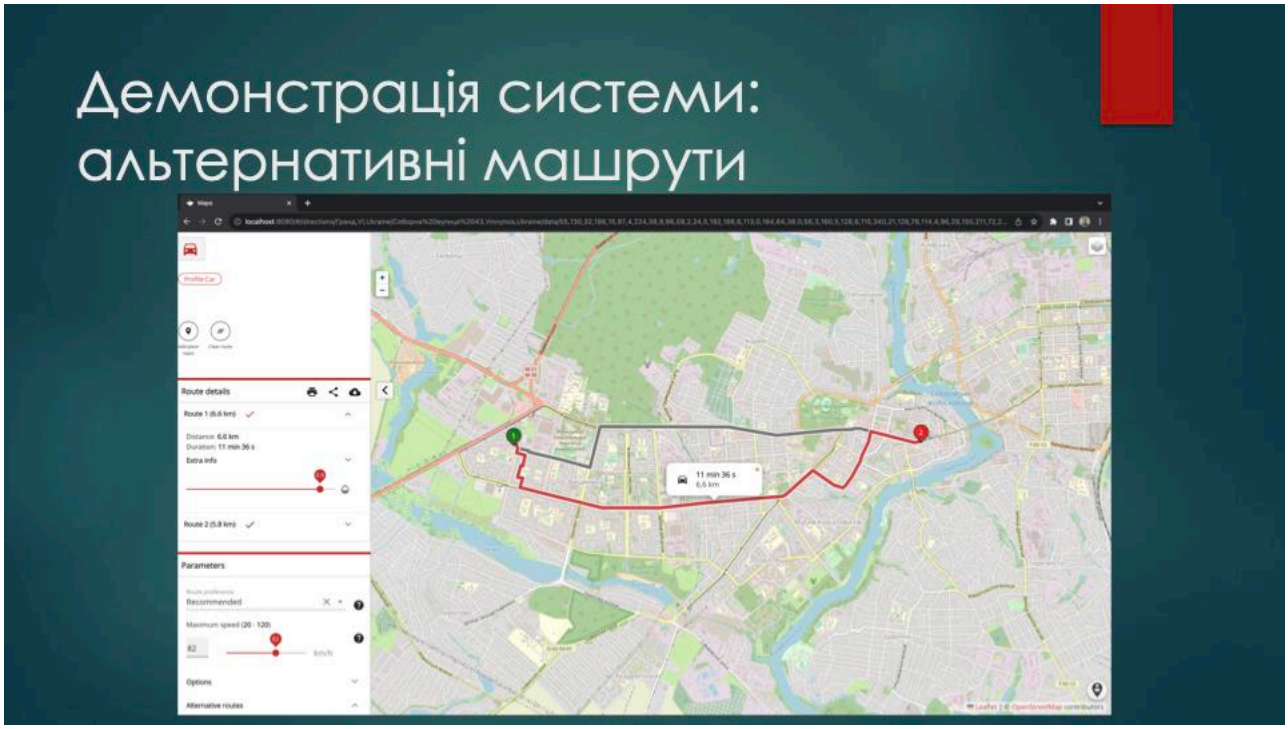

Рисунок Г.18 – Слайд «Альтернативні маршрути»

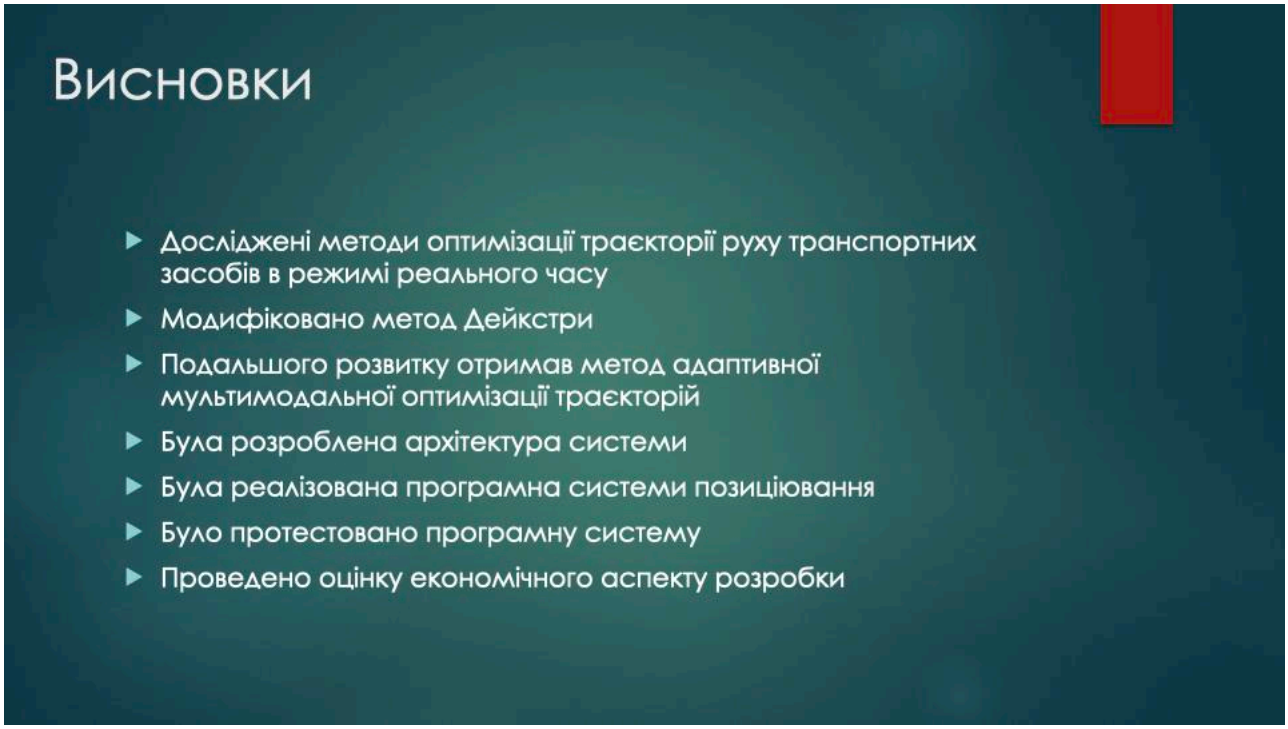

Рисунок Г.19 – Слайд «Висновки»

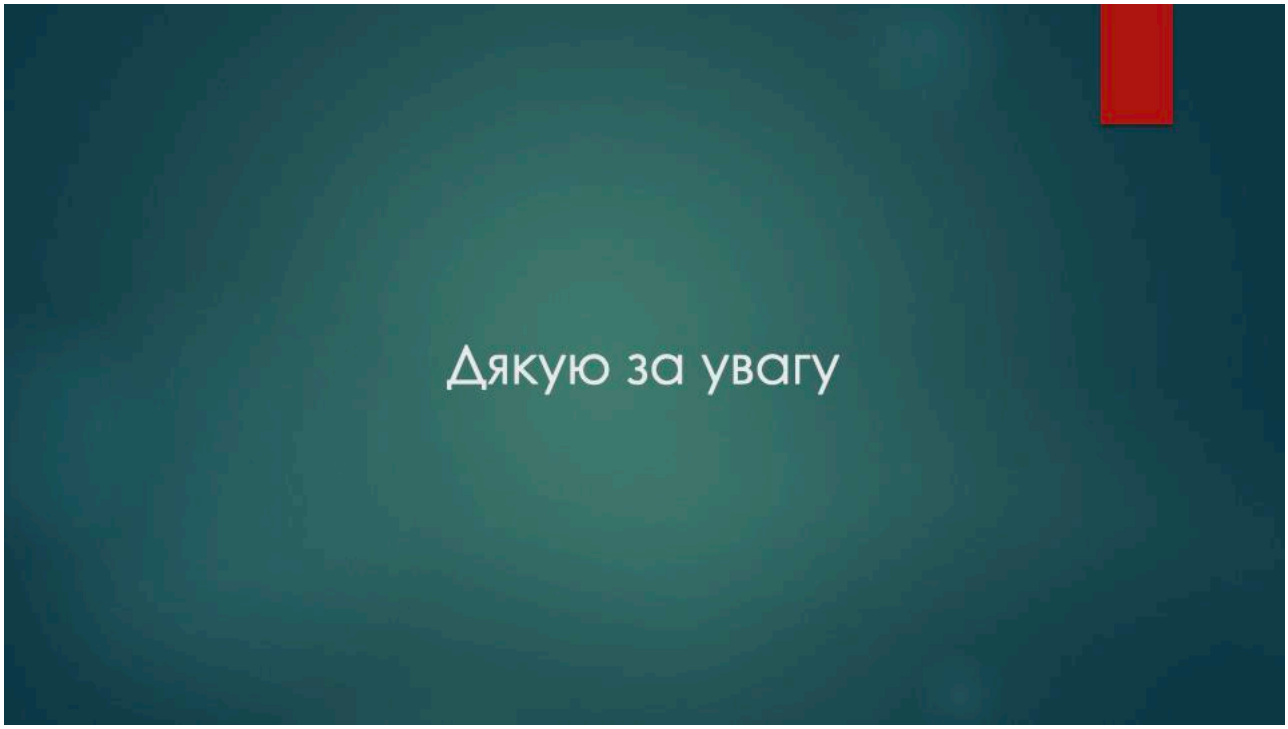

Рисунок Г.20 – Слайд «Дякую за увагу»