Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп’ютерних наук та штучного інтелекту
Кафедра комп’ютерних систем та робототехніки

«Затверджую»
в.о. завідуючого кафедри
комп’ютерних систем та робототехніки
_____ к. ф.-м. н., доцент Максим Хруслов
«___» червня 2025 р.

Пояснювальна записка

до кваліфікаційної роботи
бакалавра

на тему: «**СИСТЕМА ОПТИМІЗАЦІЇ ТРАНСПОРТНИХ МАРШРУТІВ З ВИКОРИСТАННЯМ BIG DATA**»

Спеціальність 151 –Автоматизація та комп'ютерно-інтегровані технології
Галузь знань 15 – Автоматизація та приладобудування
Освітня програма «Автоматизація та комп’ютерно-інтегровані технології»

Захищено на засіданні
Екзаменаційної комісії № 46
протокол № __ від __.06.2025 р.
Оцінка _____ / _____
Голова Екзаменаційної комісії
_____ **ЧУГАЙ А.М.**

Виконав:
Студент групи КУ– 41
САВЧЕНКО Костянтин Іванович
_____

Керівник: к.е.н., доцент,
доцент ЗВО кафедри комп’ютерних систем та робототехніки
ЧУБ Ольга Ігорівна

Рецензент: д.т.н., професор, професор кафедри теоретичної та прикладної інформатики
ФРОЛОВ В’ячеслав Вікторович

Харків – 2025

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи бакалавра складається зі вступу, 4 розділів, висновків, списку використаних джерел і 3 додатків. Загальний обсяг роботи складає 70 сторінок, із яких 42 сторінки основної частини з 9 рисунками, 1 таблицею, 18 найменуваннями списку використаних джерел та трьома додатками.

Об'єкт дослідження – процес прийняття рішень щодо побудови оптимальних маршрутів у транспортних системах із використанням аналізу великих обсягів даних для підвищення ефективності пересування.

Предмет дослідження – методи побудови оптимальних маршрутів у транспортних мережах з використанням алгоритмів пошуку на графах, засобів обробки геопросторових даних та технологій аналізу Big Data.

Мета дослідження – підвищення ефективності пересування шляхом створення системи побудови оптимальних транспортних маршрутів на основі аналізу великих обсягів геопросторових даних.

Проблема, яку вирішує кваліфікаційна робота – забезпечення побудови маршрутів із урахуванням завантаженості доріг в умовах великих масивів даних, що перевищують можливості традиційних алгоритмів без інтеграції Big Data.

Область застосування – системи навігації й планування маршрутів у громадському та комерційному транспорті, логістичні платформи, служби таксі і доставки, інформаційно-аналітичні рішення для транспорту і міського планування.

Ключові слова: ОПТИМІЗАЦІЯ МАРШРУТІВ, BIG DATA, A* АЛГОРИТМ, ТРАНСПОРТНІ ПОТОКИ, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА.

ABSTRACT

The explanatory note to the bachelor's qualification thesis consists of an introduction, 3 chapters, conclusions, a list of references, and 4 appendices. The total volume of the thesis is 70 pages, including 42 pages of the main part, comprising 9 figures, 1 tables, 18 items in the list of references, and three appendices.

The object of the study – the decision-making process for constructing an optimal transport route with efficiency recalculation criteria based on the analysis of large volumes of data.

The subject of the research – methods for constructing optimal routes in transport networks using graph search algorithms, geospatial data processing tools and Big Data analysis technologies.

The purpose of the research – increase the efficiency of transportation by creating a system for constructing optimal transport routes based on the analysis of large volumes of geospatial data.

Problem solved by qualification thesis – ensuring the construction of routes with the requested road load in conditions of large data sets that exceed the capabilities of traditional algorithms without Big Data integration.

Field of application – navigation and route planning systems in public and commercial transport, logistics platforms, taxi and delivery services, information and analytical solutions for transport and urban planning.

Keywords: ROUTE OPTIMIZATION, BIG DATA, A* ALGORITHM, TRANSPORT FLOWS, CLIENT-SERVER ARCHITECTURE.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ І УМОВНИХ ПОЗНАЧЕНЬ

Big Data – надзвичайно великі та складні набори даних.

OSM – OpenStreetMap – відкритий ресурс, що керується спільнотою, який створює та розповсюджує безкоштовні, редаговані географічні дані світу.

Neo4j – нативна система керування графовими базами даних, призначена для зберігання, керування та запитів високопов'язаних даних за допомогою вузлів, зв'язків та властивостей.

Бекенд – Backend – серверна частина програми, яка відповідає за бізнес-логіку, зберігання та пошук даних, а також інтеграцію із зовнішніми сервісами.

Фронтенд – Frontend – клієнтський рівень застосунку, з яким користувачі безпосередньо взаємодіють.

API – Application Programming Interface – набір визначених правил і протоколів, що дозволяє різним програмним застосункам взаємодіяти один з одним.

JavaScript – високорівнева інтерпретована мова програмування.

TomTom API – набір веб-сервісів, які дозволяють інтегрувати у програми багаті функції картографування, маршрутизації та дорожнього руху.

React.js – бібліотека мови програмування JavaScript з відкритим кодом для створення інтерактивних користувацьких інтерфейсів з використанням компонентної архітектури.

Leaflet – бібліотека мови програмування JavaScript з відкритим кодом для створення інтерактивних веб-карт.

Golang – мова програмування з відкритим вихідним кодом, статично типізована, компільована, розроблена Google.

Кінцева точка – endpoint – URL адреса, яка діє як точка контакту між клієнтом API і сервером API.

A* – алгоритм пошуку для знаходження шляху з найнижчою вартістю між вузлами у зваженому графі.

GPS – Global Positioning System – супутникова навігаційна система, яка надає геолокаційну інформацію та інформацію про час.

Geofabrik – компанія, яка регулярно оновлює витяги даних OpenStreetMap та надає пов'язані послуги.

XML – Extensible Markup Language – текстовий формат для представлення структурованих даних за допомогою користувацьких тегів, що визначаються користувачем.

PBF – Protocolbuffer Binary Format – двійковий формат файлів, який використовується OpenStreetMap для зберігання та обміну картографічними даними.

Vite – платформо-незалежний фронтенд інструмент для швидкого створення веб-застосунків.

RESTful – Representational State Transfer – архітектурний стиль програмного забезпечення.

CORS – Cross-Origin Resource Sharing – механізм безпеки браузера, який використовує HTTP-заголовки для контролю того, чи дозволено веб-сторінці з одного джерела (домену) запитувати ресурси (наприклад, API, шрифти, зображення) з іншого джерела.

Cypher – декларативна мова запитів до графів, розроблена для баз даних (зокрема, Neo4j).

APOC – Awesome Procedures On Cypher – комплексна бібліотека процедур та функцій для Neo4j.

Nominatim – рушій геокодування та зворотного геокодування з відкритим вихідним кодом для даних OpenStreetMap.

HTTP – Hypertext Transfer Protocol – базовий протокол прикладного рівня.

URL – Uniform Resource Locator – стандартизована адреса, яка використовується для пошуку ресурсів в Інтернеті.

CSV – Comma-Separated Values – формат звичайного текстового файлу, в якому кожен рядок представляє запис, а окремі поля в записі розділяються комами (або іншим роздільником).

JSON – JavaScript Object Notation – текстовий формат обміну даними, який представляє структуровані дані за допомогою пар ключ-значення та впорядкованих списків.

Node.js – кросплатформне середовище виконання JavaScript з відкритим кодом, побудоване на рушії V8 Chrome, яке дозволяє виконувати код JavaScript поза браузером.

Express – мінімальний, неупереджений фреймворк веб-застосунків на базі Node.js.

Google Maps – веб-сервіс картографування та платформа API від Google.

ВСТУП

Кваліфікаційна робота присвячена розробці системи оптимізації транспортних маршрутів на основі алгоритмів пошуку в графах, геопросторових даних та технологій Big Data. Запропоновано підхід, що враховує топологічну структуру дорожньої мережі та низку критеріїв ефективності пересування – таких як довжина маршруту, прогнозований час у дорозі, рівень заторів та їхнє комбіноване співвідношення для досягнення збалансованого результату.

У межах дослідження реалізовано клієнт-серверну вебсистему, яка інтегрує графову модель транспортної інфраструктури, відкриті картографічні дані (OSM), сервіси зворотного геокодування та зовнішні API для отримання інформації про дорожню обстановку. Завдяки такій архітектурі забезпечується побудова маршрутів, адаптованих до умов реального часу та змінної транспортної ситуації.

Тема кваліфікаційної роботи є актуальною через зростання інтенсивності дорожнього руху, необхідність ефективного управління транспортними потоками та зменшення втрат часу в умовах щоденних заторів. Застосування сучасних методів маршрутизації, що базуються на аналізі великих обсягів структурованих і неструктурованих даних, відкриває нові можливості для інтелектуальної підтримки прийняття рішень у сфері міської мобільності.

Технології Big Data відіграють ключову роль у маршрутизації нового покоління, дозволяючи оперативно обробляти геопросторові й трафікові дані з різнорідних джерел. Такий підхід дає змогу враховувати не лише геометричні особливості дорожньої мережі, а й динамічні показники її завантаженості, що суттєво підвищує точність та релевантність отриманих маршрутів.

Запропоноване рішення базується на поєднанні алгоритму A*, бази графових даних Neo4j та сучасних інструментів фронтенд-розробки (React, Leaflet), що забезпечує модульність, масштабованість та високу продуктивність системи. Комплексність реалізованого підходу дозволяє ефективно адаптувати систему до реальних умов експлуатації в транспортній інфраструктурі міст.

Мета дослідження – підвищення ефективності пересування шляхом створення системи побудови оптимальних транспортних маршрутів на основі аналізу великих обсягів геопросторових даних.

Об’єкт дослідження – процес прийняття рішень щодо побудови оптимальних маршрутів у транспортних системах із використанням аналізу великих обсягів даних для підвищення ефективності пересування.

Предмет дослідження – методи побудови оптимальних маршрутів у транспортних мережах з використанням алгоритмів пошуку на графах, засобів обробки геопросторових даних та технологій аналізу Big Data.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОГЛЯД ТЕХНОЛОГІЙ

1.1. Поняття оптимізації транспортних маршрутів та роль Big Data

Оптимізація транспортних маршрутів – це систематичний процес визначення найефективнішого шляху або набору шляхів між пунктами відправлення та призначення в межах транспортної мережі. Це завдання моделюється як задача пошуку графів: перехрестя та зупинки представлені як вузли, тоді як сегменти доріг та сполучення між ними утворюють ребра з відповідними вагами. Традиційні алгоритми пошуку маршрутів, такі як алгоритм Дейкстри, Беллмана-Форда та A*[1], використовують ці зважені графи для обчислення найкоротших або найменш витратних шляхів. У практичному контексті «вартість» може кодувати різноманітні критерії: фізичну відстань, орієнтовний час подорожі, витрату палива, витрати на проїзд або складені показники, які поєднують кілька цілей.

Технології Big Data відіграють ключову роль у вирішенні цих проблем, дозволяючи отримувати, обробляти та аналізувати величезні, різнорідні набори даних з високою швидкістю.

Ключові джерела даних включають:

1. Дані про рух автомобілів та GPS-траєкторії, що надають детальну інформацію про швидкість транспортних засобів та затори.
2. Дані датчиків дорожнього руху (петлеві детектори, камери), що забезпечують безперервні вимірювання потоку, щільності та заповненості.
3. Звіти про інциденти, зібрані з краудсорсингу та потоки соціальних мереж, що сповіщають про аварії, дорожні роботи або раптові уповільнення.
4. OpenStreetMap (OSM) та інші географічні краудсорсингові репозиторії, що надають актуальні топологічні атрибути дорожньої мережі [2].

Інтеграція цих потоків у рамках системи великих даних (Big Data) [3], яка часто базується на розподіленому сховищі наприклад: HDFS, NoSQL

графових базах даних. Також подібна інтеграція базується на паралельних процесорах наприклад: Apache Spark, Flink. Це дозволяє майже в режимі реального часу перераховувати вагові коефіцієнти ребер. Постійно оновлюючи прогнози часу подорожі та індекси заторів, система динамічно коригує налаштування маршрутизації, щоб відображати поточні дорожні умови.

Більше того, розширені аналітичні та машинні моделі, навчені на історичних наборах даних, забезпечують можливості прогнозування: прогнозування гарячих точок заторів, оцінку ймовірності інцидентів та рекомендації щодо коригування часу відправлення. Такі прогностичні висновки збагачують модуль планування маршруту, передбачаючи майбутні стани мережі, а не реагуючи виключно на її поточний стан.

Підсумовуючи, оптимізація транспортних маршрутів використовує алгоритми пошуку на основі графів для визначення ефективних шляхів за кількома критеріями, тоді як технології великих даних забезпечують масштаб та гнучкість, необхідні для засвоєння даних про трафік та просторові дані в режимі реального часу з великим обсягом. Ця синергія дає змогу інтелектуальним, адаптивним рішенням маршрутизації, які краще задовольняють потреби сучасної міської мобільності.

1.2. Джерела геоданих і формати OSM

Геопросторові дані, що лежать в основі сучасних систем маршрутизації, походять від різноманітних відкритих та власницьких постачальників. У контексті нашої реалізації OpenStreetMap (OSM) слугує основним джерелом топологічної та семантичної інформації про дорожні мережі. OSM – це спільний краудсорсинговий проект, який надає безкоштовний доступ до редагуваних картографічних даних за ліцензією Open Database License [4]. Учасники з усього світу постійно збагачують набір даних такими деталями, як геометрія доріг, обмеження швидкості, обмеження поворотів та теги точок інтересу.

Хоча до глобальної бази даних OSM можна отримати повний доступ через planet.osm, практичні програми зазвичай використовують регіональні або національні витяги для зменшення накладних витрат на обробку. Geofabrik пропонує регулярно оновлювані, попередньо відфільтровані витяги у форматах XML та PBF для окремих континентів, країн та субрегіонів. Ці витяги забезпечують керовані розміри даних – наприклад, витяг OSM для України має розмір близько кількох гігабайт після стиснення – та гарантують актуальність завдяки щотижневим або щоденним графікам оновлення. Альтернативні джерела включають Metro Extracts від Mapzen та Overpass API, який дозволяє на вимогу, на основі запитів, отримувати певні об'єкти карти.

Дані OSM представлені у двох основних форматах файлів:

1. OSM XML (.osm) – зрозуміле для людини представлення на основі тегів з використанням елементів `<node>`, `<way>` та `<relation>`. Кожен елемент містить унікальні ідентифікатори, атрибути координат та вкладені піделементи `<tag>`, які кодують метадані (наприклад, `highway=primary`, `name=Main Street`). Хоча великі XML-файли легко перевіряти або редагувати, вони потребують значних накладних витрат на парсинг та ввод-вивод.

2. Двійковий формат Protocolbuffer (.pbf) – компактне двійкове кодування тієї ж моделі даних, що використовує Protocol Buffers від Google для ефективної серіалізації. PBF приблизно в п'яту частину розміру менший за свій XML-аналог і підтримує швидкий потоково-орієнтований розбір. Як наслідок, він став фактичним стандартом для масового імпорту в графові бази даних та конвеєри просторової обробки [5].

У нашій системі процес отримання геоданих починається з витягу PBF з України, отриманого від Geofabrik. Парсер на основі Golang обробляє файл PBF для генерації двох CSV-файлів: `nodes.csv` (ідентифікатори вузлів з широтою/довготою) та `edges.csv` (сегменти шляху з ідентифікаторами початкових/кінцевих вузлів та обчисленими геодезичними довжинами). Ці CSV-файли потім імпортуються в Neo4j за допомогою процедур імпорту

APOC, зберігаючи багату топологію та семантику, керовану тегами, необхідні для обчислення маршруту.

1.3. Інструменти зворотного геокодування та пошуку точок

Зворотне геокодування перетворює географічні координати на адреси або адміністративні регіони, які читає людина. У нашій системі ця функціональність забезпечується Nominatim – геокодувальним механізмом з відкритим кодом, побудованим на даних OpenStreetMap [6]. Коли серверна частина отримує пару значень широти та довготи, вона надсилає запит зворотного геокодування до Nominatim, який повертає структуровану ієрархію адрес (країна, регіон, округ, місто тощо). Перевіряючи повернуті адміністративні поля, система визначає, чи знаходиться точка в Київській області.

Щоб пов'язати довільну координату карти з найближчим вузлом у нашому графі маршрутизації, ми виконуємо просторовий пошук найближчого сусіда безпосередньо в Neo4j. Серверна частина створює точку в пам'яті з вхідних координат, а потім обчислює площинні відстані до всіх вузлів графа, кожен з яких зберігає свою широту та довготу як властивості. Сортуючи ці відстані та вибираючи мінімальну, система ідентифікує єдину найближчу вершину. Цей крок зіставлення точок гарантує, що всі наступні операції пошуку маршруту будуть виконуватися на дійсних вузлах графа, тим самим зберігаючи просторову точність та стійкість, коли користувачі вибирають довільні місця розташування.

1.4. Сервіси комерційної маршрутизації

Комерційні платформи маршрутизації представляють клас сервісів корпоративного рівня, розроблених для надання високоточних, масштабованих рішень маршрутизації, що базуються на власних

геопросторових наборах даних, передових потоках телеметрії та суворих зобов'язаннях щодо рівня обслуговування. Великі постачальники, включаючи Google Maps, HERE Technologies, TomTom та Mapbox, підтримують комплексні картографічні сховища завдяки стратегічним партнерствам з урядовими установами, операторами сенсорних мереж та великими автопарками [7]. Ця спільна екосистема забезпечує поширення критичних оновлень мережі (наприклад, розширення інфраструктури, регуляторні обмеження, динамічні обмеження швидкості) в індекс маршрутизації майже в режимі реального часу, тим самим зберігаючи цілісність та своєчасність обчислень шляху.

Ці платформи пропонують широкий набір можливостей, адаптованих до вимог складних сценаріїв маршрутизації.

Моделювання дорожнього руху в реальному часі та прогнозування: поєднуючи дані про потоки в реальному часі з історичними та контекстними змінними (часові цикли, спеціальні події, погодні умови), ці сервіси використовують алгоритми машинного навчання для прогнозування траєкторій заторів та коригування очікуваного часу прибуття з високою часовою роздільною здатністю. Така прогнозна аналітика дозволяє адаптувати маршрути, зменшуючи затримки до їх виникнення.

Профілі маршрутизації, специфічні для домену: такі сервіси враховують різнорідні способи мобільності та нормативні обмеження. Приклади включають маршрутизацію вантажів та важких транспортних засобів, що включає обмеження щодо розмірів транспортних засобів, ваги та небезпечних вантажів, а також планування мультимодальних маршрутів, яке інтегрує сегменти приватного водіння з мережами громадського транспорту, велосипедних та пішохідних маршрутів. Параметри уникнення, що налаштовуються користувачем (наприклад, платні дороги, пороми, зони з низьким рівнем викидів), додатково уточнюють простір пошуку для узгодження з операційною політикою або цілями сталого розвитку.

У контексті нашого прототипу ми використовували API TomTom Traffic як модульний компонент сервісу. Спеціальна утиліта Golang запускається при викликі відповідної кінцевої точки на стороні сервера, отримуючи індекси заторів та перекалібруючи коефіцієнти вагових характеристик, що використовуються в евристиці пошуку A*. Хоча обмеження вільного рівня в п'ять запитів на секунду накладає часові обмеження на масштабні оновлення мережі, модульна архітектура дозволяє легко замінити альтернативних комерційних постачальників, для задоволення потреб, що постійно змінюються, пов'язаних з часовою деталізацією, географічним покриттям та оптимізацією витрат.

1.5. Платформи та фреймворки для клієнтської частини

Інтерфейс системи маршрутизації реалізовано за допомогою низки сучасних веб-технологій, обраних для оптимізації гнучкості, продуктивності та зручності обслуговування. React.js забезпечує компонентну архітектуру, в якій елементи керування картою, форми введення та візуалізації маршрутів інкапсульовані як модулі повторного використання [8].

Інтерактивний рендеринг карт обробляється Leaflet, інтегрованим через прив'язки React-Leaflet для перетворення імперативних операцій відображення в декларативні елементи React [9]. Ця інтеграція спрощує управління кодом та дозволяє використовувати розширені функції, такі як кластеризація маркерів та налаштування шарів плиток без зайвої складності

Процеси розробки та збірки використовують Vite як основний інструментарій [10]. Під час розробки заміна модулів забезпечує миттєвий зворотний зв'язок щодо змін коду, тоді як виробничі збірки отримують вигоду від мінімізації коду та хешування ресурсів, щоб мінімізувати розмір пакета та покращити час завантаження.

Разом ці платформи та фреймворки створюють надійну клієнтську архітектуру, здатну забезпечувати адаптивний та інтерактивний досвід відображення, зберігаючи при цьому чисту, модульну структуру коду.

1.6. Технології серверної частини та бази даних

Бекенд використовує Express.js для відкриття RESTful кінцевих точок для обчислення маршрутизації, геопросторових запитів та оновлення трафіку [11]. Проміжне програмне забезпечення налаштовує спільний доступ до ресурсів між джерелами (CORS) та базову перевірку запитів [12]. Основна логіка виконує запити Cypher до Neo4j через офіційний `neo4j-driver`, використовуючи власні функції графів, такі як процедури пошуку найкоротшого шляху та обчислення просторової відстані, для ефективного виявлення маршрутів та зіставлення точок [13].

Утиліта на базі Golang виконує обробку даних: вона аналізує екстракти OpenStreetMap PBF, створюючи файли `nodes.csv` та `edges.csv`, які імпортуються в Neo4j через процедури імпорту APOC [14]. Утиліту для оновлення даних про завантаженість доріг може запускати кінцева точка сервера: вона запитує API TomTom Traffic, перераховує коефіцієнти ваги ребер та оновлює відповідні властивості в графі. Щоб дотримуватися обмеження вільного рівня в п'ять запитів за секунду, оновлення об'єднуються в пакети та обмежуються швидкістю всередині утиліти.

Зворотне геокодування обробляється Nominatim: HTTP-запит до його API повертає структуровану адресу для заданих координат, що дозволяє системі перевіряти адміністративні межі. Змінні середовища надають конфігурацію для підключень до бази даних, ключів API та URL-адрес сервісів, зберігаючи конфіденційні параметри окремо від коду програми.

Це поєднання технологій Express.js, Neo4j та Golang створює простий, але потужний сервер, здатний приймати необроблені геопросторові дані,

виконувати складні операції з графами та інтегрувати інформацію про дорожній рух у режимі реального часу.

Висновки за розділом 1

У даному розділі було закладено фундаментальне розуміння оптимізації транспортних маршрутів, сформулювавши теоретичні та практичні аспекти пошуку шляхів на основі графів, доповнені можливостями великих даних. Обговорення розпочалося з розгляду цілей оптимальних алгоритмів, таких як A*, які балансують відстань, час подорожі та показники заторів. Потім було розглянуто OpenStreetMap як основне джерело геопросторових даних, порівняно формати XML та PBF, а також детально описано методи вилучення, парсингу та імпорту топології мережі в базу даних графів.

Згодом у цьому розділі було розглянуто механізми перетворення необроблених GPS-координат у практичні вершини графа за допомогою зворотного геокодування (через Nominatim) та зіставлення найближчих вузлів у Neo4j, що забезпечує просторову точність та стійкість системи. В огляді архітектури клієнтського фреймворку було виділено модель компонентів React, механізм візуалізації карт Leaflet та оптимізований робочий процес збірки Vite, тоді як у розділі, присвяченому серверу, було описано використання утиліт Express.js, Neo4j-driver та Golang для обробки даних та конвеєрів оновлення даних про трафік.

Інтегруючи ці компоненти, у даному розділі було продемонстровано, як цілісний технологічний стек може підтримувати адаптивні рішення маршрутизації в режимі реального часу. Представлені висновки та фреймворки створюють міцну основу для аналізу вимог та проектування системи, що йдуть в наступному розділі, гарантуючи, що подальша розробка відповідає як теоретичним принципам, так і практичним міркуванням впровадження.

РОЗДІЛ 2
ПОСТАНОВКА ЗАДАЧІ ТА ПРОЕКТУВАННЯ СИСТЕМИ

2.1. Функціональні вимоги

Система повинна дозволяти користувачам вказувати точки відправлення та призначення через інтерактивний інтерфейс карти, приймаючи довільні вхідні дані широти/довготи та перетворюючи їх на вузли графа за допомогою просторового зіставлення найближчих сусідів. Вона повинна підтримувати обчислення та отримання трьох різних профілів маршруту: шлях найкоротшої відстані, шлях з мінімальним часом подорожі з урахуванням поточних дорожніх умов та збалансований шлях, який оптимізує зважену комбінацію відстані та часу транзиту.

RESTful API повинен надавати кінцеві точки для зіставлення точок (`/snap`), генерації маршруту (`/routes`), зворотного геокодування (`/geocode`) та оновлення трафіку на вимогу (`/traffic-update`) [15]. Кожна кінцева точка повинна перевіряти вхідні параметри, повертати структуровані відповіді JSON з ідентифікаторами вузлів, географічними координатами та відповідними метриками (відстань, розрахунковий час, індекс перевантаженості), а також використовувати заголовки CORS для забезпечення міжджерельних запитів клієнтів.

Бекенд повинен отримувати необроблені дані OpenStreetMap у форматі PBF та перетворювати їх на `nodes.csv` та `edges.csv` за допомогою утиліти парсингу на базі Golang. Отримані CSV-файли мають бути імпортовані в Neo4j за допомогою процедур імпорту APOC, зберігаючи властивості вузлів (широта, довгота) та атрибути ребер (довжина шляху, початкова вага). Оновлення даних трафіку здійснюється виключно на вимогу: після активації кінцевої точки оновлення пов'язана утиліта Golang запитуватиме API TomTom Traffic та обчислюватиме ваговий коефіцієнт на ребрах графу.

На стороні клієнта програма повинна відображати шари карти, дозволяти вибір точок та відображати кілька геометрій маршруту з різними візуальними стилями. Вона повинна представляти порівняльну інформацію – довжину маршруту, орієнтовний час та рівень заторів – на панелі накладання. Інтерактивні елементи керування повинні дозволяти користувачам оновлювати дані про дорожній рух, перемикати профілі маршрутів та перевіряти окремі точки маршруту.

Ці функціональні вимоги встановлюють основні можливості, необхідні для адаптивної, орієнтованої на користувача системи маршрутизації, яка використовує алгоритми пошуку графів та аналітику трафіку в режимі реального часу.

2.2. Нефункціональні вимоги

Для забезпечення високої якості користувацького досвіду, взаємодія з картою на стороні клієнта, така як панорамування, масштабування та вибір точок маршруту, повинна послідовно відображатися протягом 100 мс після відповідних дій користувача. Ця затримка цілі допомагає підтримувати безперервність сприйняття та мінімізує ризик збоїв інтерфейсу під час дослідницьких навігаційних завдань.

Продуктивність кінцевої точки RESTful є критично важливою для швидкості реагування системи. За нормальних умов роботи виклики `/snap` та `/geocode` повинні завершуватися протягом 500 мс, тоді як більш обчислювально ресурсоємна кінцева точка `/routes`, яка виконує пошук шляху на основі Neo4j, повинна повертати результати не більше ніж за 1000 мс. Ці пороги балансують алгоритмічну складність з очікуваннями користувача щодо майже миттєвого зворотного зв'язку.

Горизонтальна масштабованість досягається за допомогою екземплярів API без збереження стану, які можна динамічно налаштовувати або виводити з експлуатації.

Вбудовуючи ці кількісні цілі в робочі процеси моніторингу та розгортання, система постійно забезпечуватиме адаптивні можливості геопросторової маршрутизації, залишаючись при цьому адаптивною до змінних умов навантаження.

2.3. Загальна архітектурна схема клієнт-серверної системи

Система використовує багаторівневу клієнт-серверну архітектуру [16], в якій користувач взаємодіє через браузерний додаток React.js. Фронтенд фіксує вхідні координати, відображає інтерактивні шари карти та візуалізує обчислені маршрути. Усі взаємодії з картою та команди керування, такі як ініціювання оновлення трафіку, перетворюються на HTTP-запити, що спрямовуються на серверну частину.

Бекенд, реалізований у Node.js з Express.js, надає набір кінцевих точок RESTful, які інкапсулюють основні доменні служби: просторовий пошук (`/snap`), обчислення маршруту (`/routes`), зворотне геокодування (`/geocode`) та повторне калібрування трафіку на вимогу (`/traffic-update`). Вхідні запити проходять через проміжні шари, які застосовують CORS, перевіряють схеми введення та обробляють проблеми автентифікації або обмеження швидкості перед відправкою до компонентів служби.

У межах серверного сервісного рівня спеціалізовані модулі координують зовнішні та внутрішні джерела даних. Модуль геокодування взаємодіє з Nominatim для вирішення адміністративної інформації для заданих координат. Модуль трафіку викликає утиліту на основі Golang для запитів до API трафіку TomTom та поступового оновлення ваг ребер графу. Модуль маршрутизації виконує параметризовані запити Cypher до графу властивостей Neo4j для виконання зіставлення найближчих вузлів та обходу A* на основі останніх параметрів ваг.

Збереження даних та операції з графами знаходяться у графовій базі данх Neo4j, налаштованому для високої доступності, з індексами властивостей

та просторовими індексами, що забезпечують швидкий пошук та обхід. Завантаження даних – розбір екстрактів OSM PBF у артефакти CSV та їх імпорт через процедури APOC – відбувається як крок перед розгортанням, встановлюючи початкову топологію графу.

Процеси API без збереження стану можна масштабувати горизонтально для керування збільшеним навантаженням, тоді як вузли Neo4j забезпечують стійкість та відмовостійкість (рис. 2.1).

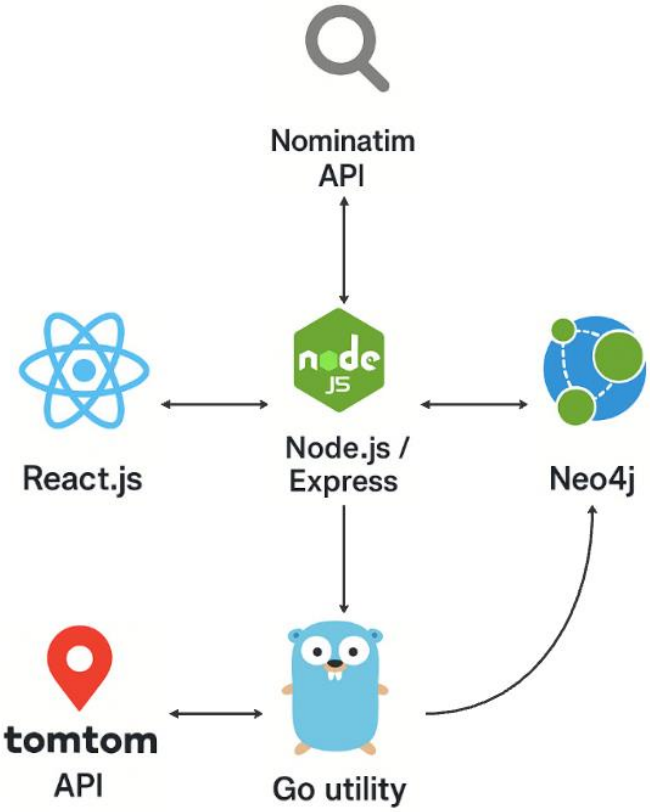

Рисунок 2.1 – Схематичне зображення архітектури системи

Ця архітектура чітко розподіляє завдання: клієнт React.js зосереджується на користувацькому досвіді та візуалізації, сервіси Express.js інкапсулюють основну логіку та зовнішні інтеграції, а Neo4j забезпечує ефективне зберігання графів та обчислювальні можливості. Разом ці компоненти утворюють цілісну систему, здатну надавати адаптивні сервіси маршрутизації на основі даних.

2.4. Проектування графової моделі в Neo4j

Транспортна мережа представлена в Neo4j як орієнтований граф властивостей, де вершини (вузли) відповідають геопросторовим точкам, а ребра (зв'язки) позначають сегменти дороги. Кожен вузол позначено як `Point` та інкапсулює тип Neo4j `Point` для географічних координат разом з ідентифікаторами, які зберігають зв'язок з оригінальним набором даних OpenStreetMap. Наприклад, типовий вузол містить (рис. 2.2):

1. `elementId`: унікальний ідентифікатор елемента OSM;
2. `id`: внутрішній числовий ключ;
3. `lat`, `lon`: значення широти та довготи;

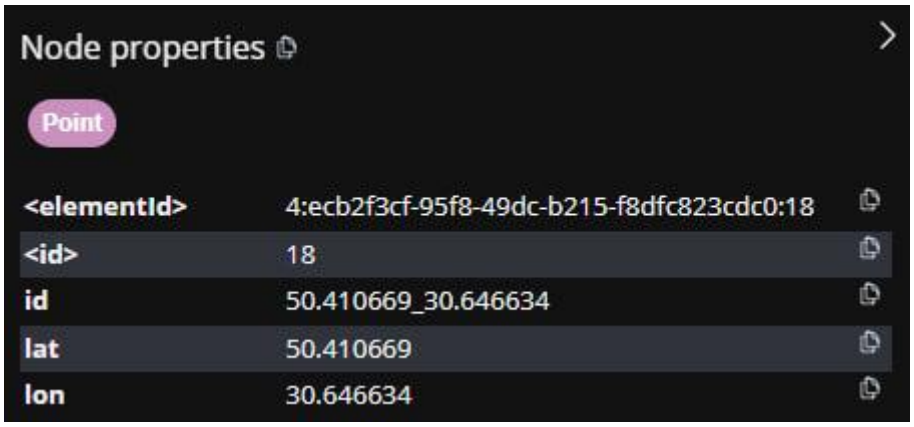

Рисунок 2.2 – Приклад вигляду `Point` у Neo4j

Просторове індексування властивості `Point` забезпечує ефективні запити на близькість та лежить в основі логіки зіставлення найближчих вузлів, що використовується службою маршрутизації.

Сегменти доріг моделюються як зв'язки типу `ROAD`, кожен з яких містить багатий набір атрибутів, необхідних для пошуку шляху.

Зразок зв'язку `ROAD` містить такі властивості, як (рис. 2.3):

1. `elementId`: посилання на елемент OSM.
2. `id`: унікальний ключ зв'язку.
3. `highway`: класифікація дороги.

4. `name`: назва вулиці.

5. `length`: геодезична довжина в метрах, розрахована за формулою гаверсинуса.

6. `maxSpeed`: середня швидкість автомобілів на цьому відрізку (згідно TomTom API).

7. `currentSpeed`: спостережена швидкість з даних про дорожній рух у реальному часі (км/год).

8. `traffic`: динамічний коефіцієнт заторів, що використовується як критерій (рівень заторів). Розраховується за формулою ((MaxSpeed - CurrentSpeed) / MaxSpeed).

9. `travelTime`: попередньо обчислений час проходження в секундах, використовується як комбінований критерій, який враховує і довжину і затори. Розраховується за формулою (length / (CurrentSpeed * 1000 / 3600)).

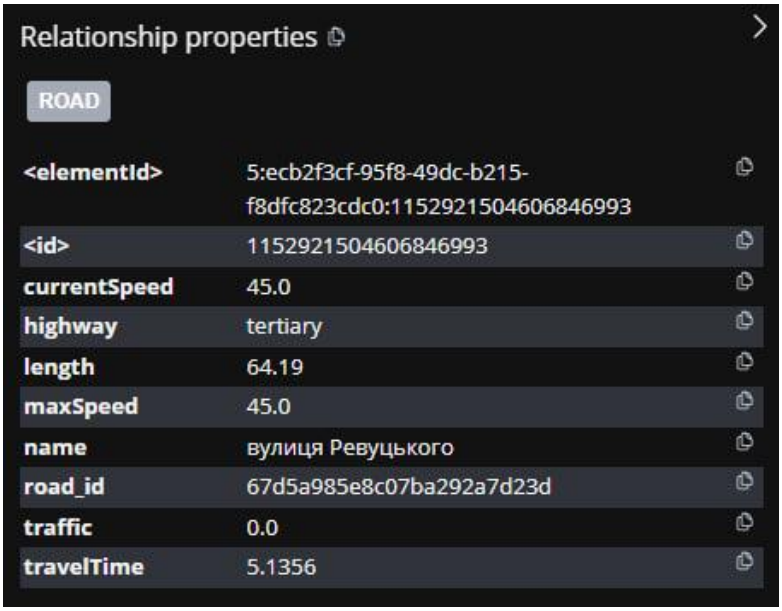

Рисунок 2.3 – Приклад вигляду ` ROAD` у Neo4j

Ці властивості дозволяють алгоритму A* обчислювати функції вартості, які враховують відстань, дорожні умови в режимі реального часу та прогнози часу подорожі. Спрямованість зв'язку відповідає природному потоку руху базової топології дороги.

Обмеження унікальності для `Point.id` та `ROAD.id` у поєднанні з індексами властивостей часто запитуваних полів (`highway`, `name`) гарантують цілісність даних та прискорюють виконання запитів. Поступові оновлення атрибутів трафіку застосовуються безпосередньо до існуючих зв'язків, уникаючи повного перезавантаження графа та зберігаючи загальну швидкість реагування системи.

Ця схема графа збалансовує потребу в просторовій точності, часовій адаптивності та обчислювальній ефективності, тим самим забезпечуючи надійну основу для оптимізації маршруту в режимі реального часу на основі даних у Neo4j [17].

2.5. Опис REST-інтерфейсів на Express

Бекенд використовує Express.js для реалізації набору RESTful кінцевих точок, які надають доступ до основних маршрутизаційних та геопросторових сервісів. Кожна кінцева точка дотримується HTTP-конвенцій, використовуючи відповідні методи, коди стану та корисні навантаження JSON.

Кінцева точка (GET /snap):

Мета: визначити найближчу вершину графа до довільних географічних координат.

Параметри: параметри запиту `lat` та `lng` (число з плаваючою комою).

Поведінка: перевіряє числовий ввід, а потім запускає запит просторової відстані Cypher для вузлів `Point`.

Відповідь: у разі успіху (200), JSON `{ key, lat, lng, distance }`. Повертає 400 для недійсних параметрів або 404, якщо точка не знайдена.

Кінцева точка (GET /routes):

Мета: Отримати кілька варіантів маршруту між двома існуючими ідентифікаторами вузлів.

Параметри: `from` та `to` як параметри запиту (ідентифікатори вузлів).

Поведінка: Перевіряє наявність обох ідентифікаторів, потім перебирає три профілі витрат – `length`, `traffic`, `travelTime` – викликаючи `apoc.algo.aStar` для кожного.

Відповідь: Об'єкт JSON з ключами `length`, `traffic` та `travelTime`, кожен з яких містить або `null`, або `{ route: [ { id, lat, lng }... ], totalLength, totalTime }`. Повертає 400, якщо параметри відсутні, 500 у разі помилок обробки.

Кінцева точка (GET /geocode):

Мета: Зворотне геокодування пари координат для отримання компонентів адреси.

Параметри: `lat`, `lng` як параметри запиту.

Поведінка: Для цілей документації діє як проксі для зворотної кінцевої точки Nominatim, повертаючи її відповідь JSON безпосередньо.

Відповідь: Структурована ієрархія адрес Nominatim або 400 у разі недійсного введення.

Кінцева точка (POST /traffic-update):

Призначення: Запустити оновлення даних трафіку на вимогу.

Поведінка: Ініціює утиліту Golang через фоновий процес, який отримує індекси перевантаження з API TomTom, перераховує ваги ребер та застосовує оновлення до графа Neo4j.

Відповідь: JSON `{ updatedCount, durationMs }` підсумовує кількість змінених зв'язків та затримку операції. Повертає 202 Прийнято, якщо завдання успішно поставлено в чергу, або 500 у разі невдалого запуску.

Кожна кінцева точка дотримується шаблонів RESTful, повертаючи відповідні коди стану HTTP (200, 202, 400, 404, 500) та уніфіковані конверти JSON. Перевірка вхідних даних використовує прості умовні перевірки в обробниках маршрутів, а помилки форматуються як `{ error: string }` для полегшення обробки помилок на стороні клієнта.

Цей дизайн інтерфейсу REST забезпечує чітке розділення проблем між семантикою транспортного рівня та бізнес-логікою, забезпечуючи надійну та видиму поверхню API для клієнтських застосунків та сторонніх інтеграторів.

2.6. Дизайн клієнтських компонентів

Клієнтська програма структурована навколо чіткого розділення питань представлення, взаємодії та управління даними. Компонент верхнього рівня ініціалізує контекст картографування за допомогою React та Leaflet, забезпечуючи адаптивне полотно для геопросторового відображення. Параметри ініціалізації карти, такі як координати центру, рівень масштабування та конфігурація шару плитки, визначаються декларативно, що забезпечує легке налаштування стилю та поведінки.

Взаємодія з користувачем абстрагована у спеціальні модулі: введення та перевірка координат, обробка кліків на карті та панелі керування інтерфейсом користувача. Події введення запускають робочі процеси отримання даних, такі як прив'язка до найближчого вузла графа та отримання варіантів маршруту, через єдиний сервісний рівень, який інкапсулює виклики REST. Сервісний рівень надає функції для просторового пошуку, отримання маршруту та оновлення трафіку, відокремлюючи мережеву логіку від візуальних компонентів.

Керування станом використовує API контексту React для поширення вибраних точок, даних маршруту та прапорців видимості по всьому дереву компонентів без надмірного деталізування. Побічні ефекти, такі як оновлення маркерів карти, рендеринг поліліній та коригування області перегляду, керуються за допомогою React-хуків, які реагують на зміни стану та відповідно узгоджують шари Leaflet.

Візуальний інтерфейс складається з трьох основних областей: полотна карти, бічної панелі керування та вбудованих сповіщень. Бічна панель представляє вибрані параметри для профілів маршрутів, відображає зведення

метрик та надає тригери дій (наприклад, побудова маршрутів, скидання карти, оновлення трафіку). Стиль компонентів регулюється централізованою темою, що сприяє узгодженості кольорових палітр, типографіки та інтервалів.

Міркування щодо продуктивності включають використання шаблонів мемоізації для запобігання надлишковим рендерингам, групування шарів Leaflet для пакетних оновлень та мінімальну залежність від глобальних повторних рендерингів, коли змінюються лише певні елементи інтерфейсу. Архітектура підтримує легке розширення – нові елементи керування або накладання можна вводити, додаючи автономні компоненти, які підписуються на спільний контекст та викликають функції рівня сервісу.

Завдяки модульності, декларативній обробці стану та єдиному інтерфейсу сервісу, цей дизайн сприяє підтримці розробки та чіткому перемиканню між проектуванням системи та детальною реалізацією в наступних розділах.

Висновки за розділом 2

У цьому розділі було створено комплексну структуру проектування запропонованої системи транспортної маршрутизації. Функціональні вимоги окреслили орієнтовані на користувача робочі процеси для вибору координат, обчислення мультимодальних маршрутів та перекалібрування трафіку на вимогу. Нефункціональні вимоги визначили якісні та кількісні цільові показники продуктивності як для інтерактивності на стороні клієнта, так і для реагування серверної частини, а також принципи масштабованості та надійності для керівництва стратегіями розгортання.

Архітектурна схема сформулювала багаторівневу топологію клієнт-сервер, в якій інтерфейс карти на основі React взаємодіє через кінцеві точки RESTful з серверною частиною Express.js. Ця серверна частина керує геопросторовими сервісами, а саме: визначення найближчого вузла, пошук шляху через Neo4j та зворотне геокодування, делегуючи операції оновлення

даних про трафік утиліті Golang, яка взаємодіє із зовнішніми API. Графова модель у Neo4j була розроблена для представлення дорожніх мереж як орієнтованих графів властивостей, з просторовими індексами та багатоатрибутивними зв'язками, що забезпечують ефективні A* обходи за динамічними ваговими параметрами.

У сукупності ці дизайнерські рішення балансують модульність, продуктивність та розширюваність. Вони створюють чіткий розмежування завдань між представленням, логікою домену та управлінням даними, закладаючи надійний план для подальшої реалізації. У розділі 3 буде впроваджено цей дизайн через детальне налаштування середовища, конвеєри обробки даних, розробку API та інтеграцію на стороні клієнта.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ СИСТЕМИ

3.1. Налаштування середовища розробки

Фаза впровадження починається зі створення узгодженого середовища розробки, яке підтримує швидку ітерацію та надійні збірки. Основою є Node.js (версії 16 або вище) разом з npm для керування залежностями та оркестрації скриптів. Ініціалізація проекту генерує маніфест, який містить усі необхідні бібліотеки та інструменти, що гарантує легке повторення налаштувань середовища на різних машинах.

Для фронтенду Vite обрано основним інструментарієм для збірки та розробки. Архітектура Vite, заснована на власних модулях ES, забезпечує майже миттєві оновлення під час змін коду та створює оптимізовані пакети з такими функціями, як розділення коду та струшування дерев. Файли конфігурації визначають, як обслуговуються вихідні ресурси та як запити API передаються до бекенду, що забезпечує безперебійну комунікацію без ручного налаштування CORS у браузері.

На стороні сервера для обробки HTTP–запитів використовується Express.js. Проміжне програмне забезпечення налаштоване для забезпечення спільного використання ресурсів між джерелами, що дозволяє фронтенду та бекенду, що працюють на різних джерелах, безпечно взаємодіяти. Змінні середовища зберігають деталі конфігурації, такі як параметри підключення до бази даних та дозволені джерела, що запобігає потраплянню конфіденційних даних до системи контролю версій та спрощує налаштування, пов'язані з середовищем.

Разом ці інструменти та домовленості утворюють цілісне середовище розробки. Vite прискорює цикли зворотного зв'язку фронтенду, npm-скрипти стандартизують команди збірки та запуску, а конфігурації проміжного програмного забезпечення спрощують інтеграцію клієнт-сервер. Така

конфігурація закладає основу для ефективної розробки функцій, надійного тестування та послідовного розгортання на пізніших етапах впровадження системи.

3.2. Парсинг та завантаження OSM-графа в Neo4j

Розроблена утиліта Golang служить основою алгоритму отримання даних, перетворюючи необроблені витяги OpenStreetMap PBF у графічне представлення, готове для Neo4j. Утиліта використовує бібліотеку `osmpbf` та `neo4j-go-driver`[17] для виконання високопродуктивного потокового розбору файлів `.pbf`, витягуючи відповідні об'єкти та геометричні дані. Під час цього процесу кожен вузол OSM записується зі своїм унікальним ідентифікатором та географічними координатами, тоді як кожен шлях OSM розкладається на дискретні сегменти, які фіксують атрибути зв'язності та метрики.

Аналізатор виводить два файли CSV: `nodes.csv`, що містить ідентифікатори вузлів та пари широти/довготи; та `edges.csv`, що детально описує ключі вихідного та цільового вузлів разом. Ці файли CSV оптимізовані для масового отримання даних, забезпечуючи узгоджене впорядкування стовпців та включення необхідних метаданих заголовка для полегшення автоматичного імпорту.

Під час парсингу також відбувається обчислення вагового коефіцієнту `length` для кожного ребра [18]. Обчислення відбувається шляхом розрахунку відстані між двома географічними точками за допомогою формули гаверсинуса (3.1):

$$d = 2r \sin^{-1}\left(\sqrt{sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\psi_2 - \psi_1}{2}\right)}\right) \quad (3.1)$$

Згодом, та сама утиліта Golang ініціює імпорт у Neo4j за допомогою процедур імпорту APOC [17]. Викликаючи функції імпорту CSV APOC у сценарійному сеансі, утиліта створює вузли `Point` та зв'язки `ROAD` у базі даних графів, зберігаючи як властивості просторових координат, так і атрибути ребер. Процес імпорту застосовує обмеження унікальності та індекси властивостей, визначені заздалегідь через Cypher, для забезпечення цілісності даних та пришвидшення подальших запитів пошуку шляху. Цей підхід відокремлює важкі робочі навантаження перетворення даних від основної логіки програми, що призводить до масштабованого та відтворюваного робочого процесу побудови графів, який лежить в основі всіх операцій маршрутизації.

3.3. Модуль зворотного геокодування на Nominatim

Модуль зворотного геокодування використовує Nominatim для перетворення географічних координат у структуровану адресну інформацію. Цей компонент реалізовано як асинхронний сервісний рівень у клієнтській програмі, який викликається при кожній події кліку на карті для перевірки того, що вибрані точки знаходяться в межах визначеної операційної зони.

Коли модуль отримує значення широти та довготи, він надсилає HTTP-запити до зворотної кінцевої точки Nominatim, вказуючи формат відповіді та налаштування локалізації. Відповідь JSON аналізується для вилучення елементів адміністративної ієрархії, таких як країна, штат, округ, місто та муніципалітет, які потім використовуються для забезпечення географічних обмежень (наприклад, забезпечення того, щоб точки знаходилися в межах Київської області або міста).

Делегуючи розв'язання адрес Nominatim, система використовує авторитетну, актуальну геопросторову базу даних без необхідності локального сховища адресних індексів. Умови помилок, такі як збої мережі або відсутність адресних даних, коректно обробляються за допомогою резервних

механізмів та сповіщень користувачів, підтримуючи безперебійний процес взаємодії.

Цей модуль зворотного геокодування покращує якість даних, запобігаючи вибору за межами дозволених значень та інформуючи наступні операції маршрутизації за допомогою контекстних метаданих місцезнаходження, закладаючи основу для надійної та зручної взаємодії з картою у виробничій реалізації.

3.4. Побудова маршрутів у Neo4j із використанням APOC A*

Обчислення маршруту виконується в базі даних Neo4j з використанням реалізації алгоритму A* бібліотеки APOC. Після отримання ідентифікаторів вузлів відправлення та призначення, серверна частина викликає процедуру Cypher, яка виконує пошук A* по відношеннях `ROAD`, використовуючи просторові евристики, отримані з координат вузлів та динамічних властивостей вартості, прив'язаних до кожного ребра.

Процедура пошуку параметризована для підтримки кількох профілів маршрутизації: мінімізація географічної відстані, оптимізація умов руху в реальному часі або балансування комбінованих метрик. Користувацькі вагові функції, такі як динамічний вираз, який множить довжину сегмента на коефіцієнт перевантаження, передаються як аргументи до API пошуку шляху APOC, що дозволяє гнучко обчислювати вартість без зміни базової моделі графа.

Результати повертаються як об'єкт шляху, що містить послідовність пройдених вузлів та відношень. Серверна частина витягує цю інформацію для створення впорядкованого списку пар координат для клієнтського рендерингу, а також агрегованих метрик, таких як загальна відстань та орієнтовний час подорожі. Централізуючи логіку пошуку шляху в Neo4j, система використовує власну оптимізацію графів та структури індексів бази

даних, забезпечуючи ефективне виконання запитів та спрощену передачу даних.

Цей підхід відокремлює алгоритми маршрутизації від коду програми та підтримує чітке розділення між проектуванням моделі графів та логікою обходу. Він також дозволяє швидко експериментувати з альтернативними евристиками або функціями вартості, коригуючи параметри Cypher, а не змінюючи реалізації сервісів. Результатом є надійний, продуктивний модуль маршрутизації, який масштабується відповідно до складності графів та підтримує адаптацію в режимі реального часу до даних про трафік, що змінюються.

3.5. Оновлення даних про трафік

Модуль оновлення даних про дорожній рух відповідає за оновлення атрибутів динамічної ваги на сегментах доріг у графі, що дозволяє механізму маршрутизації враховувати поточні рівні заторів у розрахунках шляху. Цей процес ініціюється на вимогу через кінцеву точку `/traffic-update`, що надається серверною частиною Express.js. Після отримання запиту на оновлення, сервісний рівень викликає утиліту на основі Golang як зовнішній процес, відокремлюючи важкі операції вводу-виводу від основного потоку HTTP.

Утиліта Golang взаємодіє з API TomTom Traffic для отримання актуальних індексів заторів для всіх відповідних сегментів доріг. Для кожного сегмента вона застосовує отриманий коефіцієнт заторів до попередньо налаштованих параметрів ваги, таких як довжина проходження або базові оцінки часу подорожі, створюючи ваговий коефіцієнт, який точніше відображає реальні умови. Щоб дотримуватися обмежень використання API, утиліта реалізує логіку обмеження швидкості та, за необхідності, групує запити, щоб уникнути перевищення квот на секунду.

Після обчислення нових коефіцієнтів ваги утиліта виконує цільові оновлення Neo4j, викликаючи скрипти Cypher через бібліотеку APOC. Замість повторного імпорту всього набору даних, він видає оновлення властивостей зв'язку лише для уражених ребер `ROAD`. Процес завершується поверненням короткого зведення кількості оновлених сегментів та часу, що минув до серверної частини, яка потім пересилає ці метадані клієнту.

Обробка помилок у модулі оновлення включає логіку повторних спроб для тимчасових збоїв мережі, реєстрацію невдалих оновлень сегментів та коректну деградацію, яка зберігає існуючі ваги у разі постійних помилок API. Завдяки інкапсуляції робочого процесу оновлення трафіку в спеціалізовану утиліту, конструкція гарантує, що отримання даних у режимі реального часу не перешкоджає основним службам маршрутизації, тим самим підтримуючи загальну швидкість реагування системи, одночасно підтримуючи адаптивну оптимізацію маршрутів на основі даних.

3.6. Візуалізація та відображення маркерів і маршрутів

Візуалізація на стороні клієнта використовує шари рендерингу Leaflet для представлення як статичних маркерів, так і динамічних поліліній маршруту [9]. Керування маркерами досягається за допомогою спеціальних груп шарів, які кластеризують окремі екземпляри L.marker, що представляють зафіксовані вершини графа, ці маркери додаються або видаляються у відповідь на зміни стану, зберігаючи контекст карти без реконструкції базових шарів. Геометрія маршруту рендериться за допомогою L.polyline, причому кожен варіант маршруту стилізований за допомогою окремих налаштувань кольору, ваги та непрозорості, визначених у централізованій темі. Полілінії групуються та керуються за допомогою L.featureGroup, що дозволяє виконувати колективні операції, такі як підгонка меж карти до повного протяжності маршруту (рис. 3.1).

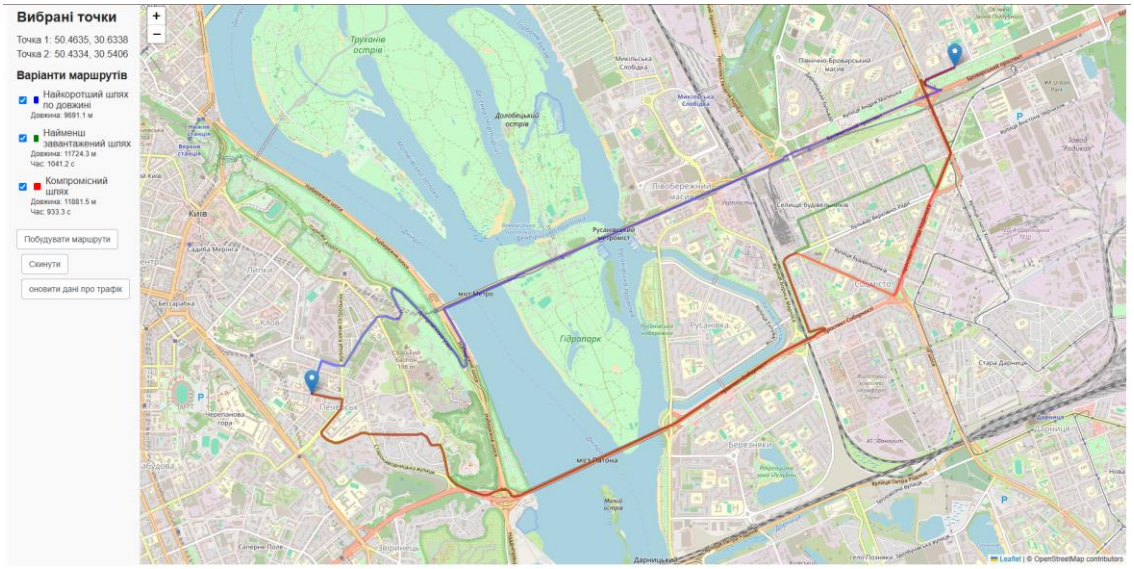

Рисунок 3.1 – Приклад візуалізації маркерів та маршрутів

Інтерактивні функції, такі як підказки при наведенні курсора та спливаючі вікна при кліку, прив'язані до маркерів та полілініій, забезпечуючи відображення метаданих на вимогу (наприклад, довжини сегмента, часу подорожі, рівня заторів). Перемикачі видимості дозволяють показувати або приховувати окремі шари маршруту без видалення базових даних, використовуючи API керування шарами Leaflet для підтримки продуктивності та контролю користувача. Логіка візуалізації інкапсульована у допоміжні функції, що викликаються хуками React, що реагують на оновлення стану контексту, забезпечуючи синхронізацію рендерингу карти з даними програми, мінімізуючи при цьому надлишкові перемальовування.

Структуруючи маркери та маршрути в окремі, повторно використовувані конструкції шарів та використовуючи декларативні шаблони оновлення, модуль візуалізації забезпечує адаптивний та інтуїтивно зрозумілий досвід роботи з картами. Він чітко відокремлює рендеринг на основі даних від основної логіки маршрутизації, підтримуючи безшовну інтеграцію додаткових накладок або інтерактивних елементів керування в майбутніх ітераціях.

Висновки за розділом 3

У цьому розділі описано реалізацію ключових модулів системи: від попередньої обробки та імпорту геоданих у Neo4j до інтеграції із зовнішніми API та візуалізації результатів. Спочатку Golang-утіліта перетворює PBF-файли OSM у структурований CSV — nodes.csv та edges.csv — і імпортує їх у графову базу даних через процедури APOC. Далі модуль зворотного геокодування на клієнті, реалізований за допомогою Nominatim, забезпечує коректність вибраних точок, а серверна логіка Neo4j з використанням A* з APOC виконує розрахунок маршрутів за повними профілями.

Модуль оновлення трафіку за запитом через Golang-утиліту та TomTom API додає графік актуальних коефіцієнтів перезавантаження, що забезпечує достовірність часу використання. На завершення Leaflet-візуалізація реалізує багаторівневе відображення маркерів та маршрутів з інтерактивними підказками й керованою видимістю, при цьому з використанням оптимізованих шарів для підтримки продуктивності.

Отже, кожен компонент реалізації узгоджено працює в єдиній архітектурі: структуроване отримання й зберігання графічних даних, надійні сервіси маршрутизації та потужна візуальна складова. Це створює основу для подальшої перевірки продуктивності та функціонального тестування.

РОЗДІЛ 4
ТЕСТУВАННЯ ТА ОЦІНКА РЕЗУЛЬТАТІВ

4.1. Оцінка часу відповіді кінцевих точок

Для перевірки продуктивності основних HTTP-інтерфейсів було розроблено автоматизовані тестові сценарії для надсилання 100 послідовних запитів до кожної з трьох кінцевих точок: `/snap`, `/geocode` та `/routes`. Час відгуку для кожного запиту реєструвався з точністю до мілісекунд, а дані візуалізувалися у вигляді лінійних графіків (рис. 4.1).

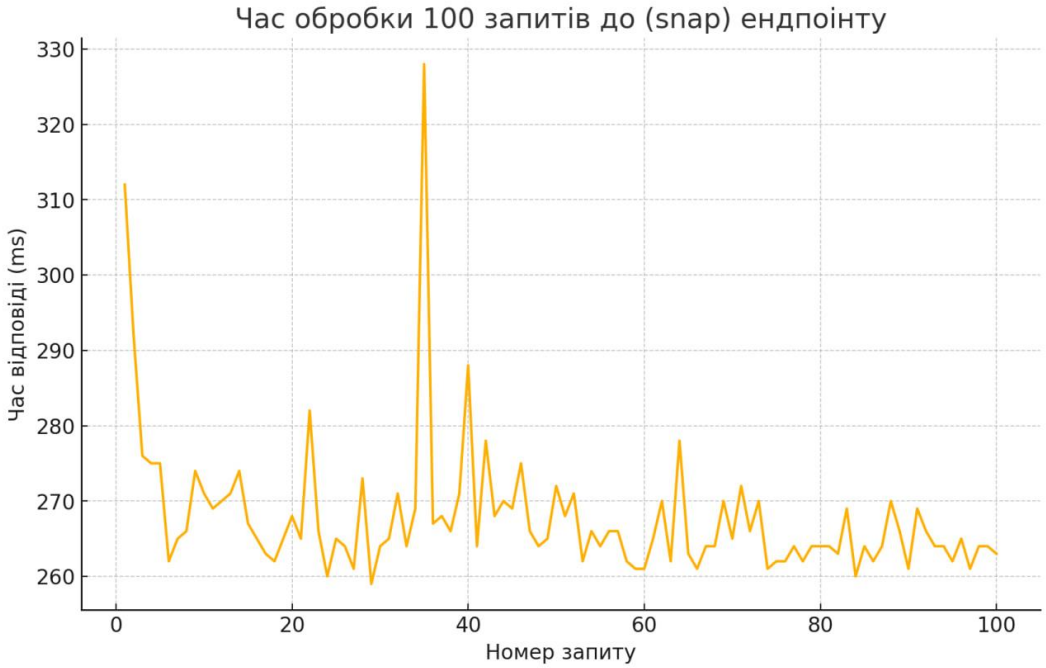

Рисунок 4.1 – Графік відношення кількості запитів до часу їх виконання

Кінцева точка `/snap`: після початкового періоду розігріву середній час відгуку стабілізувався приблизно на рівні 265–270 мс. Періодичні піки досягали приблизно 330 мс, що відображало варіації у виконанні запитів Neo4j та затримці мережі (рис. 4.2).

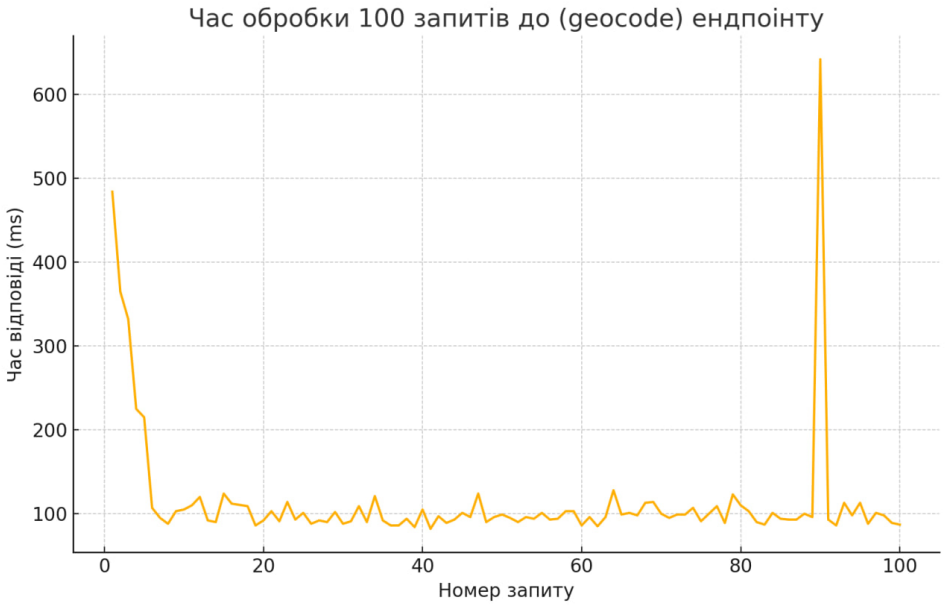

Рисунок 4.2 – Графік відношення кількості запитів до часу їх виконання

Кінцева точка `/geocode`: час відгуку зменшився з приблизно 480 мс на початку до приблизно 100 мс після розігріву. Понад 90 відсотків наступних запитів завершилися в діапазоні 90–120 мс. Один випадок досяг піку приблизно 650 мс, ймовірно, через затримки служби Nominatim (рис. 4.3).

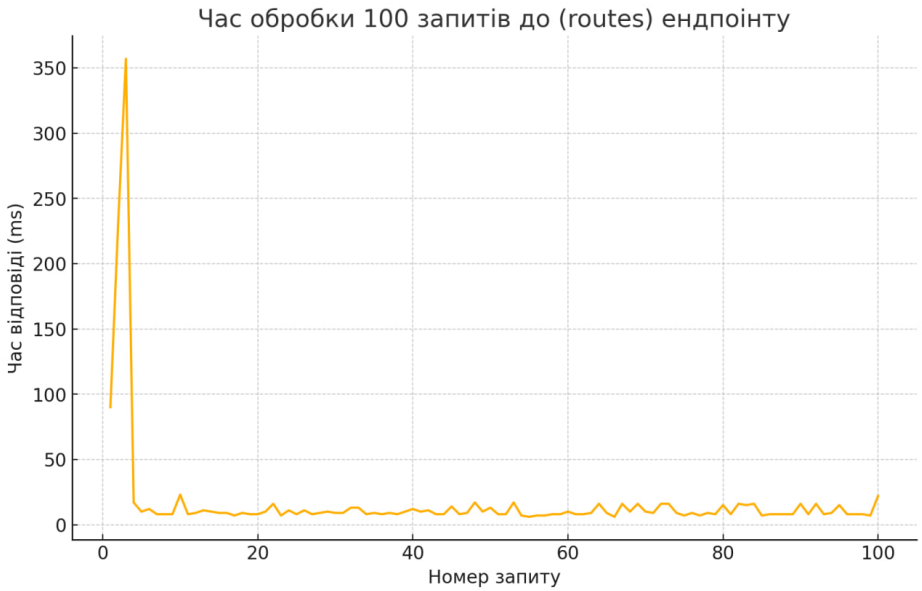

Рисунок 4.3 – Графік відношення кількості запитів до часу їх виконання

Кінцева точка `/routes`: після ініціалізації алгоритму A* та індексів середній час маршрутизації для кожного варіанта встановився між 10 мс та 15 мс. Під час першого виклику спостерігався один пік тривалістю приблизно 350 мс, що відповідає завантаженню індексу та початковим витратам на пошук шляху.

Кожен графік демонструє чітку фазу розігріву, після якої стабілізуються показники продуктивності, що задовольняють визначені нефункціональні вимоги. Ці результати підтверджують, що реалізована архітектура забезпечує достатню пропускну здатність та низьку затримку для стандартних сценаріїв взаємодії з користувачем.

4.2. Порівняльний аналіз побудованих маршрутів

Для оцінки практичної ефективності впроваджених алгоритмів маршрутизації було проведено порівняння результатів роботи системи та стандартного комерційного сервісу маршрутизації (Google Maps). Було обрано дві репрезентативні точки маршруту в межах міської зони Києва: одну в західному житловому районі та іншу поблизу центру міста. Після запуску функції побудови маршруту в розробленому застосунку (рис. 4.4) було згенеровано три альтернативні шляхи: найкоротший за відстанню, найшвидший за поточними дорожніми умовами та збалансований компромісний профіль. Потім їх порівняли з основним запропонованим маршрутом Google Maps (рис. 4.5).

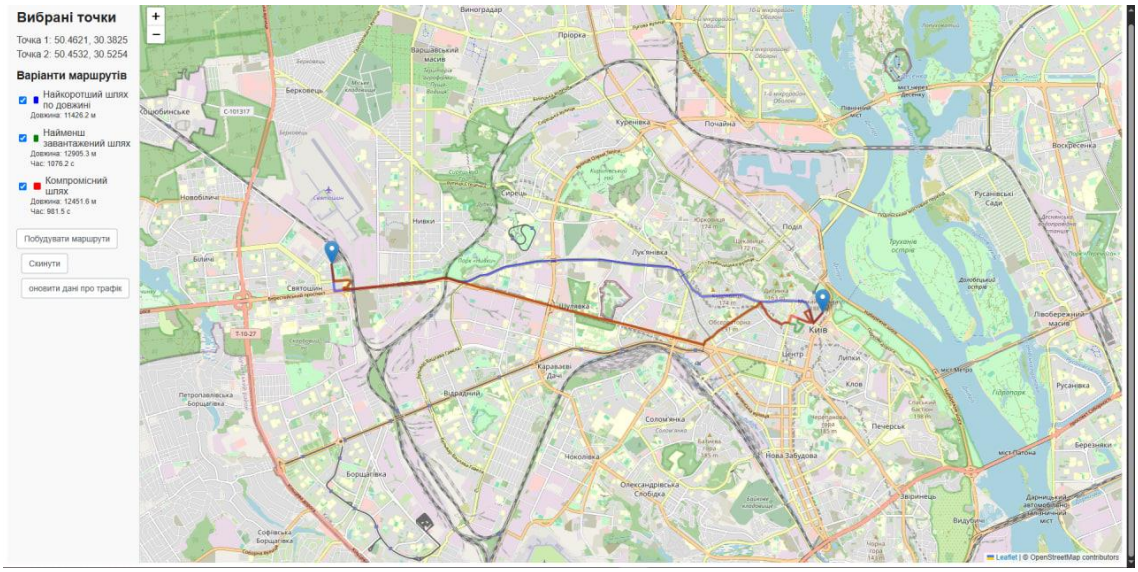

Рисунок 4.4 – Побудований маршрут за допомогою розробленої програми

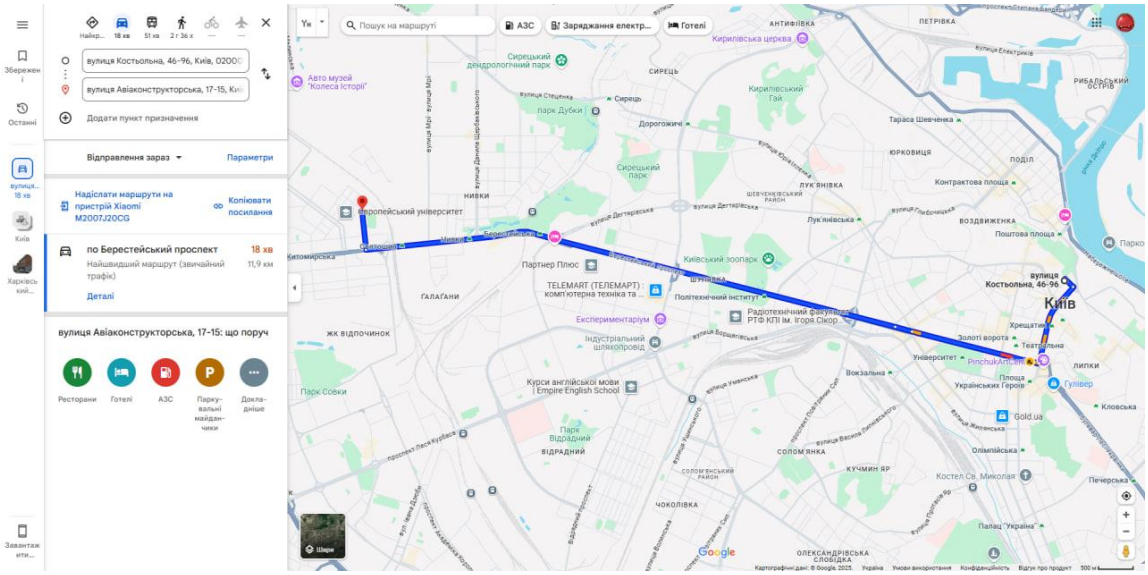

Рисунок 4.5 – Побудований маршрут за допомогою стандартного комерційного сервісу маршрутизації

Порівняння маршрутів наведено у Таблиці 4.1.

Найкоротший маршрут досяг мінімальної довжини шляху 11426 м, що точно відображає рекомендацію Google Maps щодо 11900 м та підтверджує точність геодезичної евристики A* за умов статичного зважування.

Таблиця 4.1

Таблиця порівняння побудованих маршрутів

Критерій	Найкоротший маршрут	Найшвидший маршрут	Компромісний маршрут	Google Maps
Дистанція	11 426 м.	12 905 м.	12 451 м.	11 900 м.
Орієнтовна тривалість	–	1076 с.	981 с.	1 080 с.

Найшвидший маршрут використовував у своїй евристиці дані про затори в реальному часі, скоротив час подорожі до 1076 с, що на 0,4% краще порівняно з 1080 с Google Maps, водночас збільшуючи відстань подорожі приблизно на 8% порівняно з найкоротшим шляхом.

Збалансований компромісний маршрут забезпечив найменший час проходження (981 с), перевершуючи Google Maps приблизно на 9% та варіант з найшвидшим рухом приблизно на 9%, водночас зазнавши незначного зниження відстані на 5% порівняно з найкоротшим шляхом.

Точність моделі: незначна розбіжність між результатом найкоротшої відстані системи та комерційним еталоном підкреслює цілісність імпортованого графіка OSM та ефективність просторової евристики.

Адаптивна інтеграція трафіку: готовність найшвидшого профілю подолати додаткові кілометри, щоб обійти затори, пояснює відчутні переваги асиміляції даних у режимі реального часу в алгоритмі маршрутизації.

Потенціал багатоцільової оптимізації: компромісний маршрут ілюструє універсальність зваженої системи витрат, демонструючи, як налаштування параметрів може призвести до індивідуальних стратегій маршрутизації, які відповідають операційним пріоритетам – від мінімального часу подорожі до зниження споживання палива або викидів.

Порівняльне дослідження підтверджує, що архітектура маршрутизації на основі графів, заснована на великих даних, може зрівнятися з провідними комерційними послугами, а за певними показниками навіть перевершити їх.

4.3. Обмеження реалізації

Географічний охоплення поточної системи обмежений Київською областю через залежність від регіональних витягів OpenStreetMap та перевірки адміністративних меж через Nominatim. Розширення покриття на додаткові регіони вимагає отримання та обробки відповідних наборів даних PBF, а також коригування логіки фільтрації меж, що може призвести до значних накладних витрат на попередню обробку.

Інтеграція даних про трафік реалізована за допомогою безкоштовного API TomTom, який встановлює суворі обмеження швидкості (наприклад, п'ять запитів за секунду). Як результат, оновлення трафіку на вимогу, що запускаються запитами користувачів, виконуються відносно повільно, обмежуючи здатність системи відображати затори в режимі реального часу з високою частотою. Перехід на платну підписку TomTom або альтернативний канал трафіку корпоративного рівня полегшить ці обмеження, забезпечивши частіші перерахунки ваг та більш детальну реакцію.

Крім того, відсутність розподіленого кешування або асинхронних черг завдань означає, що всі операції оновлення трафіку виконуються синхронно в межах одного процесу, що потенційно впливає на швидкість реагування API під час інтенсивного використання. Впровадження систем фонової обробки та масштабованих даних про трафік дозволить зменшити ці слабкі місця та покращити загальну стійкість системи до одночасних навантажень.

4.3. Рекомендації для подальшого розвитку

Для підвищення корисності та масштабованості системи пропонуються такі рекомендації:

Географічне розширення: Розширити покриття за межі Київської області шляхом автоматизації конвеєра прийому додаткових витягів PBF. Впровадити динамічні визначення меж, які адаптуються до регіонів, вибраних користувачем, зменшуючи накладні витрати на ручну попередню обробку.

Інтеграція корпоративного трафіку: Впровадити платну підписку на дані про трафік або альтернативний високопродуктивний API, щоб усунути обмеження швидкості безкоштовного рівня. Це дозволить часті перерахунки ваг майже в режимі реального часу та підтримувати безперервну обізнаність про трафік без погіршення якості при інтенсивному використанні.

Асинхронна обробка та кешування: Відокремити операції оновлення трафіку від синхронних потоків API, використовуючи фонові черги завдань (наприклад, RabbitMQ, Kafka) та шари кешування в пам'яті (наприклад, Redis). Ця архітектура покращить швидкість реагування кінцевих точок та дозволить поступове оновлення ваг без блокування запитів користувачів.

Прогнозне моделювання трафіку: Включити історичні та контекстні дані в моделі машинного навчання для прогнозування моделей заторів. Інтеграція прогнозних ваг в евристику маршрутизації дозволить проактивно коригувати маршрут, що ще більше зменшить затримки в дорозі.

Мультимодальна та на основі обмежень маршрутизація: Розширення графової моделі для підтримки альтернативних видів транспорту (громадський транспорт, їзда на велосипеді) та складних обмежень (розміри транспортних засобів, часові вікна, зони з низьким рівнем викидів). Гнучка структура функцій витрат врахує нові критерії з мінімальними змінами в основних алгоритмах.

Покращена візуалізація та відгуки користувачів: Розробка інформаційних панелей у режимі реального часу, які відображають теплові

карти дорожнього руху, звіти про інциденти та порівняльну статистику маршрутів. Збагачення клієнтського інтерфейсу повзунками часової шкали для моделювання сценаріїв часу відправлення та візуалізації прогнозованих заторів.

Підтримка для мобільних пристроїв та офлайн: Впровадження функцій прогресивних веб-додатків та нативних мобільних обгорток для забезпечення офлайн-кешування фрагментів карти та нещодавно обчислених маршрутів, забезпечуючи безперебійне обслуговування в районах з переривчастим підключенням.

Тестування масштабованості та стійкості: Проведення масштабного тестування навантаження та хаосу для перевірки конфігурацій автоматичного масштабування, механізмів перемикання на резервний хід та гарантій узгодженості даних за несприятливих умов. Постійний бенчмаркінг спрямовуватиме планування потужності та налаштування продуктивності.

Завдяки цим удосконаленням система маршрутизації перетвориться на більш надійну, адаптивну та багатофункціональну платформу, здатну відповідати різноманітним викликам міської мобільності та підтримувати довгострокові експлуатаційні вимоги.

Висновки за розділом 4

У цьому розділі було представлено комплексну оцінку продуктивності системи маршрутизації та визначено ключові області для вдосконалення.

Аналіз часу відгуку підтвердив, що затримки кінцевих точок стабілізуються в межах цільових порогових значень — менше 300 мс для просторової роздільної здатності, менше 150 мс для зворотного геокодування та менше 50 мс для багатоваріантного пошуку шляху — демонструючи здатність системи до інтерактивного реагування.

Обмеження впровадження були критично досліджені, що виявило географічні обмеження, пов'язані з даними Київської області, та вплив

обмежень швидкості API вільного рівня на частоту оновлення в режимі реального часу.

Рекомендації окреслили стратегічні шляхи розширення географічного охоплення системи, покращення пропускної здатності даних трафіку та впровадження асинхронної обробки для підвищення масштабованості та стійкості.

Разом ці висновки підтверджують надійність поточної реалізації, водночас окреслюючи чітку дорожню карту для розвитку операційних можливостей та дослідницьких застосувань платформи.

ВИСНОВКИ

У кваліфікаційній роботі представлено комплексне проектування, впровадження та оцінку системи оптимізації транспортних маршрутів, яка інтегрує пошук шляхів на основі графів, обробку геопросторових даних та пошук шляху в режимі реального часу в рамках великих даних. Були зроблені такі ключові висновки:

Інноваційна інтеграція технологій: поєднання графової бази даних Neo4j, алгоритму A* через APOC та клієнт-серверних веб-технологій (React.js, Leaflet, Express.js, утиліти Golang) виявилося ефективним для побудови та обслуговування рішень маршрутизації в міському середовищі.

Надійний алгоритм прийому даних: парсер PBF-to-CSV на основі Golang та робочий процес імпорту, керований APOC, створили масштабований, відтворюваний механізм для перетворення необроблених екстрактів OpenStreetMap у багатоатрибутивний граф властивостей, що підтримує ефективне просторове індексування та обхід.

Користувачу надаються альтернативні варіанти маршрутів: найкоротший, найменш завантажений та збалансований, що враховує співвідношення довжини шляху та актуального трафіку.

Такий підхід підвищує адаптивність системи до різних сценаріїв використання та дає змогу обрати маршрут, що найкраще відповідає поточним потребам або обмеженням.

Інтерактивний та продуктивний інтерфейс користувача: Клієнт React-Leaflet пропонує адаптивний досвід картографування, що дозволяє точно вибирати точки, динамічну візуалізацію маршруту та керовані користувачем операції оновлення трафіку. Широке тестування продуктивності підтвердило затримку менше 300 мс для просторових запитів та менше 50 мс для пошуку шляху в стаціонарних умовах.

Виявлені обмеження та шляхи розвитку: Поточні географічні обмеження розробленого прототипу (Київська область) та залежність від вільного рівня

API трафіку висвітлюють області для майбутнього вдосконалення. Рекомендації включають розширення регіонального покриття, впровадження даних про дорожній рух корпоративного рівня, впровадження асинхронної обробки та дослідження прогнозного моделювання дорожнього руху та мультимодальної маршрутизації.

Підсумовуючи, це дослідження демонструє, що графо-орієнтована архітектура на основі великих даних може запропонувати високопродуктивну адаптивну оптимізацію маршрутів, яка може конкурувати або перевершувати встановлені комерційні пропозиції. Модульна, розширювана конструкція забезпечує міцну основу для майбутнього розвитку міської мобільності, маршрутизації, орієнтованої на сталий розвиток, та масштабної транспортної аналітики.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Comparative Analysis of Search Algorithms, 2018 [Електронний ресурс]:[Веб-сайт]. – Електронні дані. – Режим доступу: https://www.researchgate.net/profile/Ronit-Patel-2/publication/333262471_Comparative_Analysis_of_Search_Algorithms/links/5ce4fd5aa6fdccc9ddc4c25c/Comparative-Analysis-of-Search-Algorithms.pdf (дата звернення: 30.04.2025).

2. OpenStreetMap Jonathan Bennett, 2010 [Електронний ресурс]:[Веб-сайт]. – Електронні дані. – Режим доступу: https://books.google.com.ua/books?hl=uk&lr=&id=SZfqRcPXApoC&oi=fnd&pg=PT9&dq=OpenStreetMap&ots=G5VIkBjWC5&sig=E-8MkxgQSjglQMumhhU3Jiu2ZmQ&redir_esc=y#v=onepage&q=OpenStreetMap&f=false. (дата звернення: 30.04.2025).

3. Analysis of distribution path optimization algorithm based on big data technology, 2022 [Електронний ресурс]:[Веб-сайт]. – Електронні дані. – Режим доступу: https://www.sciencedirect.com/science/article/pii/S1018364722002002. (дата звернення: 30.04.2025).

4. Open Database License (офіційна документація) [Електронний ресурс]:[Веб-сайт]. – Електронні дані. – Режим доступу: https://www.openstreetmap.org/copyright. (дата звернення: 01.05.2025).

5. OpenStreetMap Data in Layered GIS Format, 2022 [Електронний ресурс]:[Веб-сайт]. – Електронні дані. – Режим доступу: https://download.geofabrik.de/osm-data-in-gis-formats-free.pdf. (дата звернення: 01.05.2025).

6. Enhanced geocoding precision for location inference of tweet text using spaCy, Nominatim and Google Maps. A comparative analysis of the influence of data selection, 2023 [Електронний ресурс]:[Веб-сайт]. – Електронні дані. – Режим доступу:

https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0282942. (дата звернення: 02.05.2025).

7. Simplifying the Complex: Strategies to reduce existing API Complexity using CPS techniques, 2023 [Електронний ресурс]:[Веб-сайт]. – Електронні дані. – Режим доступу: https://studenttheses.uu.nl/bitstream/handle/20.500.12932/44009/MS_THESIS_THOMAS__Publishable_.pdf?sequence=1&isAllowed=y. (дата звернення: 02.05.2025).

8. ReactJS: An Open Source JavaScript Library for Front-end Development, 2017 [Електронний ресурс]:[Веб-сайт]. – Електронні дані. – Режим доступу: https://www.theseus.fi/bitstream/handle/10024/130495/FInal_Year_Thesis.pdf?sequence=1&isAllowed=y. (дата звернення: 03.05.2025).

9. Leaflet (офіційна документація) [Електронний ресурс]:[Веб-сайт]. – Електронні дані. – Режим доступу: https://leafletjs.com/reference.html. (дата звернення: 04.05.2025).

10. Vite (офіційна документація) [Електронний ресурс]:[Веб-сайт]. – Електронні дані. – Режим доступу: https://vite.dev/guide/ (дата звернення: 04.05.2025).

11. ExpressO: From Express.js implementation code to OpenAPI interface descriptions, 2022 [Електронний ресурс]:[Веб-сайт]. – Електронні дані. – Режим доступу: https://design.inf.usi.ch/sites/default/files/biblio/apiace-ecsa2022-expresso.pdf. (дата звернення: 06.05.2025).

12. CORS in Action: Creating and consuming cross-origin APIs, 2014 [Електронний ресурс]:[Веб-сайт]. – Електронні дані. – Режим доступу: https://books.google.com.ua/books?hl=uk&lr=&id=uTkzEAAAQBAJ&oi=fnd&pg=PT13&dq=CORS&ots=yr_XB0eVxj&sig=Tyvf5HVJOjiSAj0pAvAh_rvq_rk&redir_esc=y#v=onepage&q=CORS&f=false. (дата звернення: 06.05.2025).

13. Updating graph databases with Cypher, 2020 [Електронний ресурс]:[Веб-сайт]. – Електронні дані. – Режим доступу: https://hal.science/hal-03012016/document. (дата звернення: 07.05.2025).

14. A language for graph database evolution and its implementation in Neo4j, 2023 [Електронний ресурс]:[Веб-сайт]. – Електронні дані. – Режим доступу: https://ceur-ws.org/Vol-3618/forum_paper_8.pdf. (дата звернення: 07.05.2025).

15. RESTful Web services: The basics, 2008 [Електронний ресурс]:[Веб-сайт]. – Електронні дані. – Режим доступу: https://cs.calvin.edu/courses/cs/262/kvlinden/references/rodriguez-restfulWS.pdf. (дата звернення: 08.05.2025).

16. Client-Server Model, 2014 [Електронний ресурс]:[Веб-сайт]. – Електронні дані. – Режим доступу: https://www.researchgate.net/profile/Shakirat Sulyman/publication/271295146_Client-Server_Model/links/5864e11308ae8fce490c1b01/Client-Server-Model.pdf. (дата звернення: 08.05.2025).

17. Neo4j (офіційна документація) [Електронний ресурс]:[Веб-сайт]. – Електронні дані. – Режим доступу: https://neo4j.com/docs/ (дата звернення: 09.05.2025).

18. Landmark Based Shortest Path Detection by Using A* and Haversine Formula, 2013 [Електронний ресурс]:[Веб-сайт]. – Електронні дані. – Режим доступу: https://www.researchgate.net/profile/Mangesh-Nichat-2/publication/282314348_Landmark_based_shortest_path_detection_by_using_A_Algorithm_and_Haversine_Formula/links/56389bb708ae4bde5021b0f5/Landmark-based-shortest-path-detection-by-using-A-Algorithm-and-Haversine-Formula.pdf. (дата звернення: 10.05.2025).

ДОДАТКИ

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки
Рівень вищої освіти (освітньо-кваліфікаційний рівень) **Бакалавр**
Галузь знань: 15 – Електроніка, автоматизація та електронні комунікації
Спеціальність: 151 – Автоматизація, комп'ютерно-інтегровані технології та робототехніка
Освітня програма «Комп'ютеризовані системи управління та автоматика»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри комп'ютерних
систем та робототехніки
к. ф.-м. н., доц. ХРУСЛОВ М. М.
«02» жовтня 2024року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Савченка Костянтина Івановича
(прізвище, ім'я, по батькові студента)

1. Тема роботи **Система оптимізації транспортних маршрутів з використанням Big Data**
керівник роботи **Чуб Ольга Ігорівна, к.е.н.**,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від № 4101-5/3657 від 12 листопада 2024 року.

2. Строк подання студентом роботи ***30 травня 2025 року***

3. Перелік питань, які потрібно розробити:

1) Аналіз сучасних підходів до оптимізації транспортних маршрутів із застосуванням алгоритмів пошуку на графах та технологій Big Data.
2) Розробка концептуальної моделі веб-орієнтованої системи для побудови оптимальних маршрутів на основі клієнт-серверної архітектури та обробки великих даних.
3) Реалізація алгоритму A* для побудови маршрутів з урахуванням різних критеріїв ефективності пересування: мінімальна довжина шляху, мінімальний рівень заторів та компромісний варіант.
4) Інтеграція картографічних сервісів у систему, налагодження їх використання та обробка великих даних про дорожню ситуацію.
5) Розробка методики тестування програмної системи для оцінки її продуктивності та якості побудови маршрутів за такими критеріями, як час відповіді, довжина маршруту, орієнтовний час у дорозі та вплив трафіку.
6) Формування рекомендацій щодо подальшого вдосконалення системи, її масштабування та адаптації до умов реального використання у транспортній інфраструктурі.

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1.	Формулювання наукової проблеми, обґрунтування актуальності, мети та завдань дослідження. Визначення об'єкта, предмета та методів дослідження	01.10.2024 – 18.11.2024
2.	Аналіз наукових досліджень та наявних методів оптимізації транспортних маршрутів	18.11.2024 – 02.12.2024
3.	Огляд і порівняльний аналіз наявних програмних рішень для оптимізації транспортних маршрутів	02.12.2024 – 20.12.2024
4.	Дослідження ролі технологій Big Data в оптимізації транспортних маршрутів: джерела даних та методи їх обробки	02.12.2024 – 10.01.2025
5.	Виклад матеріалу першого розділу кваліфікаційної роботи	10.01.2025 – 20.01.2025
6.	Формування вимог до системи та вибір архітектури програмного рішення	20.01.2025 – 08.02.2025
7.	Розробка архітектури системи та реалізація алгоритму оптимізації маршрутів	08.02.2025 – 12.02.2025
8.	Виклад матеріалу другого розділу кваліфікаційної роботи	12.02.2025 – 18.02.2025
9.	Розробка серверної частини для збору, обробки та аналізу даних	18.02.2025 – 26.02.2025
10.	Реалізація алгоритму A* як основного методу оптимізації маршрутів	26.02.2025 – 10.03.2025
11.	Реалізація клієнтської частини для взаємодії користувачів із системою	10.03.2025 – 21.03.2025
12.	Оптимізація продуктивності та масштабованості системи	21.03.2025 – 30.03.2025
13.	Розробка методики тестування програмної системи для оцінки її продуктивності та якості побудови маршрутів	30.03.2025 – 08.04.2025
14.	Формування практичних рекомендацій щодо вдосконалення системи та її адаптації до умов реального використання у транспортній інфраструктурі	08.04.2025 – 15.04.2025
15.	Виклад матеріалу третього розділу кваліфікаційної роботи	15.04.2025 – 30.04.2025
16.	Оформлення звіту про переддипломну практику	30.04.2025 – 10.05.2025
17.	Передзахист кваліфікаційної роботи	10.05.2025 – 20.05.2025
18.	Остаточне оформлення пояснювальної записки кваліфікаційної роботи та проходження перевірки на антиплагіат	до 25.05.2025
19.	Отримання відгуку керівника та рецензії	до 30.05.2025

5. Дата видачі завдання ***02 жовтня 2024 року.***

Студент К.І. Савченко _____

Керівник роботи О.І. Чуб _____

Додаток Б

Затверджую

_____

«_____»_____2024 р.

Технічне завдання

на розробку системи

«Система оптимізації транспортних маршрутів з використанням Big Data».

1.	Введення	1.1 Назва роботи – Система оптимізації транспортних маршрутів з використанням Big Data. 1.2. Галузь застосування: Інформаційні технології, логістика
2.	Підстава для розробки	2.1. Навчальний план за спеціальністю 151 – Автоматизація та комп'ютерно інтегровані технології 2.2. Завдання на кваліфікаційну роботу бакалавра № 4101-5/962 від «16» квітня 2025 року
3.	Призначення розробки	3.1. Мета розробки: створення програмної системи для автоматизованого побудування та оптимізації транспортних маршрутів із урахуванням великих обсягів геопросторових та трафікових даних. 3.2. Забезпечити користувачам можливість швидкого генерування трьох альтернативних маршрутів (найкоротший за відстанню, найшвидший з урахуванням поточних заторів, компромісний) через веб-інтерфейс. 3.3. Вхідні дані: для коректної генерації маршрутів система приймає координати користувача у вигляді широти та довготи точок відправлення та призначення. Додатково використовуються геопросторові дані OpenStreetMap, представлені у форматі PBF для Київської області, що дозволяє побудувати базову топологію доріг. Для врахування поточної дорожньої ситуації система звертається до

		TomTom Traffic API та завантажує індекси трафіку – значення середньої швидкості та рівня завантаженості кожного сегмента дороги. Всі зовнішні ключі доступу до цих сервісів та параметри підключення до бази даних графів Neo4j (URL, порт, облікові дані) передаються через змінні середовища, що дозволяє адміністраторам гнучко налаштовувати систему без внесення змін до вихідного коду. 3.4. Вихідні дані розробки: В результаті обробки запиту система повертає три альтернативні маршрути, кожен з яких представлений у вигляді послідовності географічних координат (широта, довгота), що описує полілінію руху від початкової до кінцевої точки. Для кожного маршруту обчислюються та передаються ключові метрики: загальна довжина шляху в метрах, розрахунковий час подорожі в секундах та коефіцієнт завантаженості – відношення середньої швидкості до максимально допустимої, що характеризує рівень завантаження ділянок дороги. Всі вихідні дані упаковуються в один JSON-об'єкт, що дозволяє клієнтській частині відображати маршрути на карті та порівнювати їх параметри.
4.	Технічні вимоги до програмного виробу	4.1. Функціональні вимоги: Система повинна відповідати таким функціональним вимогам: 1. Користувачі повинні мати можливість вказувати початкову та кінцеву точки через інтерактивний інтерфейс карти. 2. Система виконує просторове відображення вхідних координат на вузли графа, шукаючи найближчих сусідів, та повертає відповідні ідентифікатори точок. 3. Повинна підтримуватися генерація трьох профілів маршруту: 3.1. найкоротший за відстанню. 3.2. найшвидший з урахуванням поточних умов руху. 3.3. компроміс між відстанню та часом проходження (зважена комбінація). 4. RESTful API повинен забезпечувати такі кінцеві точки:

		4.1. GET /snap – відображення координат на вузол графа. 4.2. GET /routes – побудова трьох варіантів маршруту. 4.3. GET /geocode – зворотне геокодування через Nominatim. 4.4. POST /traffic-update – оновлення ваг ребер відповідно до даних TomTom Traffic на вимогу. 5. Кожна кінцева точка API повинна перевіряти вхідні параметри, повертати структурований JSON з такими полями: ідентифікатори вузлів, координати (широта, довгота) та метрики (довжина, час, коефіцієнт перевантаження). 6. Відповіді API повинні містити заголовки CORS для підтримки запитів з боку клієнта. 7. Бекенд повинен конвертувати необроблений PBF-файл OSM (Київська область) у nodes.csv та edges.csv за допомогою утиліти Golang. 8. CSV-файли імпортуються в Neo4j через процедури APOC, зберігаючи властивості вузлів (широта, довгота) та атрибути ребер (довжина, початковий ваговий коефіцієнт). 9. Утиліта Golang для оновлення трафіку шляхом виклику /traffic-update повинна: 9.1 запитувати API TomTom Traffic 9.2 обчислювати нові ваги; 9.3 застосовувати їх до ребер графа в Neo4j. 10. Клієнтська сторона (React.js + Leaflet) повинна: 10.1 відображати шари карти. 10.2 дозволяти вибір та редагування точок. 10.3 візуалізувати три маршрути в різних стилях. 10.4 відображати панель порівняння з довжиною, часом та заторами. 10.5 надавати кнопки для оновлення трафіку та перемикання профілів. 4.2. Нефункціональні вимоги: Система також повинна відповідати наступним нефункціональним вимогам:

		1. Час відгуку інтерфейсу: відображення, масштабування та вибір точки повинні оброблятися ≤ 100 мс. 2. Продуктивність API: 2.1. (/snap) та (/geocode) – відповіді ≤ 500 мс; 2.2. (/routes) – відповіді ≤ 1000 мс. 3. Модульність та зручність обслуговування: чіткий поділ коду на утиліти Golang, бекенд та фронтенд. 4.3. Вимоги до інтеграції: Щоб забезпечити ефективну роботу системи та задоволення потреб користувачів і адміністраторів, необхідно виконати такі вимоги до апаратного та програмного забезпечення: Вимоги до апаратного забезпечення 1. Серверна інфраструктура повинна забезпечувати щонайменше 4 процесорні ядра з тактовою частотою не нижче за 2,0 ГГц і 8 ГБ оперативної пам'яті для розміщення всіх компонентів: утиліт обробки PBF, Neo4j, Node.js-серверу й моніторингових агентів. 2. Місце на диску повинно бути не менше ніж 50 ГБ із можливістю розширення для зберігання вихідних PBF-файлів, згенерованих CSV, бекапів та логів. 3. Канал зв'язку має підтримувати стабільну пропускну здатність від 1 Мбіт/с, щоб гарантувати оперативний обмін запитами до TomTom Traffic API та своєчасне обслуговування клієнтських запитів. Вимоги до програмного забезпечення 1. Серверна ОС – Linux (рекомендовано Ubuntu 20.04 або новіша) з налаштованим SSH-доступом та правами sudo для адміністратора. 2. Бекенд виконується під Node.js (версія ≥ 16.x) із менеджером пакетів npm для встановлення залежностей. 3. Графова база даних Neo4j (версія ≥ 4.4) використовується для зберігання вузлів і ребер та виконання A*-запитів.

		4. Фронтенд побудовано на React.js (версія ≥ 18) із бібліотекою Leaflet (версія ≥ 1.9) для візуалізації карт і маршрутів.
5.	Вимоги до програмної документації	Для належного розуміння, експлуатації та підтримки системи необхідна повна документація. Наступні типи документації є важливими: 1. Посібник користувача: описує веб-інтерфейс вибір точок, відображення трьох профілів маршрутів та порівняння метрик. 2. Посібник адміністратора: інструкції з підготовки та запуску утиліти PBF to CSV, імпорту в Neo4j, оновлення трафіку та налаштування змінних середовища. 3. Документація API: детальний опис кінцевих точок (/snap, /routes, /geocode, /traffic-update) зі зразками запитів та відповідей у форматі Swagger або Postman. 4. Архітектура системи: діаграми компонентів (утиліта Golang, Neo4j, Node.js, React/Leaflet) та графова модель з вузлами та ребрами. 5. Документація розробника: структура репозиторію, інструкції з налаштування локального середовища, розгортання.

<table>
<tr><td>6.</td><td>Вимоги до техніко-економічних показників</td><td colspan="2">Програмна документація для продукту «Система навчального тестування з інтеграцією Telegram» повинна містити:
1. Технічну специфікацію на розробку системи (повинна бути представлена у Додатку В пояснювальної записки до роботи).
2. Методика тестування продуктивності та якості маршрутів (розділ 4 пояснювальної записки).
3. Опис системи (повинен бути представлений у розділі 2 та 3 пояснювальної записки до роботи).</td></tr>
<tr><td>7.</td><td>Стадії і етапи розробки</td><td>Дата</td><td>Назва етапу</td></tr>
<tr><td></td><td></td><td>01.11.2024 – 02.12.2024</td><td>Аналіз наукових досліджень та наявних методів оптимізації транспортних маршрутів</td></tr>
<tr><td></td><td></td><td>25.01.2025 – 02.02.2025</td><td>Формування вимог до технічних характеристик.</td></tr>
<tr><td></td><td></td><td>20.01.2025 – 08.02.2025</td><td>Формування вимог до системи та вибір архітектури програмного рішення.</td></tr>
<tr><td></td><td></td><td>08.02.2025 – 12.02.2025</td><td>Розробка архітектури системи та реалізація алгоритму оптимізації маршрутів.</td></tr>
<tr><td></td><td></td><td>18.02.2025 – 26.02.2025</td><td>Розробка серверної частини для збору, обробки та аналізу даних.</td></tr>
<tr><td></td><td></td><td>10.03.2025 – 21.03.2025</td><td>Реалізація клієнтської частини для взаємодії користувачів із системою.</td></tr>
<tr><td></td><td></td><td>30.03.2025 – 08.04.2025</td><td>Розробка методики тестування програмної системи для оцінки її продуктивності та якості побудови маршрутів</td></tr>
<tr><td></td><td></td><td>30.04.2025 – 10.05.2025</td><td>Розробка пояснювальної записки.</td></tr>
<tr><td></td><td></td><td>10.05.2025 –</td><td>Представлення кваліфікаційної роботи керівнику та рецензенту.</td></tr>
</table>

		20.05.2025 10.05.2025	Оформлення пояснювальної записки та підготовка презентації.
8.	Порядок контролю і приймання програмного продукту (моделі)	1. Перевірку ходу розробки комп'ютерної моделі виконувати раз в 3 тижні. 2. Захист розробленої моделі провести на засіданні Атестаційної комісії. 3. Пояснювальну записку подати в електронному вигляді в 1 примірнику.	

Виконавець
Студент групи КУ-41
Савченко К. І.

Замовник
к. е. н.
Чуб О. І.

Додаток В

Програма і методика випробувань програмного виробу
«Система оптимізації транспортних маршрутів з використанням Big Data»

1. **Об’єкт випробувань**

1.1. Назва розробленого прототипу: «Система оптимізації транспортних маршрутів з використанням Big Data».

1.2. Галузь застосування: Інформаційні технології, логістика

1.3. Перераховані відомості запозичуються з відповідних розділів Технічного завдання.

2. **Мета випробувань**

Перевірка відповідності функціональні можливості системи заявленим функціональним можливостям в технічному завданні (Додаток Б до пояснювальної записки до кваліфікаційної роботи).

3. **Загальні положення**

3.1. Підстави для проведення випробувань

Підставою для проведення випробувань є наказ про призначення атестаційної комісії.

3.2. Місце і тривалість випробувань

Приймальні (приймально-здавальні) випробування проводяться на базі комп’ютерного класу кафедри в період роботи атестаційної комісії.

3.3. Обсяг випробувань

Приймальні випробування програмного виробу проводяться в обсязі відповідному цієї програми і методики випробувань.

3.4. Організації, які беруть участь у випробуваннях

Приймальні випробування проводяться атестаційною комісією напередодні засідання (або в процесі засідання) за участю Замовника, Виконавці та інших осіб, присутніх на засіданні.

4. **Вимоги до програми або програмного виробу**

4.1 Функціональні вимоги:

1. Користувачі можуть вказувати точки відправлення й призначення через інтерактивну мапу.

2. Система виконує просторове зіставлення введених координат із вузлами графа (пошук найближчого сусіда) і повертає їхні ідентифікатори.

3. Має підтримуватися генерація трьох варіантів маршруту: найкоротший за відстанню, найшвидший з урахуванням поточного трафіку та компромісний який враховує і відстань і трафік.

4. REST-API надає кінцеві точки:

4.1 GET /snap – відображення координат на вузол графа.

4.2 GET /routes – побудова трьох варіантів маршруту.

4.3 GET /geocode – зворотне геокодування через Nominatim.

4.4 POST /traffic-update – оновлення ваг ребер відповідно до даних TomTom Traffic на вимогу.

5. Кожна кінцева точка має перевіряти коректність параметрів, повертати JSON з полями: ідентифікатор вузла, координати, перелік маршрутів із метриками (довжина, час, індекс заторів).
6. Клієнтська частина (React + Leaflet) відображає шари карти, наносить маршрути різними стилями, показує порівняльну панель метрик і надає кнопки для оновлення трафіку та перемикання варіантів маршрутів.
7. Утиліта на Go перетворює PBF-файл OSM в nodes.csv та edges.csv, які імпортуються в Neo4j через APOC із збереженням властивостей вузлів (lat, lng) і ребер (довжина, початковий ваговий коефіцієнт).

4.2 Нефункціональні вимоги

1. Час відгуку інтерфейсу: відображення, масштабування та вибір точки повинні оброблятися ≤ 100 мс.
2. Продуктивність API:
2.1 (/snap) та (/geocode) – відповіді ≤ 500 мс.
2.2 (/routes) – відповіді ≤ 1000 мс.
4. Модульність та зручність обслуговування: чіткий поділ коду на утиліти Golang, бекенд та фронтенд.

4.3. Вимоги до інтеграції:

1. Інтеграція з OpenStreetMap

Утиліта Golang повинна імпортувати файл OSM PBF (Київська область) та конвертувати його у (nodes.csv) та (edges.csv) у форматі, сумісному з імпортом Neo4j APOC.

2. Інтеграція з TomTom Traffic API

Модуль оновлення дорожнього руху повинен запитувати TomTom Traffic API, отримувати середню швидкість та рівень трафіку для кожного сегмента дороги та передавати їх до серверної частини як ваги для ребер графіка.

3. Інтеграція з Neo4j

Бекенд Node.js повинен підключатися до Neo4j (протокол Bolt) для імпорту CSV-файлів, індексації вузлів (Point) та виконання запитів A* з динамічними вагами ребер.

4. Інтеграція фронтенду та бекенду

Клієнтська частина React.js + Leaflet взаємодіє з сервером через безпечні виклики RESTful API (/snap, /routes, /geocode, /traffic-update) для отримання вузлів, побудови маршрутів та оновлення трафіку в режимі реального часу.

5. Вимоги до документації програмного забезпечення

Документація до програмного забезпечення для продукту «Система оптимізації транспортних маршрутів з використанням Big Data» повинна містити:

1. Посібник користувача: описує веб-інтерфейс вибір точок, відображення трьох профілів маршрутів та порівняння метрик.

2. Посібник адміністратора: інструкції з підготовки та запуску утиліти PBF to CSV, імпорту в Neo4j, оновлення трафіку та налаштування змінних середовища.

3. Документація API: детальний опис кінцевих точок (/snap, /routes, /geocode, /traffic-update) зі зразками запитів та відповідей у форматі Swagger або Postman.

4. Архітектура системи: діаграми компонентів (утиліта Golang, Neo4j, Node.js, React/Leaflet) та графова модель з вузлами та ребрами.

5. Документація розробника: структура репозиторію, інструкції з налаштування локального середовища, розгортання.

6. Засоби і порядок випробувань

6.1 інструменти випробування

1. Модульне тестування:

Кожен окремий модуль системи тестується незалежно. Для утиліт Go (конвертер PBF to CSV та оновлення трафіку) використовується вбудований фреймворк go test з тестами, що охоплюють логіку парсингу PBF, обчислення ваг ребер та запису CSV-файлів. У компонентах React (шари карти, панель метрик) Jest використовується разом з бібліотекою React Testing Library: вони перевіряють правильність рендерингу, обробку кліків та відображення даних за допомогою змінних props.

2. Наскрізне тестування:

Повний сценарій взаємодії клієнт-сервер тестується автоматично за допомогою Cypress. Тестові випадки симулюють вибір початкової та кінцевої точки на карті, виклик побудови маршруту та відображення трьох поліліній з метриками. Окремий скрипт тестує процес оновлення трафіку: кнопка «Оновити» запускає запит до /traffic-update та змінює стиль ребер на карті.

3. Тестування API:

Для тестування функціональності REST-інтерфейсів використовується Postman, який генерує колекцію запитів до всіх кінцевих точок (/snap, /routes, /geocode, /traffic-update). Кожна кінцева точка тестується на обробку коректних даних, очікуючи статус відповіді 200 та наявність повного набору полів в отриманому JSON, а також на реакцію на некоректні параметри, коли система повинна повернути код 4xx з чітким повідомленням про помилку. Результати цих тестів експортуються у вигляді звіту, що містить інформацію про час відповіді та статус кожного запиту.

4. Ручне тестування:

За допомогою Chrome DevTools вимірюється час відгуку інтерфейсу (панорамування, масштабування, побудова маршруту). Окремо тестується відновлення системи після примусової зупинки сервісів (Neo4j або Node.js): після перезапуску перевіряється API та інтерфейс користувача, щоб переконатися, що вони працюють коректно та не було втрачено дані.

6.2. Процедура тестування

1. Попередня підготовка середовища:

На етапі підготовки було встановлено Neo4j та імпортовано в нього геодані OSM за допомогою утиліти Golang (PBF to CSV). Було запущено сервер (Node.js) та клієнт (React + Leaflet), встановлено всі необхідні змінні середовища (ключ TomTom Traffic API, облікові дані Neo4j) та перевірено правильність роботи компонентів.

2. Модульне тестування:

За допомогою go-test було протестовано утиліти конвертації PBF to CSV та оновлення трафіку – усі тести пройшли успішно, покриття коду становило понад 90%. Для компонентів React (шари карти, панель метрик) запуск Jest з бібліотекою React Testing Library підтвердив правильність рендерингу, обробки кліків та відображення даних.

3. Інтеграційне тестування:

Було протестовано взаємодію між утилітою Golang, Neo4j та сервісом Node.js. Запити до кінцевих точок (/snap, /routes, /geocode, /traffic-update) виконувалися за допомогою Postman, що гарантувало коректну передачу даних з бази даних клієнту та зміну ваг ребер у графі оновленнями трафіку.

4. Наскрізне тестування

Cypress було використано для автоматичної перевірки повного сценарію. Вибір початкової та кінцевої точки на карті, виклик /routes і відображення трьох маршрутів різними стилями, оновлення трафіку через кнопку (Оновити) та перевірка зміни відображення.

5. Ручне тестування:

За допомогою Chrome DevTools ми виміряли час відгуку інтерфейсу під час панорамування карти, масштабування та побудови маршрутів – значення не перевищували 100 мс. Також ми змоделювали завершення роботи сервісів Neo4j та Node.js, після чого перевірили відновлення системи без втрати даних.

6.3. Перевірка функціональності

Перевірка функціональності системи включала в себе тестування таких функцій:

1. Вибір початкової та кінцевої точки:

Було зроблено клік на карті, щоб встановити координати початку та кінця маршруту. Було перевірено, що система коректно відображає маркери та передає їх на серверну частину.

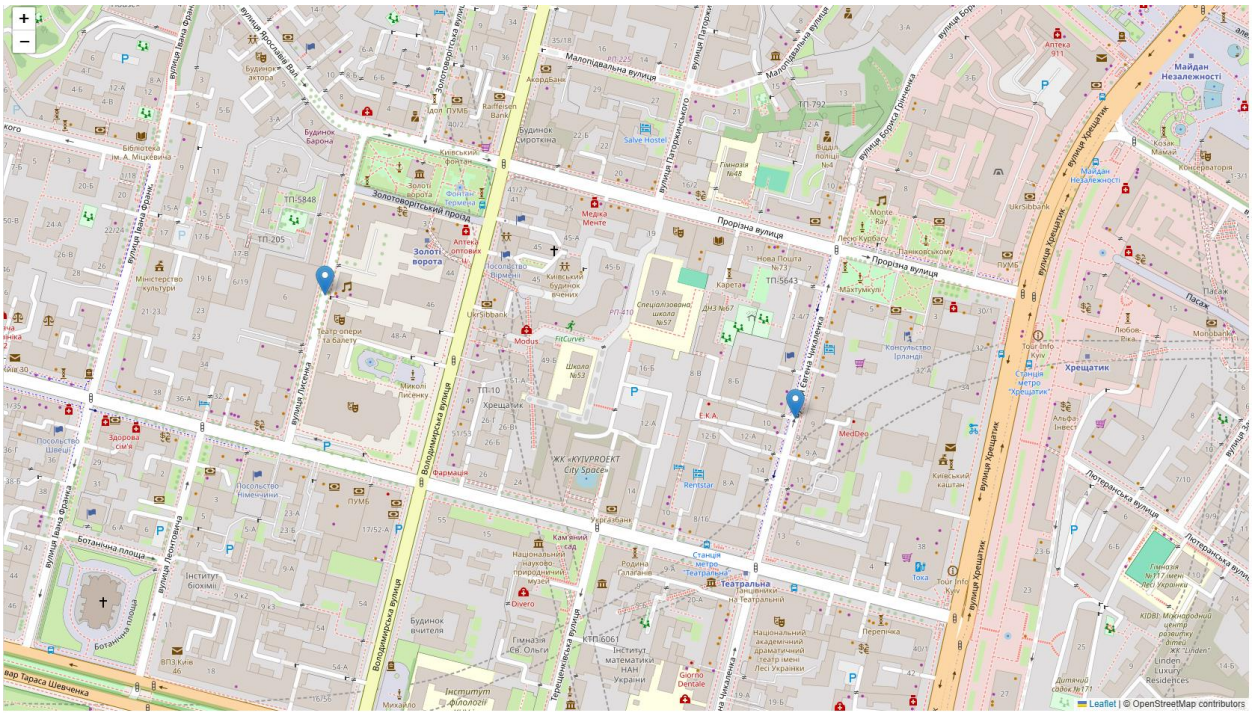

Рис. В.1 – Тестування відображення маркерів.

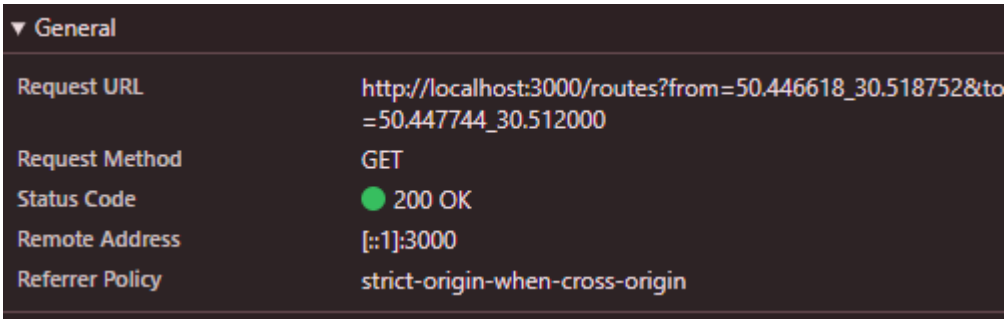

Рис. В.2 – Передача точок на серверну частину.

2. Кінцева точка (/snap):

Запити були зроблені з реальними координатами та координатами поза мережею вузлів OSM. У першому випадку було отримано статус 200 та JSON з полями nodeId, lat, lng, distance; у другому – статус 404 з повідомленням «Точку не знайдено».

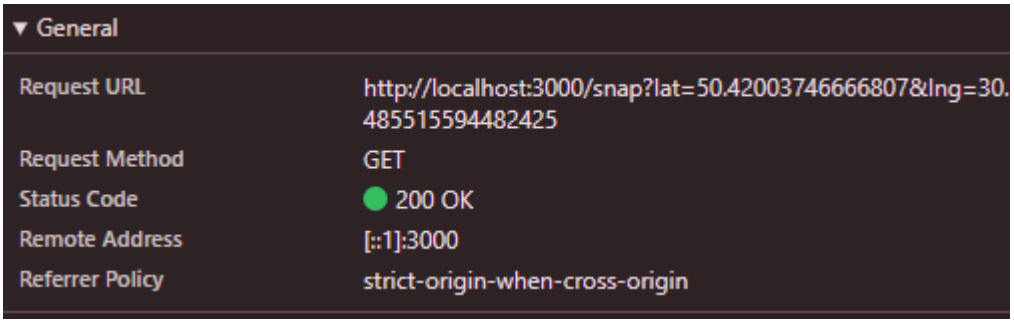

Рис. В.3 – Тестування кінцевої точки з реальними координатами.

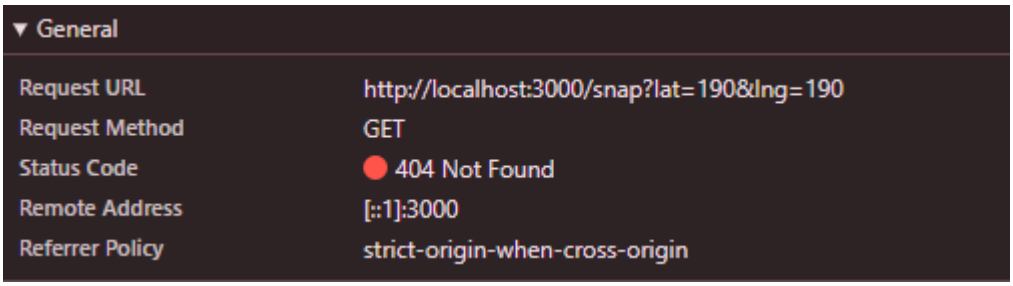

Рис. В.4 – Тестування кінцевої точки з координатами які лежать за межами вузлів.

3. Кінцева точка (/geocode):

Було протестовано зворотне геокодування кількох координат: запит повертав статус 200 та повну адресу в полі адреси.

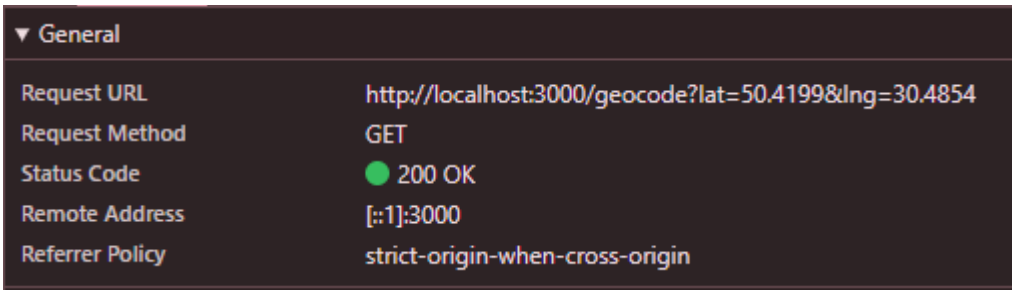

Рис. В.5 – Тестування кінцевої точки з реальними координатами.

Рис. В.6 – Відповідь кінцевої точки.

Якщо координати були по за межами Київської області, повертався статус 400 з повідомленням про помилку в параметрах.

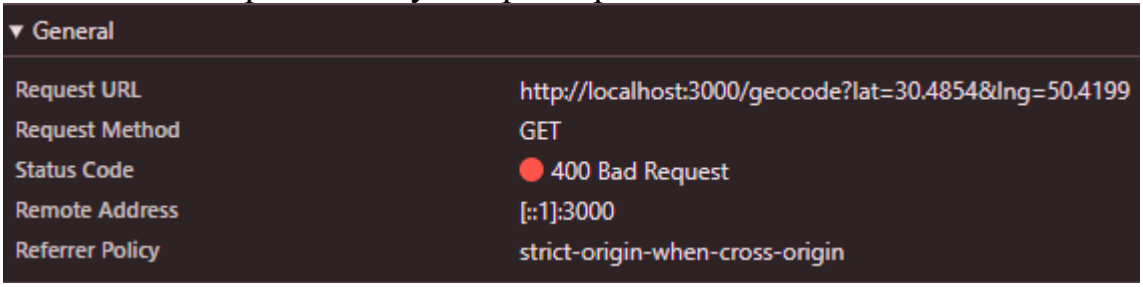

Рис. В.7 – Тестування кінцевої точки з координатами по за межами Київської області.

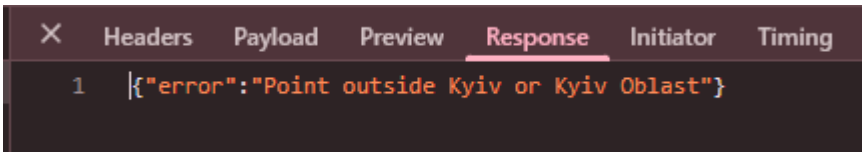

Рис. В.8 – Відповідь кінцевої точки.

4. Кінцева точка (/routes):

Було виконано запит з параметрами (from=<id1>&to=<id2>). Було отримано три масиви координат для трьох варіантів маршрутів: довжина, трафік та комбінований, кожен з повними метриками (відстань, тривалість, індекс заторів).

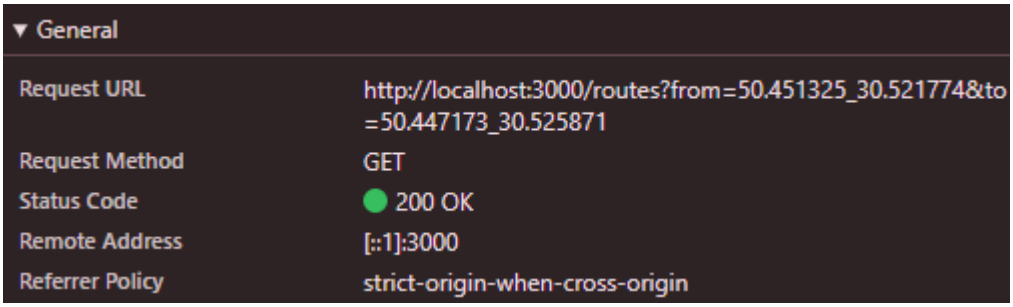

Рис. В.9 – Тестування кінцевої точки з реальними координатами.

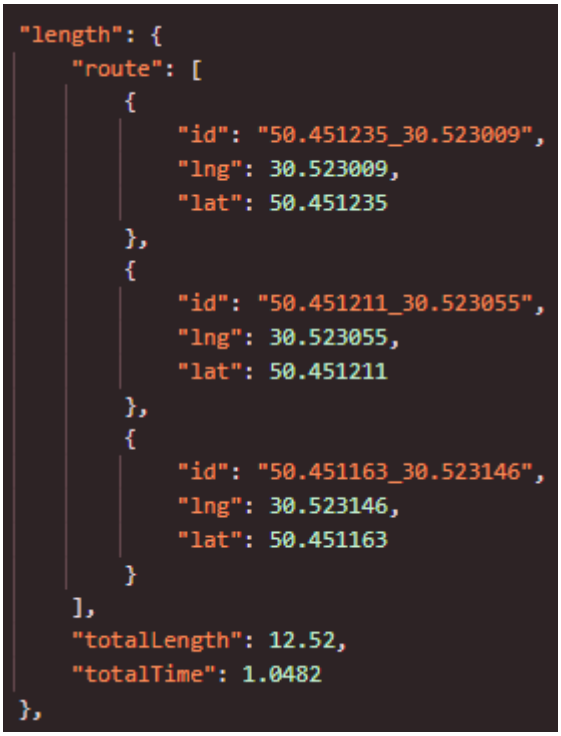

Рис. В.10 – Відповідь кінцевої точки.

Візуально маршрути відображаються на карті трьома різними кольорами.

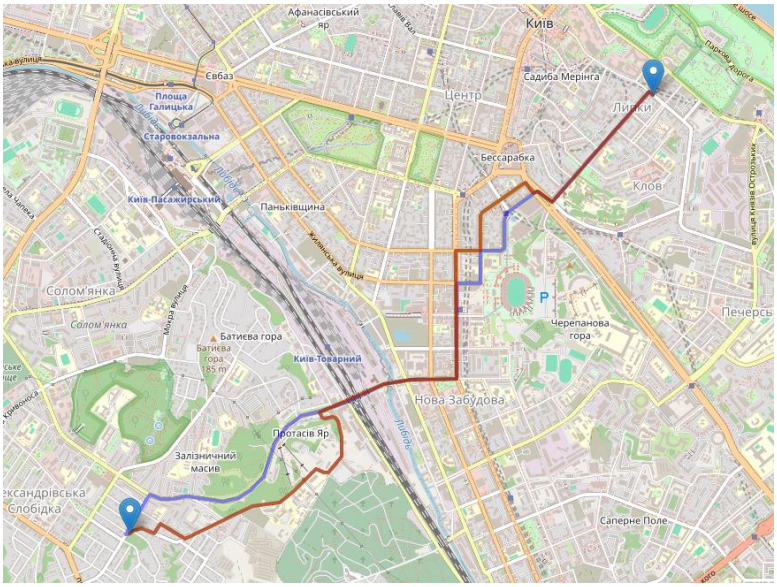

Рис. В.11 – Відображення маршрутів на мапі.

5. Кінцева точка (/traffic-update):

Після натискання кнопки «Оновити дані про трафік» система викликала утиліту Golang, оновила ваги ребер у Neo4j та повернула статус 200 з кількістю оновлених сегментів.

Рис. В.12 – Тестування кінцевої точки.

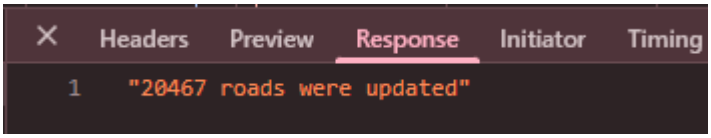

Рис. В.13 – Відповідь кінцевої точки.

6. Клієнтський інтерфейс:

Для кожного сценарію була реалізована відповідна логіка, під час вибору відображення відповідного маршруту (натискання відповідної кнопки) на карті активується лише відповідна полілінія.

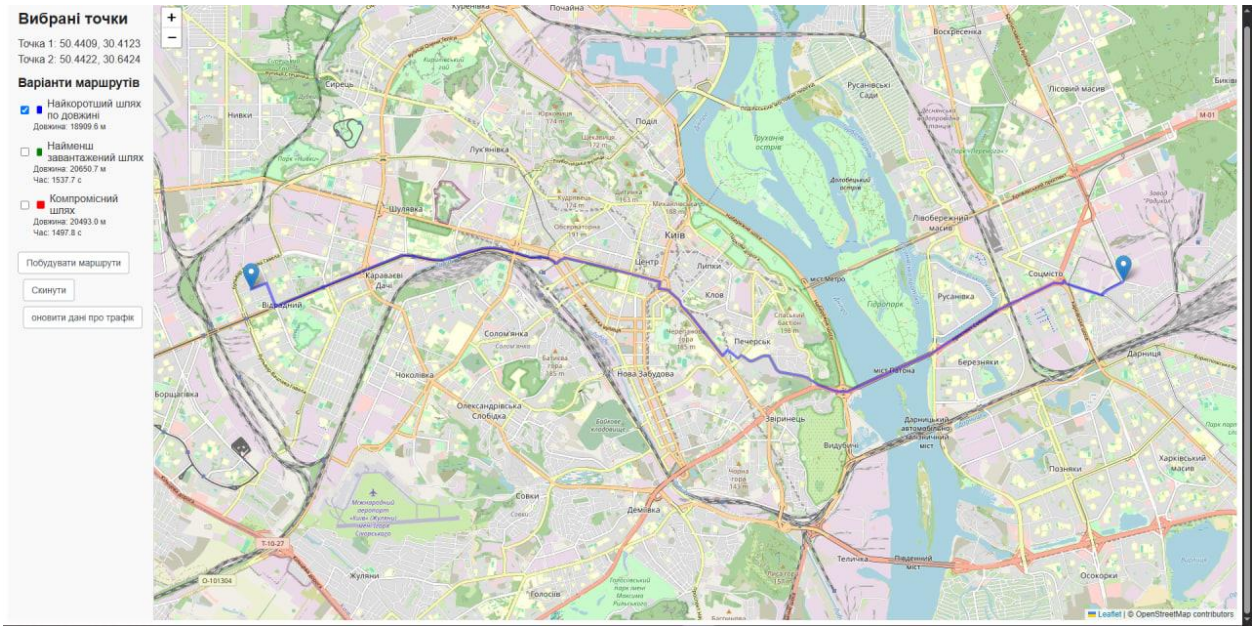

Рис. В.14 – Відображення найкоротшого по довжині маршруту на мапі.

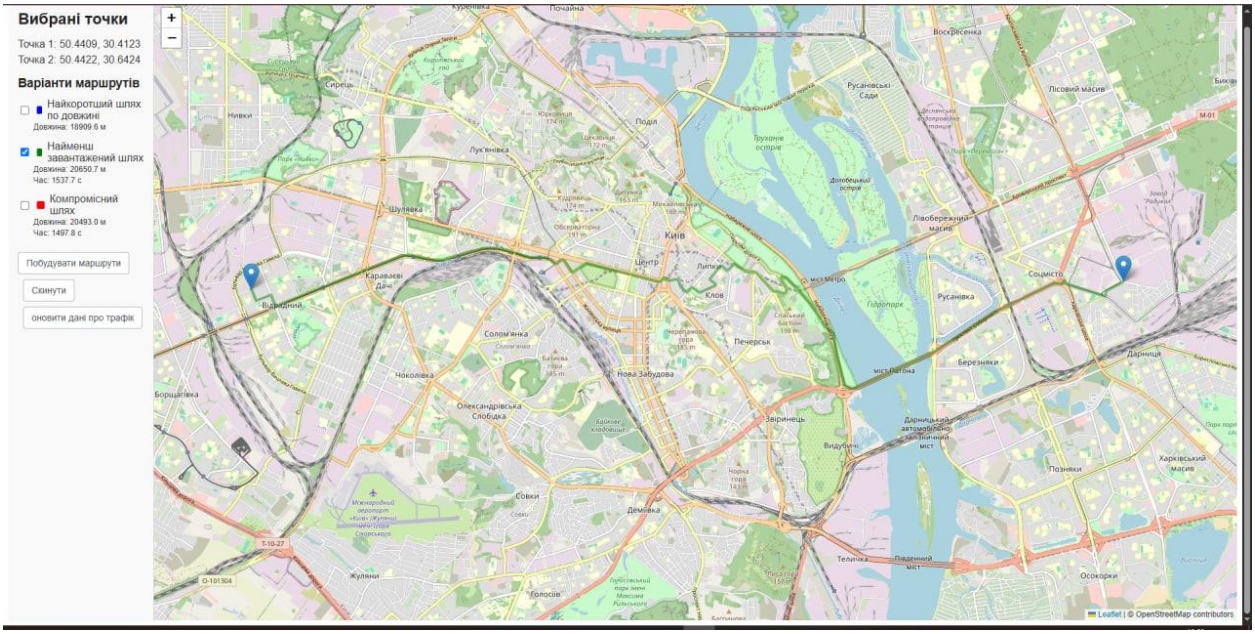

Рис. В.15 – Відображення найменш завантаженого маршруту на мапі.

Рис. В.16 – Відображення компромісного маршруту на мапі.

6.4. Висновок

Тестування та налагодження підтвердили стабільну та надійну роботу системи. Було перевірено коректність усіх ключових можливостей, включаючи генерацію маршрутів, взаємодію з користувачем та адміністративні операції. Для подальшого розвитку системи основна увага має буде приділена масштабованості, підвищенню безпеки та покращенню взаємодії з користувачем.

Виконавець: студент групи КУ-41, Савченко К.І. _____