Міністерство освіти і науки
України Національний
технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(навчально-науковий інститут)
факультет інформаційних технологій
(факультет)
Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

# ПОЯСНЮВАЛЬНА ЗАПИСКА
## кваліфікаційної роботи ступеня магістр
(бакалавра, магістра)

**Здобувача вищої освіти** Панасенка Івана Олеговича
(ПІБ)

**академічної групи** 126м-23-1
(шифр)

**спеціальності** 126 «Інформаційні системи та технології»
(код і назва спеціальності)

**спеціалізації за освітньо-професійною (освітньо-науковою) програмою**
(за наявності)
«Інформаційні системи та технології»
(офіційна назва)

**на тему** Дослідження алгоритмів знаходження оптимальних маршрутів у складі Web-додатку на базі мікросервісної архітектури
(назва за наказом ректора)

| Керівники | Прізвище, ініціали | Оцінка за шкалою | | Підпис |
|---|---|---|---|---|
| | | рейтинговою | інституційною | |
| кваліфікаційної роботи | Гаркуша І. М. | 90 | відмінно | |
| розділів: | | | | |
| Розділ 1 | | | | |
| Розділ 2 | | | | |
| Розділ 3 | | | | |
| | | | | |

| | | | | |
|---|---|---|---|---|
| **Рецензент** | Ширін А.Л. | 90 | відмінно | |
| **Нормоконтролер** | Коротенко Г. М. | 90 | відм. | |

**Дніпро**
**2024**

**ЗАТВЕРДЖЕНО:**
завідувач кафедри
інформаційних технологій та комп'ютерної інженерії
(повна назва)
_____ В.В. Гнатушенко
(підпис) (ініціали та прізвище)
«17» 10 2024 року

# ЗАВДАННЯ

## на кваліфікаційну роботу

**ступеня** магістр
(бакалавра, магістра)

**здобувача вищої освіти** Панасенка І. О **академічної групи** 126м-23-1
(прізвище та ініціали) (шифр)

**спеціальності** 126 «Інформаційні системи та технології»

**спеціалізації за освітньою-професійною програмою** «Інформаційні системи та технології»
(за наявності)

**на тему** Дослідження алгоритмів знаходження оптимальних маршрутів у складі Web-додатку на базі мікросервісної архітектури

затверджену наказом ректора НТУ «Дніпровська політехніка» від 17.10.2024 № 1388-С

| Розділ | Зміст | Термін виконання |
|---|---|---|
| Розділ 1 | Аналіз існуючих алгоритмів знаходження найкоротшого маршруту | 15.09.2024 – 01.12.2024 |
| Розділ 2 | Огляд архітектури програмного забезпечення | 20.09.2024 – 05.11.2024 |
| Розділ 3 | Аналіз результатів знаходження оптимального маршруту | 05.11.2024 – 10.11.2024 |
| | | |

**Завдання видано** _____ І. М. Гаркуша
(підпис керівника) (ініціали та прізвище)

**Дата видачі** 30.08.2024

**Дата подання до екзаменаційної комісії** 20.12.2024

**Прийнято до виконання** _____ І. О. Панасенко
(підпис здобувача вищої освіти) (ініціали та прізвище)

# РЕФЕРАТ

**Пояснювальна записка:** 82 стор., 32 рис., 3 додатки., 33 джерел.

**Об’єкт дослідження:** алгоритми маршрутизації.

**Предмет дослідження:** мікросервісна архітектура Web-додатку для підвищення ефективності використання алгоритмів знаходження оптимальних маршрутів.

**Мета магістерської роботи:** дослідження алгоритмів знаходження оптимальних маршрутів з подальшою вдосконаленою реалізацією у складі Web-додатку на базі мікросервісної архітектури.

У вступі подано опис проблеми, сформульовано мету та підхід до виконання кваліфікаційної роботи, сформовано задачі, які потрібно виконати.

У першому розділі розглянуто алгоритми маршрутизації, популярні системи навігації, проведено їх порівняння, визначено змістовну та математичну постановку задачі та обґрунтовано обрані алгоритми.

У другому розділі були розглянуті архітектуру системи, визначено інструменти для її розробки та проведено тестування програмного забезпечення. Було описано специфікації функцій застосунків та роботу системи.

У третьому розділі було розглянуто вхідні та вихідні дані системи, створено ряд сценаріїв для перевірки методів маршрутизації та проведено аналіз ефективності алгоритмів на основі отриманих результатів.

МАРШРУТИЗАЦІЯ, ЕВРИСТИЧНІ АЛГОРИТМИ, АЛГОРИТМ A*, АЛГОРИТМ ДЕЙКСТРИ, JAVASCRIPT, NODE.JS, POSTGRESQL.

ABSTRACT

**Explanatory note:** 82 pages, 32 figures, 3 applications, 33 sources.

**Object of research:** routing algorithms.

**Subject of research:** microservice architecture of Web-application for improvement of efficiency of routing algorithms for optimal routes.

**Purpose of Master's thesis:** research of routing algorithms for optimal routes with further improved implementation as part of Web-application with microservice architecture.

The introduction outlines the problem statement, formulates goal and approach to completing the qualification work, defines the tasks required for its completion.

The first section covers routing algorithms, popular routing methods, their comparison, determines the problem and its mathematical description, justifies the chosen algorithms.

The second section describes system architecture, instruments for its development and software tests. The description of the system functions were provided along with the system documentation.

The third section researches input and output data, provides description of scenarios for the testing of routing algorithms efficiency and analyzes the results of the tests to compare the efficiency of the algorithms.

ROUTING, HEURISTIC ALGORITHMS, A* ALGORITHM, DIJKSTRA ALGORITHM, JAVASCRIPT, NODE.JS, POSTGRESQL.

**ЗМІСТ**

# ВСТУП

Питання побудови маршрутів є актуальним в повсякденні та в професійному середовищі, оскільки ефективне використання наявної інформації про території дозволяє визначити оптимальний шлях, надаючи найкоротший маршрут та дозволяючи зекономити час на подорожі. Було досліджено значну кількість алгоритмів на їх ефективність у вирішенні цього питання для забезпечення коректної відповіді на посталий запит.

З розвитком технологій, доступ до засобів маршрутизації спростився, і потреба у дорогому обладнанні тепер є необов'язковим, оскільки достатньо пристрою з підтримкою GPS-сигналу та відповідного програмного забезпечення. Але і в цьому випадку доцільним лишається використання ефективних рішень для знаходження оптимального маршруту, що провокує використання кожною програмою різних алгоритмів у відповідних сценаріях.

Ключовим є використання ефективних алгоритмів для знаходження оптимальних маршрутів. Найпоширенішим рішення поставленої задачі є використання алгоритму Дейкстри, разом з іншими алгоритмами, які при комбінованому використанні у різних додатках для навігації по місцевості дозволяють знайти маршрут до місця призначення.

Однією з проблем сучасних навігаційних систем є неможливість ефективного прокладання маршруту через кілька заданих точок, що часто призводить до необхідності багаторазового використання програм та створення неоптимальних маршрутів. Такі обмеження не враховують індивідуальні побажання користувачів, через що загальна дистанція подорожі може суттєво збільшуватися порівняно з оптимальним маршрутом.

Іншою важливою складовою є наявність бази даних із інформацією про маршрути, пам'ятки, ресторани та готелі, яка дозволяє одночасно вирішувати завдання маршрутизації та інформування користувачів. Забезпечення доступу до такої інформації дає змогу розробляти маршрути, які відповідатимуть потребам користувачів і включатимуть цікаві місця. Відсутність подібного

функціоналу створює незручності, оскільки користувач не може одразу отримати повну картину подорожі з урахуванням своїх інтересів.

Мета дослідження: дослідження алгоритмів знаходження оптимальних маршрутів з подальшою вдосконаленою реалізацією у складі Web-додатку на базі мікросервісної архітектури.

Об'єкт дослідження: алгоритми маршрутизації.

Предмет дослідження: мікросервісна архітектура Web-додатку для підвищення ефективності використання алгоритмів знаходження оптимальних маршрутів.

Завдання дослідження:

- створення вершин графів на карті для виконання пошуку маршрутів;
- імплементація алгоритмів для прорахунку найкоротшого шляху між точками;
- створення бази даних для зберігання інформації про вершини графів та дороги;
- розробка функціоналу для прокладання маршрутів через усі точки, обрані користувачем;
- підтримка обрання типу транспортного засобу для врахування обмежень при переміщенні;
- реалізація виведення інформації про маршрут для користувачів;
- імплементація можливості збереження графів для їх подальшого використання;
- порівняння ефективності реалізованих алгоритмів на основі результатів виконання на прикладах.

Дослідження спрямоване на подолання існуючих обмежень у системах маршрутизації шляхом удосконалення алгоритмів та інтеграції інформаційних баз, що забезпечить ефективне планування маршрутів з урахуванням потреб користувачів.

# 1. АНАЛІЗ ІСНУЮЧИХ АЛГОРИТМІВ ЗНАХОДЖЕННЯ НАЙКОРОТШОГО МАРШРУТУ

Для даного дослідження використовується задача, опис якої підпадає під опис модифікованої задачі комівояжера [1]. Завдання вивчається принаймні з 19 століття. За нею, кур'єр відвідує всі міста у своєму списку і повертається до початкового міста, витрачаючи мінімум часу. Головна відмінність, що є елемент питання про найкоротший шлях, оскільки наприкінці є можливість повернутися до початкового міста.

Ще одним важливим фактором, який слід враховувати, є порівняння найкоротших та найкращих маршрутів. Хоча короткий маршрут включає лише шлях від початкової точки до кінцевої і може скоротити відстань поїздки, оптимальний маршрут передбачає врахування ряду інших факторів, таких як огородження, ремонтні роботи, висоту підйому під час руху або наявність цікавих об'єктів як АЗС чи магазини, що може забезпечити більш ефективну подорож [2].

## 1.1 Зміст та математична постановка виконання маршрутизації

Графом є структура, яка складається з вершин - точок, між якими визначаються шляхи, - та ребер - з'єднань між вершинами, які мають вагу.

Вага ребер є числовим значенням, яке характеризує "вартість" переходу між двома вершинами. Вага може бути виражена у дистанції, часу на проходження, витрату ресурсів і не тільки.

В задачі визначаються початкова вершина, від якої починається обчислення найкоротших шляхів, та цільова вершина, до якої потрібно знайти найкоротший шлях.

Неорієнтованим називають граф в якому ребра між вершинами графу дозволяють пересування в обидва боки. Відповідно, в орієнтованому графі ребра мають односторонній напрямок, і зворотній рух по ним неможливий.

Коли ребра графу мають вагу назначену їм, він є зваженим. Від’ємна вага ребра означає, що перехід між двома вершинами дає "вигоду" або знижує

вартість (тобто, перехід між відповідними вершинами графу знижує витрати на подорож), що може зустрічатися у фінансових задачах і не тільки.

Приклад орієнтованого зваженого графу наведено на рисунку 1.1.

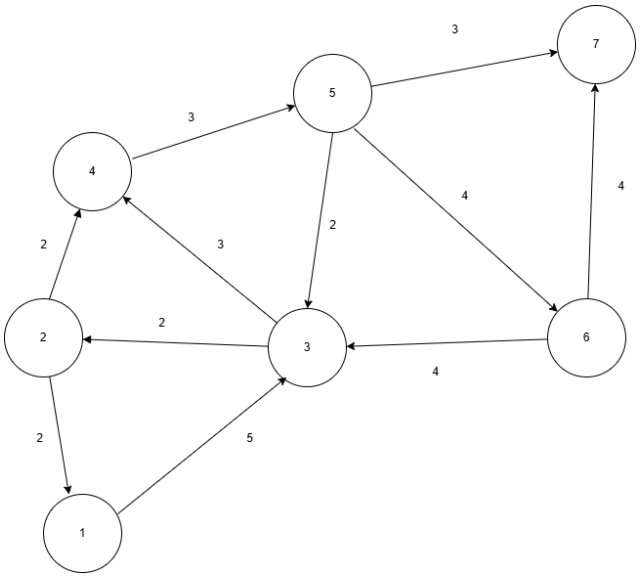


Рисунок 1.1 – Орієнтований зважений граф

В системі дано граф G = (V, U), де V – це множина вершин, а U – множина ребер графа. Початкова вершина позначається як v_start, кінцева – v_end. Окрім того, визначена підмножина V_intermediate, що складається з проміжних вершин, які необхідно включити до маршруту.

Завдання полягає у побудові шляху, що починається з вершини. v_start, завершується v_end, і проходить через усі точки множини V_intermediate.

Метою є визначення послідовності вершин P = {v_start, v_1, v_2, ..., v_n, v_end}, де кожна вершина v_i належить V_intermediate для i = 1, 2, ..., n, а кожна пара суміжних вершин (v_i, v_{i+1}) утворює ребро у графі G.

Граф G(V, U) представляє з себе модель місцевості. Потрібно знайти оптимальний маршрут, що з’єднує вершини 1 і 2, обов’язково проходячи через вершини 5 та 6.

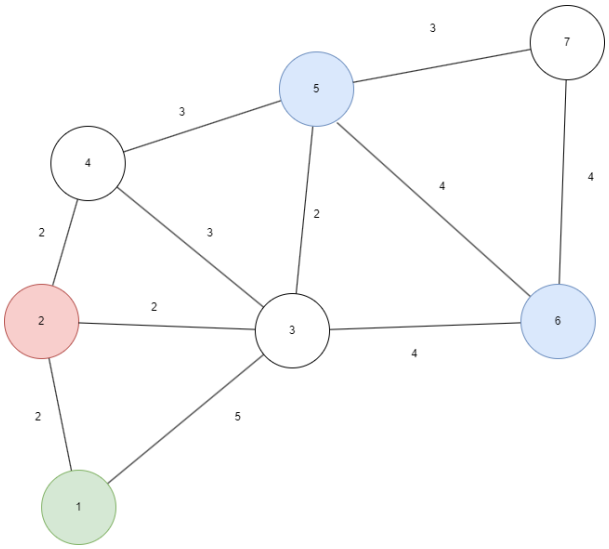


Рисунок 1.2 – Система вершин графів з позначеннями початкової, проміжних та кінцевих вершин графів

Умови можна корегувати до потреб: змінювати початкову точку, кількість проміжних вершин, або видалити кінцеву вершину. Ребра графа поєднують лише одну пару вершин і мають вагу, яка відображає відстань між з’єднаними вершинами. Існує можливість обрання транспортного засобу для врахування обмежень при маршрутизації, роблячи граф орієнтовним.

Поставлене завдання подібне до пошуку найкоротшого шляху, однак із необхідністю відвідання кількох проміжних вершин. Для її розв’язання задача розбивається на підзадачі, кількість яких дорівнює числу вершин, що потрібно пройти. У цьому прикладі таких вершин три.

На першому етапі алгоритм використовується для визначення найкоротших маршрутів від стартової точки до проміжних вершин. Це дозволяє вибрати оптимальну першу точку для прокладання маршруту.

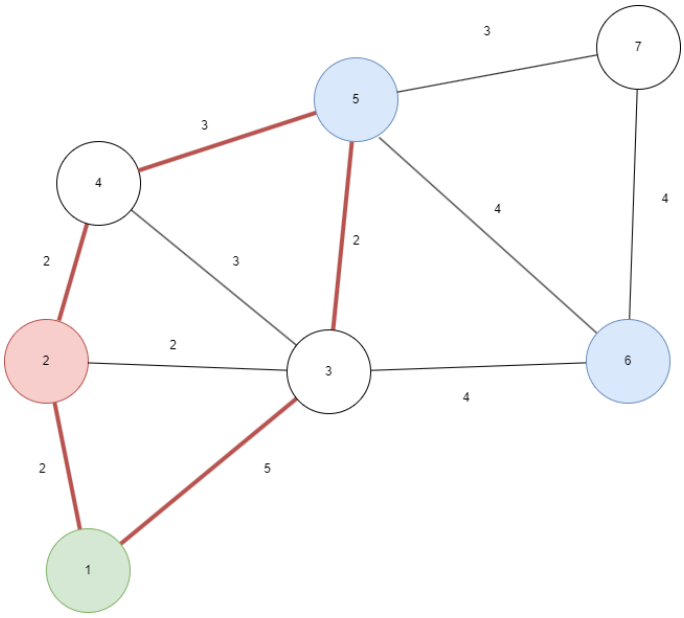


Рисунок 1.3 – Визначені шляхи між стартовою точкою 1 і вершиною графу 5

По проходженню першої вершини яку треба пройти, задача переходить до наступного етапу. Знову визначаються найкоротші маршрути від поточної точки до інших визначених вершин. Після вибору наступної найближчої вершини маршрут між ними прокладається аналогічно до попереднього етапу.

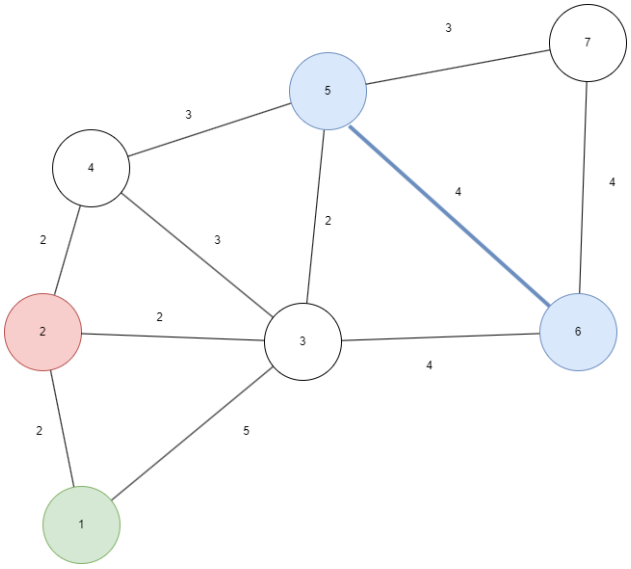


Рисунок 1.4 – Визначений маршрут між точкою 5 і вершиною графу 6

По проходженню всіх проміжних вершин, якщо завдання передбачає кінцеву точку, прокладається заключна частина маршруту від останньої проміжної точки до фінальної. У результаті отримуємо оптимальний маршрут, що проходить через усі задані вершини графа.

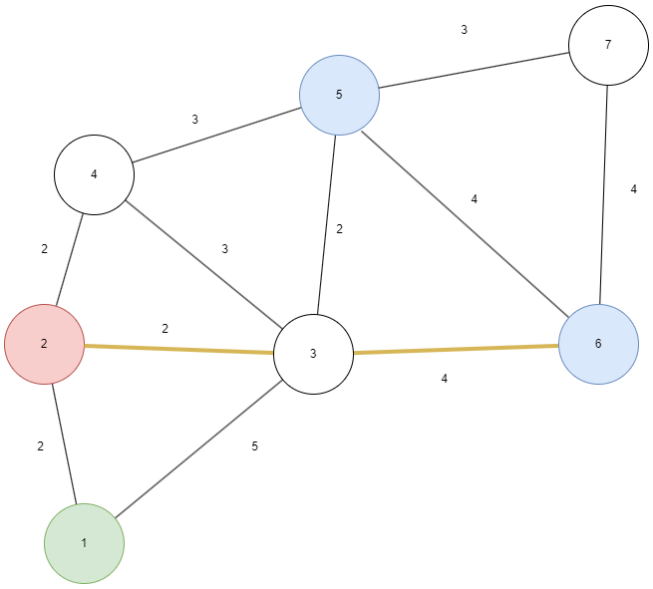


Рисунок 1.5 – Визначений маршрут між точкою 6 і вершиною графу 2

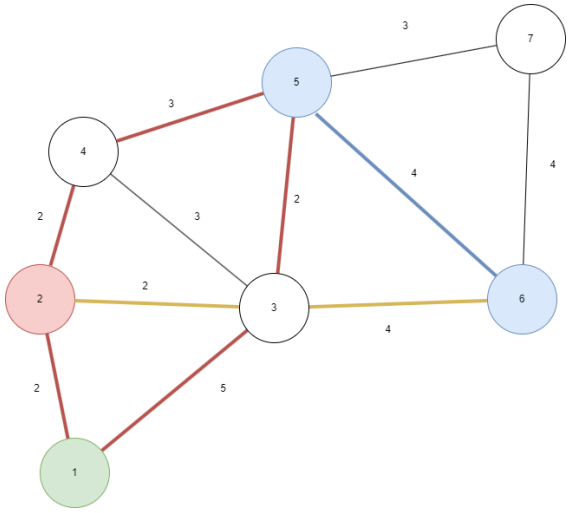


Рисунок 1.6 – Фінальний оптимальний маршрут від початкової вершини графу 1 через точки 5, 6 та до вершини графу 2

У наданому прикладі загальна довжина маршруту становить 17. Маршрут може бути побудований у такій послідовності: 1–2–4–5–6–3–2 або 1–3–5–6–3–2.

### 1.2 Опис існуючих алгоритмів розв’язання.

З огляду на вік і популярність задачі про найкоротший маршрут, за минулі роки було вивчено велику кількість алгоритмів для її вирішення. Найбільш поширені алгоритми розглядають різні варіанти переміщення по вершинах графа, і їх підхід дає різні результати.

Опис кожного алгоритму містить в собі сценарії найгіршої швидкодії та опис використання пам’яті. Найгірша швидкодія – теоретична оцінка максимальної кількості операцій, які алгоритм виконує для розв'язання задачі у найскладнішому можливому випадку. Вона використовується для визначення верхньої межі часу виконання алгоритму, забезпечуючи гарантії його поведінки за найбільш несприятливих обставин.

Використання пам’яті – це оцінка обсягу пам'яті, яку потребує алгоритм для обробки вхідних даних і проміжних результатів. Так само, як швидкодію, використання пам'яті часто оцінюють у найгіршому випадку, щоб дізнатися, скільки ресурсів потрібно в найбільш ресурсомісткому сценарії.

#### 1.2.1 Алгоритм Дейкстри

Алгоритм Дейкстри використовується для знаходження найкоротших шляхів від заданої вершини графа до всіх інших вершин. Він ефективний для графів із невід’ємними вагами ребер, але не може обробляти графи з від’ємними вагами [3], оскільки алгоритм базується на припущенні, що при опрацюванні вершини її поточна оцінка $g(v)$ є остаточною. Проте у графах із від’ємними вагами це припущення стає хибним. Якщо в графі є цикл із від’ємною сумарною вагою, поточна оцінка $g(v)$ може бути знижена необмежено багато разів при повторному опрацюванні вершини. Це може призвести до того, що алгоритм не завершить роботу, або надасть неправильні результати

Результатом виконання алгоритму є набір найкоротших шляхів, які можуть бути використані для досягнення бажаних цілей у задачах маршрутизації.

Основні принципи роботи алгоритму:

Ініціалізація:

Нехай V – множина вершин графа, E – множина ребер графа, A – множина опрацьованих вершин (із уже знайденими найкоротшими шляхами, X – множина вершин, для яких шлях ще не визначено (черга на опрацювання).

- Для кожної вершини $v$ встановлюється:
  - $g(v)$: поточна оцінка найкоротшого шляху від початкової вершини s до v. Початкова вершина s має $g(s) = 0$, а всі інші $g(\text{v}) = \infty$.
- Встановлюється $A = \emptyset$ та $X = \{s\}$.

Основний цикл:

Поки $X \neq \emptyset$:

1. Знаходиться вершина v із X, яка має найменше значення $g(v)$, тобто: $g(v)$: = min $_{u \in X}$ $g(u)$,

Видаляється v із X та додається до A.

2. Для кожної суміжної вершини u, яка задовольняє умову $(v, u) \in E$:

- Якщо $u \in$ A, дії не виконуються.
- Якщо $u \in$ X, перевіряється умова: $(v) + w(v, u) < g(u)$. Якщо умова виконується, оновлюється значення $g(u)$ та предок вершини u встановлюється як v.
- Якщо $u \notin$ A $\cup$ X, додається u до X, і для неї встановлюються:

$$g(u) = g(v) + w(v, u)$$

$$parent(u) = v$$

Завершення:

Алгоритм завершується, коли множина X стає порожньою, або (за бажанням) як тільки цільова вершина t додається до A. У цьому випадку найкоротший шлях до t уже знайдено, що може зменшити час виконання.

Недоліки:

- неможливість обробки графів із від’ємними вагами;
- висока обчислювальна складність для графів із великою кількістю вершин і ребер.

Переваги:

- гарантоване знаходження найкоротших шляхів для всіх вершин у графах із невід’ємними вагами;
- можливість зупинити виконання після досягнення цільової вершини, якщо необхідно знайти лише один шлях.

У випадку реалізації алгоритму з використанням черги пріоритетів, найгірша швидкодія досягає O((V + E) log V), коли використання пам’яті – O(V).

Алгоритм Дейкстри широко використовується у задачах маршрутизації, зокрема у транспортних системах та навігаційних додатках, де потрібно ефективно знаходити шляхи між заданими точками [4].

### 1.2.2 Алгоритм Беллмана-Форда

Алгоритм Беллмана-Форда, розроблений в 1956 році, відрізняється від алгоритму Дейкстри тим, що він може обробляти вершини графу, вага яких може бути негативною. Однак існують обмеження, які не дозволяють йому працювати належним чином при негативних циклах. Однак алгоритм не працює коректно, якщо в графі є негативні цикли — такі цикли, у яких сума ваг усіх ребер дорівнює від’ємному значенню. У цьому випадку алгоритм застрягає в нескінченному процесі оновлення відстаней, оскільки значення шляхів можуть зменшуватися без кінця.

Для виявлення негативних циклів алгоритм перевіряє, чи можна після n-1 ітерацій зменшити вагу будь-якого шляху. Якщо така можливість існує, це означає наявність негативного циклу, і алгоритм припиняє виконання без знаходження найкоротших шляхів.

Реалізація алгоритму подібна до реалізації алгоритму в Дейкстри, з тією різницею, що не використовуються черга пріоритетів. Натомість алгоритм проходить по всіх ребрах графіка кілька разів і, як у попередньому алгоритмі, оновлює відстані. Якщо n – це кількість вузлів у графі, то алгоритм Беллмана-Форда без циклу не зможе включити в шлях у графі більше ребер, ніж n-1.

Найгірша швидкодія алгоритму досягає O(|V||E|), коли використання пам’яті – O(V).

Через підвищену складність, рекомендується використовувати алгоритм Беллмана-Форда тільки при наявності вершин графа з негативними вагами [5].

### 1.2.3 Алгоритм A*

Алгоритм A* (читається як "A зірочка") належить до класу евристичних алгоритмів пошуку. Він застосовується для знаходження найкоротшого шляху між початковою та кінцевою точками на графі, де вершини мають додатні ваги. Завдяки евристичній функції алгоритм спрямовує пошук у потрібному напрямку, що дозволяє зменшити час виконання. У більшості випадків A* забезпечує знаходження оптимального рішення, роблячи його повним алгоритмом.

Функція оцінки $f(x)$ визначає пріоритет вершини x для обробки, і визначається формулою:

$$f(x) = g(x) + h(x),$$

де:

- g(x) – вартість шляху, яка обчислюється як сума ваг ребер, що утворюють шлях від початкової вершини s до поточної x, яку можна описати наступною формулою:

$$g(x) = \sum_{(u,v)\in P(s,x)} w(u,v)\,,$$

де P(s, x) - послідовність ребер (шлях) від початкової вершини s до поточної вершини x, і w(u, v) - вага ребра між вершинами u та v.

- h(x) – евристична функція, прогноз залишкової вартості, який повинен ефективно підібраним для конкретної задачі та бути допустимим:

$$\forall x, h(x) \leq h^*(x),$$

Де $h^*(x)$ – справжня ціна досягнення вершини з x.

В контексті маршрутизації по місцевості, може бути використана евклідова або мангеттенська відстань.

Алгоритм A* використовується у задачах, які потребують динамічного планування, наприклад, у створенні маршрутів у реальному часі, і популярний у вирішенні логічних задач, таких як задача восьми ферзів [6]. Найгірша швидкодія алгоритму досягає O(|E| log |V|) = O($b^d$), коли використання пам'яті – O(|V|) = O($b^d$).

Недоліком алгоритму є те, що він обчислює шлях лише між початковою та кінцевою точками, на відміну від алгоритмів, які знаходять маршрути до всіх вершин.

### 1.2.4 Алгоритм Флойда-Воршелла

Алгоритм Флойда-Воршелла належить до алгоритмів динамічного програмування і використовується для визначення найкоротших відстаней між усіма парами вершин у зваженому орієнтованому графі. Алгоритм здатний працювати з графами, де ваги ребер можуть бути як додатними, так і від'ємними, але не допускає циклів із сумарно від'ємною вагою.

Цей підхід базується на розбитті вихідної задачі на менші частини, що вирішуються окремо, після чого їхні результати об'єднуються у повне рішення [7]. Таким чином, задача найкоротшого шляху через граф поступово розв'язується шляхом послідовного розгляду всіх можливих варіантів з використанням проміжних вершин.

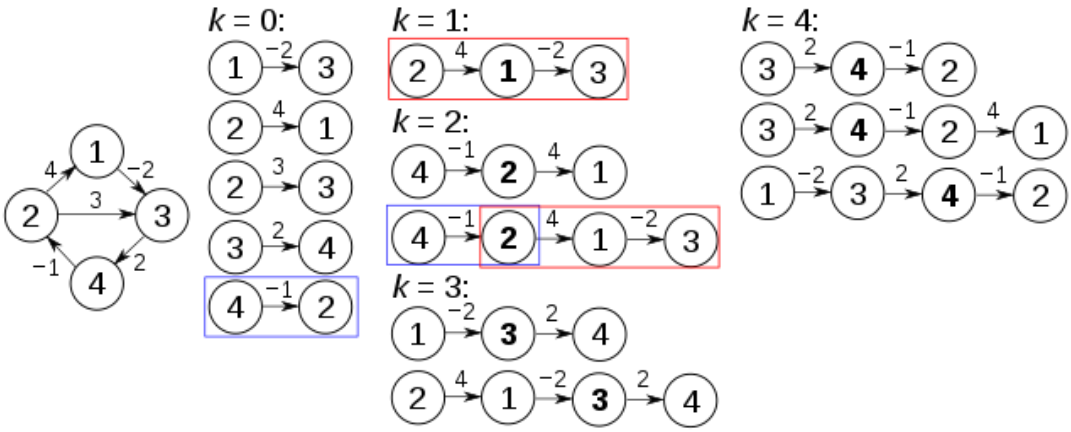


Рисунок 1.7 – Приклад роботи алгоритму Флойда-Воршелла

Реалізація алгоритму здійснюється за допомогою трьох вкладених циклів. Зовнішній цикл послідовно перебирає вершини графа, які можуть слугувати проміжними точками на шляху між іншими вершинами. Два внутрішні цикли аналізують усі можливі пари вершин, перевіряючи, чи можна покращити відстань між ними через поточну проміжну вершину. Якщо такий маршрут коротший, то відповідне значення в матриці відстаней оновлюється.

Алгоритм також враховує ситуації, коли прямий шлях між вершинами вже існує. У такому разі його довжина не змінюється, якщо не буде знайдено кращого варіанту через проміжну точку. У випадках, коли шлях між вершинами відсутній, створюється новий маршрут із використанням проміжної вершини.

Кінцевим результатом виконання алгоритму є заповнена матриця, яка містить найкоротші відстані між кожною парою вершин. У графах із від’ємними вагами алгоритм забезпечує коректний результат, якщо не виникають цикли з від’ємною сумарною вагою.

Найгірша швидкодія алгоритму досягає $O(V^3)$, коли використання пам’яті – $O(V^2)$.

### 1.2.5 Алгоритм Вітербі

Для визначення найбільш ймовірної послідовності станів у системах, де інформація про спостереження є неповною або неточною, застосовується алгоритм Вітербі. Цей алгоритм широко використовується для виправлення помилок у кодах зв'язку чи звукових даних [8], а також у таких галузях, як синтез мовлення та біоінформатика.

У сфері маршрутизації алгоритм дозволяє визначити оптимальний шлях для передачі даних у мережі, враховуючи такі параметри, як стан мережевих вузлів, пропускна здатність, затримка, і не тільки. На початку, маршрути моделюються у вигляді графа, де кожне ребро відповідає можливому переходу між вузлами. Далі формується таблиця, що зберігає ймовірності переходів, а також характеристики мережі, зокрема пропускну здатність і затримки.

Початковий етап передбачає заповнення першого стовпця таблиці, використовуючи початкові ймовірності станів та параметри з'єднань. Після цього таблиця поступово заповнюється згідно з формулою Вітербі, враховуючи всі фактори, що впливають на маршрути. Алгоритм здійснює обчислення від кінця таблиці до початку враховуючи параметри з'єднань, щоб визначити оптимальний маршрут.

Найгірша швидкодія алгоритму досягає $O(TN^2)$, коли використання пам'яті – $O(Tx|S|^2)$.

Алгоритм ефективний завдяки здатності враховувати різноманітні умови, що впливають на ймовірність вибору маршруту, наприклад, коли пропускна здатність знижена. Це робить алгоритм надійним у використанні, оскільки він мінімізує ймовірність помилок, забезпечуючи точний вибір оптимального шляху.

## 1.3 Аналіз евристичного алгоритму пошуку

Алгоритми, які виконують пошук рішень у просторі з обмеженням на перебирання для підвищення їх продуктивності називаються евристичними. Їх використовують для знаходження прийнятних рішень для великих NP-повних

проблем за помірний час. Однак знайдені маршрути можуть значно відрізнятися від оптимальних, і можуть вважатися наближеними.

Евристичні алгоритми поділяються на три основні категорії:

Конструктивні алгоритми, які будують розв’язок через спостереження за підвищенням вартості без подальших фаз покращення. Такими є алгоритми найближчого сусіду, жадібний, економії, та вставка.

Двофазними, або кластерними називають алгоритми, у яких задача ділиться на два: спочатку групуються вершини для кожного маршруту (кластеризація), а потім розв'язується задача пошуку найкоротшого шляху для кожної групи (TSP). Прикладами є алгоритм Фішера і Джайкумара, замітання, пелюсток, та Османа.

У таких алгоритмах можлива наявність зворотного зв'язку між етапами.

Покращуючі алгоритми: на першому етапі спочатку знаходиться деяке початкове рішення, а потім здійснюються спроби заміни вершин графів або ребер всередині кожного маршруту або між маршрутам. Прикладами є алгоритми внутрішнього маршруту, міжмаршрутний алгоритм, та k-opt, 2-opt, 3-opt.

Дані алгоритми дозволяють знайти допустимі рішення для складних завдань, зокрема в задачах маршрутизації, де точні рішення мають можливість бути ускладненими або навіть неможливими для знаходження через велику кількість варіантів.

### 1.4 Порівняння існуючих алгоритмів розв’язку

Різниці між розглянутими алгоритмами маршрутизації та їх характеристики наведено у таблиці 1.1.

Таблиця. 1.1 – Порівняння розглянутих алгоритмів пошуку маршрутів

| Алгоритм | Найгірша швидкодія | Використання пам'яті | Евристичний |
|---|---|---|---|
| Дейкстри | $O((V + E) \log V)$ | $O(V)$ | - |
| Беллмана-Форда | $O(\|V\|\|E\|)$ | $O(V)$ | - |
| A* | $O(\|E\| \log \|V\|) = O(b^d)$ | $O(\|V\|) = O(b^d)$ | + |
| Флойда-Воршелла | $O(V^3)$ | $O(V^2)$ | - |
| Вітербі | $O(TN^2)$ | $O(Tx\|S\|^2)$ | - |

Кожен алгоритм має свою унікальну складність, і багато популярних рішень не базуються на евристиці. Це зумовлено як складністю їхньої реалізації, так і надійністю інших підходів. Однак, варто зазначити, що правильно обрані евристичні алгоритми можуть значно пришвидшити розв'язання завдання. Саме тому алгоритм A* часто застосовують у поєднанні з алгоритмом Дейкстри для підвищення продуктивності, особливо при роботі з великими обсягами даних. Використання додаткових евристичних функцій дозволяє швидше знайти оптимальне рішення.

### 1.5 Обґрунтування методу розв’язання

Розглянуті алгоритми є найбільш поширеними рішеннями питання маршрутизації, застосування яких спостерігається і в інших сферах діяльності. Алгоритм Дейкстри гарантує надання найкоротшого маршруту між вершинами графів, але при цьому недієвий при наявності вершин з негативними вагами. Алгоритм Беллмана-Форда вирішує цю проблему, що дозволяє його використання у фінансовому моделюванні чи при маршрутизації, коли передбачається можливість зекономити і зменшити вартість пересування.

Алгоритм Флойда-Воршелла на додачу до роботи з вершинами з від’ємними вагами, розбиває задачу на підзадачі, вирішує їх окремо, та по завершенню поєднує результати одне рішення, коли алгоритм Вітербі використовується для визначення ймовірної послідовності неповної інформації у широких сферах застосування

Але у великих системах недоліком цих алгоритмів може бути перегляд зайвих вершин, що спричиняє збільшення часу та необхідного об’єму ресурсів для прорахунку маршруту [9].

Для вирішення цього недоліку, ефективним рішенням є використання евристичних алгоритмів на кшталт A*, ефективна реалізація та використання евристичних функції яких дозволяє значно зменшити обсяг використаної пам’яті та покращити продуктивність.

У розробленій системі виконується навігація по вершинам графу, які розташовані на мапі. Дані вершини отримують дані про висоту з зовнішнього джерела на основі своїх координат, що може бути використано як додатковий параметр для підвищення точності навігації. Вершини графів з’єднані між собою зв’язками, які відображають дороги місцевості та включають інформацію про відстань між точками, а також тип дороги – пішохідна чи автомобільна.

Оскільки система працює через з’єднання вершин у графі, створення складних графіків вимагає значних часових ресурсів. Це ускладнює задачу через необхідність перевірки великої кількості можливих маршрутів. Тому графи початково створювалися для невеликих ділянок місцевості, а згодом модифікувалися з урахуванням більшої кількості вершин і їхніх зв’язків.

Для розв’язку завдання та проведення дослідження було обрано алгоритм Дейкстри, який ефективно знаходить найкоротший шлях від стартової точки і до пункта призначення, та алгоритм A*, який з використанням допоміжної евристичної функції здатен врахувати додаткові параметри разом з координатами для визначення оптимального маршруту.

Для алгоритму A* було використано формулу Евклідової відстані для евристики, яка дозволяє виміряти традиційну відстань між двома точками у n-вимірному Евклідовому просторі.

Для точок p та q з координатами:

$$p = (p_1, p_2, \dots, p_n),$$

$$q = (q_1, q_2, \dots, q_n)$$

Для визначення Евклідової відстані використовується наступна формула:

$$D(p,q) = \sqrt{((p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2}$$

Евристика допомагає оцінити шлях між поточною двома вершинами, і її використання всередині алгоритму маршрутизації допомагає визначити найліпший маршрут до бажаної точки [10]. Використання Евклідової дистанції у тривимірному просторі дозволяє враховувати висоту до точки і уникнути неоптимальних маршрутів, а сама евристична функція є ефективною лише коли вона оптимальна – не переоцінює фактичну відстань [11].

В розробленій системі для маршрутизації з використанням алгоритму A* також використовується значення висоти отримане за допомогою API, за рахунок чого евристична формула була модифікована для прорахунку висоти. Враховуючи це, точка z має координати:

$$z = (z_1, z_2, \dots, z_n)$$

Коли формула для визначення Евклідової відстані модифікована під умови завдання для використання у тривимірному просторі і виглядає наступним чином:

$$D(p,q) = \sqrt{((x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2},$$

де $(x_1,y_1,z_1)$ і $(x_2,y_2,z_2)$ – координати точок p та q, а z представляє значення висоти цих точок.

Після додавання вершин графа, вибору початкової точки, та обранням користувачем пунктів через які необхідно пройти, система створює структуру, яка зберігає перелік усіх вершин, через які необхідно пройти. Стартова

вершина позначається як поточна. Далі розпочинається процес, що триває доти, доки залишаються невідвідані пункти призначення.

У кожному циклі обчислюються найкоротші дистанції між поточною точкою та всіма іншими пунктами призначення за допомогою заданого алгоритму пошуку найкоротшого шляху. Серед цих пунктів вибирається той, до якого відстань є мінімальною. Потім, із застосуванням бібліотеки D3, будується маршрут до вибраної вершини. Після цього поточна точка оновлюється на досягнуту вершину, а відповідний пункт видаляється зі списку.

Процес повторюється, якщо залишаються невідвідані пункти. Коли всі проміжні вершини пройдено, перевіряється, чи вказана кінцева точка маршруту. Якщо кінцева точка задана, система формує останній відрізок шляху від поточної вершини до кінцевої, використовуючи обраний алгоритм пошуку.

Після завершення маршрутизації користуваче показано створений маршрут разом із даними про його довжину та час, витрачений на розрахунок.

## 1.6 Огляд існуючих програмного забезпечення із функцією планування маршруту

### 1.6.1 QGIS

QGIS представляє з себе потужну та безкоштовну, вільну крос-платформену геоінформаційну систему, яка є однією з найбільш функціональних систем що розвиваються. Створена у 2002 році Гарі Шерманом та отримавши версію 1.0 у 2009 році, використовуючи C++, Python та бібліотеки Qt, основним її призначенням є обробка і аналіз просторових даних, підготовка різної картографічної продукції з підтримкою інтеграції плагінів [12].

Ряд плагінів надають можливість проводити маршрутизацію з використанням QGIS. Серед них - ORS Tools, який утилізує сервіс OpenRouteService для розрахунку маршруту з врахуванням транспортного засобу. Для цього використовується алгоритм Contraction hierarchies [13], який

пришвидшує пошуки найкоротшого маршруту для подорожей на автомобілях та для вирішення логістичних питань.

### 1.6.2 ArcGIS

ArcGIS – комплексна геопросторова платформа розроблена Environmental Systems Research Institute у 1982 році для дослідження географічної інформаційної системи, закономірностей, маршрутизації для бізнесу чи наукових середовищ різних робітничих галузей [14]. Написана з використанням мови програмування C++, за допомогою розширення ArcGIS Network Analyst, система дозволяє проводити маршрутизацію, та утилізує для пошуків маршруту ряд алгоритмів, які включають алгоритм Дейкстри, його різновиди (найближчий пункт обслуговування, матриця джерело-призначення), та евристичні алгоритми [15].

Завдяки доступності ArcGIS у форматі веб-додатку та застосунків для мобільних пристроїв та персональних комп’ютерах, система є ефективним рішенням яке пропонує всі компоненти, необхідні для побудови інфраструктури просторових даних. У ньому є засоби підготовки і ведення геоданих, засоби публікації веб-служб і ГІС-функціональності для віддаленого доступу, засоби створення каталогів геоданих і геопорталів.

### 1.6.3 Google Maps

Google Maps, створений у 2005 році, спочатку був розроблений як комп'ютерний застосунок мовою програмування C++. Наразі цей сервіс доступний на мобільних пристроях, а в його клієнтській частині використовуються JavaScript та XML.

Google Maps дозволяє користувачам планувати маршрути, обираючи різні види транспорту [16]: пішки, автомобілем, велосипедом або громадським транспортом. Система оптимізує автомобільні маршрути, використовуючи дані геолокації інших смартфонів через метод краудсорсингу. Це дозволяє визначати ділянки з заторами та пропонувати більш зручні альтернативи [17]. З метою економії трафіку Google Maps уникає передавання інформації використовуючи JSON.

В роботі застосунку задіяні два основні алгоритми: Дейкстри та A*. Пошук маршрутів за допомогою алгоритму Дейкстри може займати багато часу через велику кількість вершин графів, що використовуються для зображення світового з’єднання доріг. Окрім цього, цей алгоритм не враховує типів доріг, які можуть бути важливими для користувача. Для підвищення ефективності Дейкстра часто використовується разом із алгоритмом A* [18]. Завдяки своїй евристичній природі та орієнтованості на зважені графи, A* здатен враховувати різноманітні параметри, такі як відстань і дорожні умови. Це дозволяє системі ефективно працювати з великим обсягом даних і забезпечувати точні результати.

### 1.6.4 Apple Maps

Apple Maps, презентований у 2012 році, був створений для заміни попереднього додатку-навігатора і використання пристроями Apple [19]. Відразу після запуску додаток зіткнувся з критикою через застарілі карти та дані, що призводили до помилкових маршрутів і недостатньо точної інформації про громадський транспорт [20].

Сервіс надає користувачам можливість будувати маршрути з урахуванням зазначених точок, додавати нотатки щодо подорожі та використовувати голосові підказки віртуального асистента Siri.

Серед інших можливостей – рекомендації цікавих місць поруч із поточним місцезнаходженням користувача та інформація про завантаженість доріг.

Для розробників сервіс підтримує JSON для API, а фреймворк MapKit JS дозволяє інтегрувати Apple Maps у програми для iOS та Android за допомогою JavaScript.

### 1.6.5 Bing Maps

Bing Maps, представлений у 2005 році як Live Search Maps, є онлайн-сервісом маршрутизації, розробленим компанією Microsoft [21]. Додаток використовує технології .NET, що відповідає стратегії Microsoft .NET strategy.

Однією з головних переваг Bing Maps є можливість працювати з локальними базами даних, що дозволяє адаптувати карти для конкретних регіонів.

Користувачі можуть планувати маршрути через кілька вибраних точок із врахуванням виду транспорту та пересадок, що дозволяє створювати оптимальні шляхи. Система також підтримує внесення даних іншими користувачами, завдяки чому на карті можна знаходити цікаві місця, будівлі та інші об'єкти. Завдяки технології ClearFlow, Bing Maps надає актуальну інформацію про завантаженість доріг і пропонує альтернативні маршрути, що можуть бути ефективнішими за стандартні.

Для розробників Bing Maps пропонує інструменти створення застосунків для Android та iOS з використанням технологій змішаної реальності та Streetside. API, що підтримують JavaScript і TypeScript, дозволяють інтегрувати сервіс у веб-додатки. Крім того, є API з використанням REST URL для вирішення геолокаційних завдань і засоби роботи з даними у Windows Presentation Foundation (WPF). Завдяки Windows 10 Universal Platform розробники можуть створювати додатки для пристроїв на Windows 10 із використанням таких мов програмування, як C# і XAML.

### 1.6.6 Waze

Waze — це мобільний додаток для навігації, який надає дані про дорожню ситуацію, маршрути та інші корисні функції в режимі реального часу [22].

Ця програма використовує складний алгоритм для визначення найоптимальніших маршрутів, враховуючи актуальні умови на дорогах, аварії, затори та інші перешкоди. Waze отримує інформацію від своїх користувачів, які передають дані про трафік, швидкість руху та можливі перепони. Ці відомості використовуються для оперативного оновлення маршрутів, що дозволяє уникати заторів і скорочувати час подорожі. Користувачі також можуть додавати кілька пунктів призначення, плануючи маршрут через кілька проміжних точок.

Додаток здобув популярність завдяки зручному інтерфейсу, схожому на елементи відеоігор, а також завдяки наявності розважальних функцій. Крім мобільної версії, Waze має веб-інтерфейс, який дозволяє користувачам визначати кілька варіантів маршруту між заданими точками на карті, а потім синхронізувати ці дані з мобільним додатком. Для інтеграції на сторонніх вебсайтах Waze пропонує функцію вставки карти через embed-посилання, а маршрути можна ділитися окремо для зручності користувачів.

### 1.6.7 Routific

Routific — це навігаційна система, спеціально розроблена для оптимізації процесів доставки кур'єрами або працівниками сервісів, які здійснюють перевезення та доставку товарів [23]. Вона дозволяє диспетчерам легко розподіляти замовлення між водіями, враховуючи встановлені часові рамки. Зазначивши час доставки та відповідального водія, система автоматично формує маршрут, який мінімізує час поїздки, забезпечуючи доставку у визначений термін.

У разі потреби диспетчер може перенаправити замовлення іншому кур'єру. Система одразу враховує зміни та оновлює маршрут відповідно до нових умов. Водії можуть стежити за своїми призначеними завданнями та місцями доставки через мобільний додаток, а отримувачі мають змогу відслідковувати статус замовлення та місцезнаходження кур'єра в реальному часі.

Крім цього, Routific автоматизує процес обробки замовлень, розподілу завдань і контролю їх виконання, використовуючи різні засоби, включно з фотографіями для підтвердження доставки.

Система надає API для інтеграції своїх функцій у сторонні проєкти [24]. Завдяки цьому інструменту, який використовує машинне навчання для підвищення ефективності, розробники можуть оптимізувати маршрути відповідно до конкретних потреб, використовуючи мови програмування, такі як PHP, Ruby, Python, Node.js та cURL.

### 1.6.8 Roadtrippers

Roadtrippers — це веб-додаток для автомобільних подорожей по території Північної Америки [25]. Однією з головних особливостей системи є показ туристичних місцевостей та пам’яток та прокладання маршруту, що проходить через ці точки. Користувач може вказати початкову, кінцеву та проміжні точки, а система побудує маршрут і покаже його на карті з позначенням рекомендованих місць.

Roadtrippers також рекомендує туристичні локації, враховуючи побажання користувача. На етапі планування, система розраховує додаткову корисну інформацію, таку як час проходження маршруту чи витрати пального.

Відгуки користувачів та наданий рейтинг локацій дозволяє визначити та обрати локації через які можна пройти. Надається можливість поширення маршруту та організацію подорожі з іншими користувачами. Підтримується робота з системою навігації автівок для зручнішої маршрутизації.

## 1.7 Порівняння існуючих навігаційних систем

Порівняння функцій, врахованих даних, та унікальні функції згаданих прикладів представлено в таблиці 1.2.

Таблиця 1.2 – Порівняння навігаційних систем

| Система | Маршрутизація через декілька точок | Вибір транспорту | Врахування навантаження трафіку | Додатковий функціонал |
|---|---|---|---|---|
| QGIS | + | + | | Плагіни для вирішення логістичних питань |
| ArcGIS | + | + | | ГІС-функціональність для побудови інфраструктури просторових даних |
| Google Maps | + | + | + | |
| Apple Maps | + | + | + | |
| Bing Maps | + | + | + | |
| Waze | Через мобільний додаток | + | + | |
| Routific | + | Лише автомобілем | + | Комерційне використання |
| Система побудови маршрутів | + | + | | |

Згадані системи об'єднує здатність проведення шляху через декілька вказаних точок і показ ефективних результатів. Також частина з них підтримує впровадження до інших рішень за рахунок API та інструментів для розробників. Включення специфічних умов пересування в підрахунки є важливою частиною роботи багатьох з цих систем. Такі додатки як Routific потребують ускладнених вимагає методів маршрутизації, що потребує зважання не тільки на дорожні умови, а й такі фактори, як час доставки, а й відстань до точок призначення та місцезнаходження кур’єрів. Через це, система ефективно оптимізує маршрути для доставок, забезпечуючи точність

і своєчасність. Зважаючи на великі масштаби локацій, які потрібно обробити, поширені методи маршрутизації можуть витрачати значну частину часу, через що існуючі системи можуть використовувати алгоритм A* чи інші евристичні рішення, що спрямовують пошук у правильному напрямку та дозволяють швидше знаходити найкращий маршрут. Додатково, інші системи, такі як QGIS, використовують алгоритм Contraction Hierarchies, який дозволяє ефективніше знаходити оптимальні маршрути для автомобільних подорожей і логістичних завдань.

### 1.8 Мікросервісна архітектура

Мікросервісна архітектура – це підхід до розробки програмного забезпечення, за яким продукт складається з незалежних компонентів – мікросервісів. Така архітектура реалізує отримання даних із зовнішніх джерел та розширення набору функцій чи інтеграцію до інших систем [26]. На відміну від традиційних ГІС-інструментів, забезпечується отримання актуальних даних, гнучка масштабованість системи та можливість використання даних з інших систем. Недоліком архітектури є залежність від інших сервісів для роботи та окреме збереження даних для кожної з них, коли ГІС-рішення незалежні від інших сервісів, та пропонують високу функціональність за рахунок обмеженості імплементації з іншими системами та залежності від локального середовища.

Існують різні шаблони побудови мікросервісних систем [27].

#### 1.8.1 База даних на сервіс

Кожний сервіс всередині системи може залежати від даних та як вони постачаються, що впливатиме на їх продуктивність. У випадку, коли на кожен сервіс виділена окрема база даних, то дані ізольовані від інших сервісів, спрощуючи їх підтримку та забезпечуючи працездатність решти системи у випадку непередбачуваного збою. Однак для кожної бази даних необхідний запобіжний механізм у випадках коли зв’язок між нею та відповідним сервісом невдалий.

Коли на кожний сервіс призначена одна база даних, можливість масштабування системи значно зменшується, роблячи кожний її елемент залежним від одного чинника. Хоча це може надавати переваги в окремих випадках, для сервісів вимагається налаштування прав доступу та обмежень для запобігання впливу на дані не пов’язаних з нею.

### 1.8.2 Saga

Шаблон Saga допомагає забезпечувати постійність у даних шляхом управління серіями транзакцій між сервісами. Розрізняють два підходи до його виконання:

- Хореографія – сервіс виконає транзакцію, а вже після публікуватиме подію. В деяких випадках, інші сервіси відповідатимуть на події та виконуватимуть відповідні інструкції, і за потреби також надаватимуть події.
- Оркестрація (Orchestration), коли транзакції та події виконуються та викликаються окремим об’єктом – оркестратором.

Saga — це складний шаблон проєктування, перевага належної реалізації якого полягає в збереженні узгодженості даних у кількох сервісах без сильного зв’язку між ними.

### 1.8.3 API Gateway

Для великих систем із декількома клієнтами можна використовувати API gateway є ефективним рішенням. Однією з найбільших переваг є те, що клієнту не потрібно знати як розділено сервіси. Шаблон надає єдину точку входу для групи сервісів.

Шаблон дозволяє розділити доступ до систем на різні типів клієнтів (наприклад, серверні частини для зовнішніх інтерфейсів), та забезпечує безпеку за рахунок того, що API endpoints не піддаються безпосередньому впливу клієнтів, а авторизація і SSL можуть бути ефективно реалізовані.

Шаблон можна використовувати для відокремлення внутрішніх сервісів від клієнтських програм. Це гарантує, що весь запит не завершиться

через те, що один мікросервіс не відповідає. Для цього використовується кеш, щоб надати порожню відповідь або повернути код помилки.

### 1.9. Обґрунтування вибору мови програмування

Мови програмування JavaScript, C#, і Python часто використовуються для створення Web-систем та проєктів.

#### 1.9.1 C#

Об’єктно орієнтована мова програмування C# була розроблена Microsoft для власної платформи .NET. Її активно застосовують для створення різних видів програмного забезпечення, таких як Windows-програми, Web-додатки, мобільні програми, ігри тощо. Синтаксис C# відрізняється своєю чистотою та зрозумілістю, що спрощує процес розробки [28]. Мова надає розробникам доступ до великої кількості бібліотек і фреймворків, які значно прискорюють створення проєктів. Крім того, C# підтримує багатопоточність, що дозволяє ефективно працювати з паралельними задачами.

#### 1.9.2 Python

Інтерпретована мова програмування Python зосереджена на забезпеченні простоти та зрозумілості створення коду. Її застосування охоплює широкий спектр сфер, таких як веб-розробку, і дозволяє ефективне використання для обчислення даних дослідження, розробки штучного інтелекту, автоматизації, тощо [29]. Python також часто використовується для навчання програмуванню завдяки своєму простому та інтуїтивно зрозумілому синтаксису. Стандартна бібліотека включає модулі для виконання багатьох типових завдань, а також підтримує багатопоточність і інтеграцію зі сторонніми бібліотеками, що робить її універсальним вибором для розробників.

#### 1.9.3 JavaScript

JavaScript — це популярна скриптова мова програмування, розроблена компанією Netscape у 1995 році для інтерактивної взаємодії з вебсторінками у браузерах. З часом JavaScript стала основною мовою програмування для створення вебдодатків. Її можливості охоплюють розробку динамічних

елементів вебсторінок, отримання та валідацію даних із форм, анімацію, інтерактивність із користувачами, а також створення вебфреймворків та бібліотек [29].

JavaScript є універсальною завдяки широкій підтримці сучасними браузерами і можливості працювати безпосередньо у них, що дозволяє взаємодіяти зі сторінками без необхідності встановлення додаткових плагінів. Крім роботи в браузерах, JavaScript активно використовується для серверної розробки завдяки платформі Node.js. Це забезпечує їй можливість функціонувати як на стороні клієнта, так і на стороні сервера, що робить мову ідеальною для створення повноцінних вебдодатків.

Оскільки JavaScript є універсальною мовою з підтримкою багатьох методів і технологій, вона поєднує в собі переваги, характерні для Python і C#. Завдяки цьому, а також через наявний досвід роботи з JavaScript, саме цю мову програмування було обрано для розробки системи.

## 1.10 Обґрунтування вибору бібліотек

### 1.10.1 Leaflet

Leaflet — це бібліотека з відкритим кодом, яка надає розробникам на JavaScript широкий набір функцій для роботи з картами. Її головними перевагами є ефективність, оптимізованість для всіх платформ, включаючи мобільні пристрої, підтримка плагінів і простий, добре документований API.

Через ці особливості, а також її безкоштовність і зручність у використанні, для розробки системи було обрано саме Leaflet. Ця бібліотека дозволяє додавати вершини графів на карту місцевості, а також розраховувати маршрути відповідно до координат та відстаней, заданих користувачем.

### 1.10.2 jQuery

jQuery — це бібліотека з відкритим кодом, створена для спрощення взаємодії JavaScript із вебсторінками. Вона оптимізує процес отримання даних із HTML-документів і забезпечує функції для анімації через CSS та технік програмування Ajax.

Для роботи системи jQuery було обрано через її здатність спрощувати отримання даних із головної сторінки системи. Вона допомагає швидко приймати запити користувачів щодо бажаних призначень і параметрів маршруту, що робить взаємодію зі сторінкою більш ефективною.

### 1.10.3 Bootstrap

Bootstrap — це бібліотека з відкритим кодом, яка полегшує створення HTML-сторінок. Вона включає шаблони для HTML і CSS, а також інтегровані плагіни JavaScript, що пришвидшує роботу з вебінтерфейсами.

Цей фреймворк було використано для створення простого, ефективного інтерфейсу системи. Bootstrap дозволяє легко налаштовувати зовнішній вигляд сторінки, приймати введені користувачем конфігурації та показувати результати роботи системи. У майбутньому цей інтерфейс може бути вдосконалено.

### 1.10.4 D3.js

D3.js (Data-Driven Documents) — це бібліотека для створення інтерактивних і динамічних візуалізацій даних у вебдодатках. Вона використовує Scalable Vector Graphics (SVG), HTML і CSS для ілюстрації даних у вигляді графіків або графів.

У системі D3.js використовується для динамічного додавання вершин графів на карту місцевості, їх візуального оновлення та переміщення за потреби. Це дозволяє створювати точну навігацію й інтерактивну візуалізацію маршрутів, підвищуючи загальну функціональність і зручність роботи із системою.

### 1.10.5 PostgreSQL

PostgreSQL є потужною реляційною базою даних яка підтримує SQL та JSON запити [30]. База даних підтримує зовнішні ключі, тригери, збережені процедури для виконання складних обчислень та забезпечує можливість користувачам переглядати зміст бази даних у режимі реального часу [31].

PostgreSQL дозволяє зберігати данні про вершини графів для системи разом з інформацією про шляхи, їх стан та деталі щодо них – цікаві локації і

не тільки незалежно від самої системи і ефективно редагувати та реалізовувати інформацію в програмі чи перевірити їх стан.

### 1.10.6 Open-Elevation API

Open-Elevation – безкоштовна альтернатива сервісу Google Elevation API та подібних до нього з відкритим кодом, яка пропонує доступ до наборів даних з інформацією про висоту точки координат на місцевості [32]. З використанням публічного API або через налаштування сервісу та даних з власної сторони, користувач здатен при введені координат отримати необхідне значення про висоту рельєфу.

В контексті маршрутизації, використання Open-Elevation API дозволяє отримати дані про вершини графів за їх координатами для майбутнього використання для визначення необхідних маршрутів та забезпечення оптимального шляху.

## 1.11 Обґрунтування вибору середовища розробки

Для розробки програмного забезпечення на мові JavaScript існує кілька популярних середовищ розробки, серед яких Visual Studio та Visual Studio Code. Обидва інструменти мають свої особливості та переваги, які було розглянуто нижче.

### 1.11.1 Visual Studio

Visual Studio — це інтегроване середовище розробки (IDE), створене компанією Microsoft, яке підтримує розробку на різних платформах, таких як Windows, .NET Framework, ASP.NET, а також на мовах C++, C#, JavaScript та інших.

Головними перевагами Visual Studio є підтримка широкого спектра мов програмування, наявність численних розширень, плагінів, комфортного візуального редактору графічних інтерфейсів, що спрощує дизайн UI, та ефективні інструменти налагодження.

Потужний набір функцій та широкий спектр можливостей роблять Visual Studio універсальним середовищем для розробки різних систем. Воно

підходить для масштабних проєктів, але може бути громіздким для більш легких задач.

### 1.11.2 Visual Studio Code

Visual Studio Code є легким редактор коду від Microsoft. На відміну від Visual Studio, це не повноцінне IDE, а редактор із підтримкою розширень, код якого у відкритому доступі.

Особливостями редактору є працездатність на всіх популярних операційних системах, що забезпечує кросплатформену підтримку, багатий вибір розширень, які дозволяють налаштовувати редактор для роботи з різними мовами програмування та технологіями [33], та простий і зрозумілий інтерфейс, зручний як для новачків, так і для досвідчених розробників.

Висока швидкість роботи навіть на великих проєктах робить редактор популярним вибором серед розробників.

Враховуючи простоту використання, легкість, кросплатформеність і мету дослідження, систему маршрутизації було розроблено з Visual Studio Code. Редактор забезпечує достатню функціональність для роботи з JavaScript і дозволяє розширювати свої можливості завдяки гнучким налаштуванням і великій кількості доступних розширень.

## 1.12 Висновки до першого розділу

У розділі було проведено огляд найвідоміших алгоритмів для знаходження найкоротшого шляху між заданими точками. Розглянуто алгоритми Дейкстри, Беллмана-Форда, A*, метод Флойда-Воршелла та алгоритм Вітербі, а також проведено їх порівняння за продуктивністю та принципами роботи. Поширені алгоритми демонструють здатність забезпечувати оптимальні маршрути за різних обсягів даних, тоді як евристичні підходи дозволяють скоротити час виконання завдяки використанню додаткових функцій. Для реалізації системи були обрані алгоритми Дейкстри та A*, оскільки їхні різні підходи та відповідність масштабам системи дають змогу оцінити їхню ефективність.

Окремо було вивчено популярні навігаційні системи, їхні функції, методи розробки та особливості. Зроблено висновок, що більшість таких систем пропонують схожий функціонал, регулярно вдосконалюються та інтегрують нові можливості для вирішення логістичних задач. Найбільш розповсюдженим підходом до побудови маршрутів є комбіноване застосування класичних пошукових алгоритмів та евристичних методів. Такий підхід мінімізує час на пошук оптимального шляху. Зокрема, алгоритм Дейкстри потребує більше часу для обробки всіх варіантів, тоді як метод A* використовує додаткові функції для врахування характеристик дороги, напрямку руху та завантаженості маршруту, що дозволяє адаптувати маршрутизацію до змін на дорогах.

Дослідження також підкреслило важливість збору даних для роботи навігаційних систем. Проблеми Apple Maps у перші роки після запуску показали, як недостатній обсяг інформації може вплинути на якість роботи та репутацію продукту. Сучасні сервіси активно використовують краудсорсинг для збирання даних. Наприклад, аналіз геолокаційної інформації пристроїв дозволяє визначати завантаженість доріг, а звіти користувачів допомагають додавати нові точки інтересу чи відзначати дорожні пригоди.

Популярні сервіси також пропонують засоби для розробників, зокрема набори інструментів для JavaScript. Хоча самі системи створювались на різних мовах програмування (наприклад, Bing Maps — на C#, відповідно до політики Microsoft), надані SDK дозволяють створювати додатки для різних платформ або інтегрувати функціонал у сторонні програми.

Було розглянуто мікросервісну архітектуру системи, її переваги, зауваження до реалізації, та найбільш поширені шаблони.

Завдяки огляду було визначено основні особливості популярних систем, підходи до маршрутизації, а також мови програмування, що застосовуються для їх розробки. Зібрані дані дозволяють обрати оптимальні методи реалізації системи, відповідні інструменти та забезпечити наявність необхідних прикладів і документації для подальшого розвитку системи.

# 2. АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 2.1 Структура системи

Розроблена система отримує карту місцевості від зовнішнього сервісу. Після завантаження необхідних даних карта відображається на інтерфейсі користувача, а всі наступні обчислення виконуються в межах самої системи. Вершини графів зберігаються в окремому масиві, який використовується для подальшого відображення на отриманій карті в інтерфейсі системи. Зміни, що відбуваються на карті, синхронізуються з графами та маршрутами, щоб гарантувати їх точне позиціювання відповідно до місцевості.

На початку маршрутизації, всі дані, введені користувачем, передаються до серверної частини системи, де виконується обробка інформації. На цьому етапі для знаходження оптимального маршруту, система проводить прорахунок шляху між вершинами графів. Отриманий результат повертається до інтерфейсу, де новий маршрут відображається на карті. Поруч з ним надається додаткова інформація про шлях, зокрема її деталі та характеристики. Структурну схему системи представлено на рисунку 2.1.

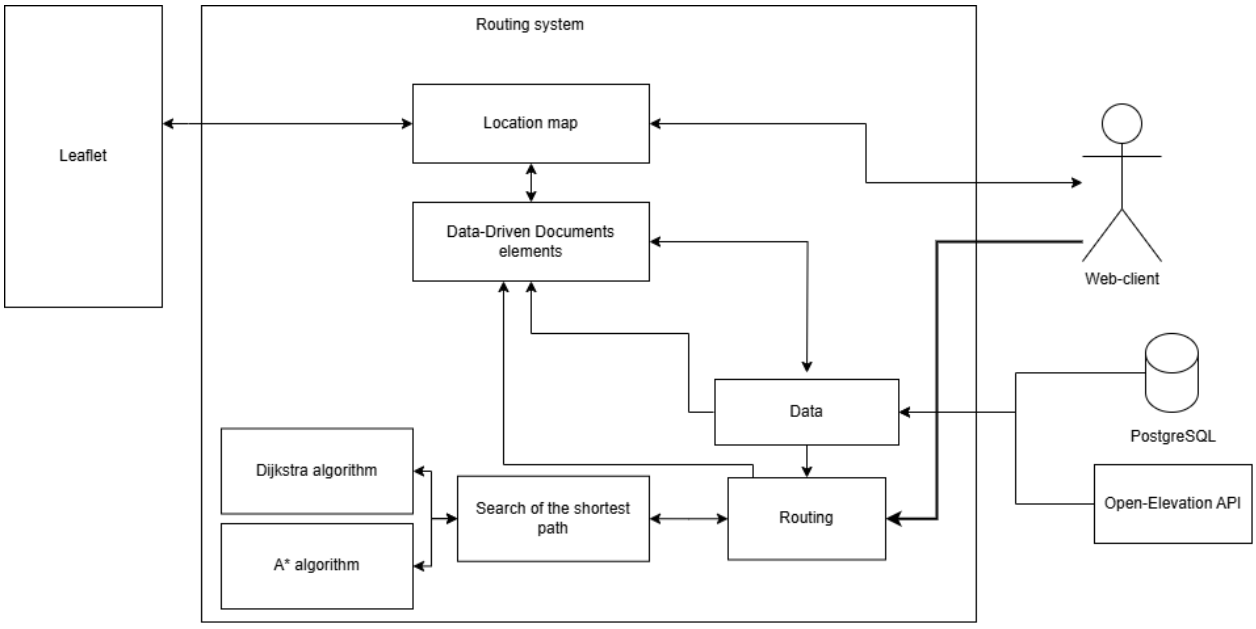


Рисунок 2.1 – Структурна схема

### 2.2 Мікросервісна архітектура системи

Схема мікросервісної архітектури системи наведено на рисунку 2.2.

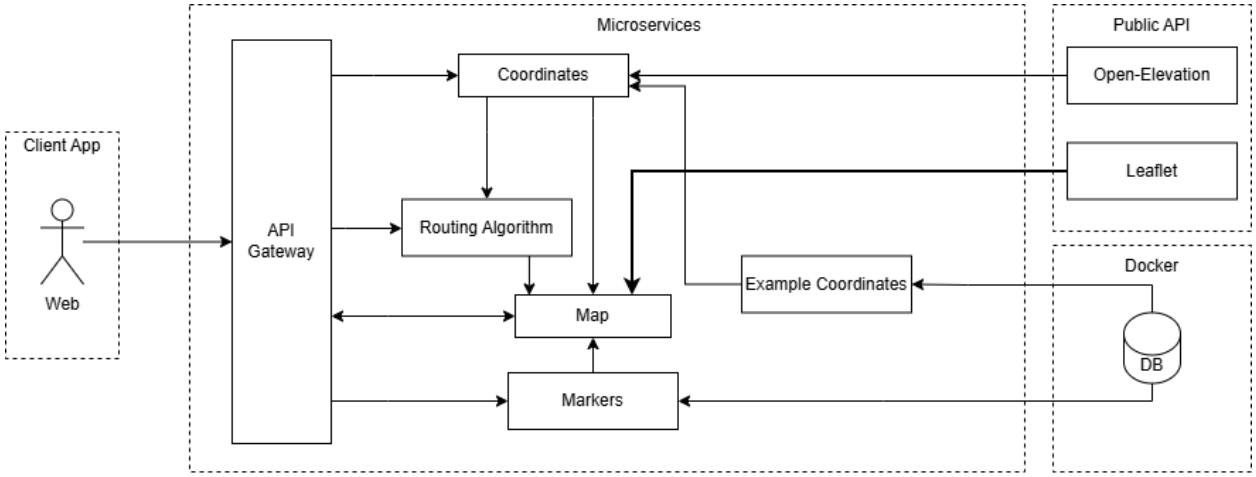


Рисунок 2.2 – Схема мікросервісної архітектури системи

Реалізовані сервіси отримують інформацію незалежно один від одного за рахунок використання окремих публічних API для отримання актуальної інформації щодо місцевості та бази даних створених з використанням контейнеру Docker спеціально для збереження даних прикладів та координат для використання в маршрутизації. Ця інформація надалі передається відповідним сервісами для подальшої обробки і зображення користувачу через карту на головній сторінці системи.

Для створення контейнеру Docker використовується офіційний образ postgres, в який надалі додаються таблиці з прикладами вершин графів та локацій для маркерів.

### 2.3 Діаграма компонентів

Компонентна діаграма, створена за допомогою Unified Modeling Language (UML), демонструє взаємодію основних компонентів системи, включаючи використані бібліотеки та фреймворки.

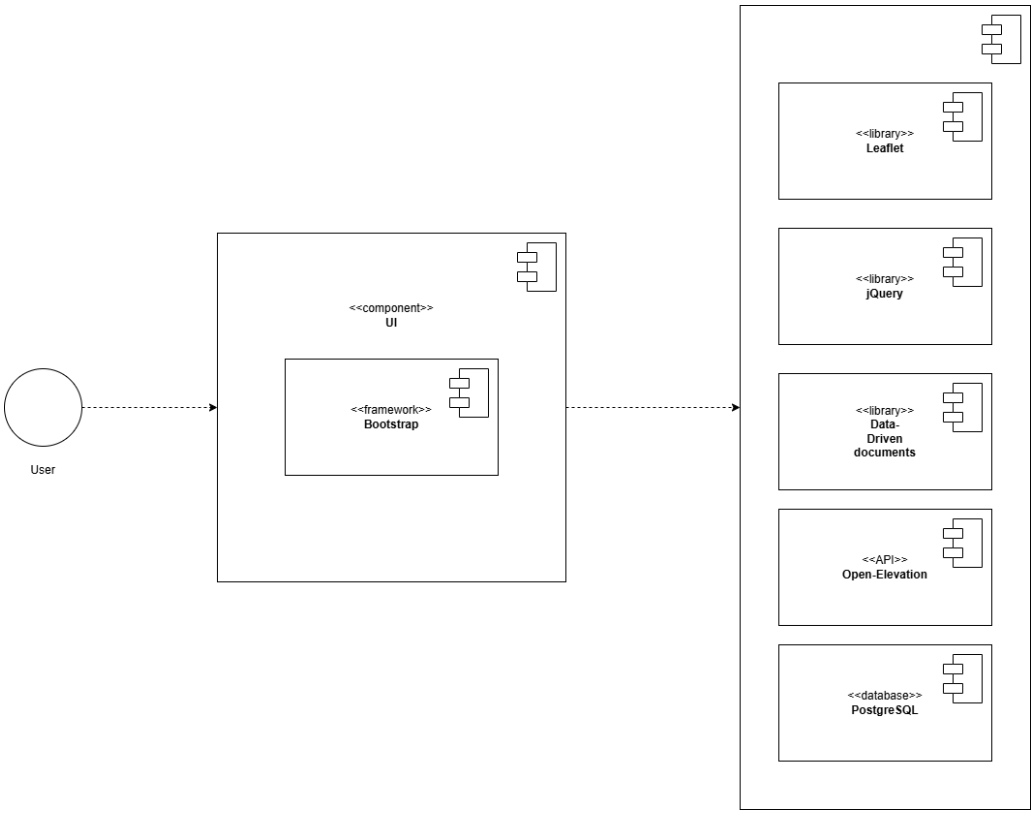


Рисунок 2.3 – Діаграма компонентів

У системі використано бібліотеку Bootstrap для створення зручного та адаптивного інтерфейсу користувача, jQuery для передачі даних з web-сторінки до серверної частини та обробки запитів користувача, Leaflet для візуалізації картографічних даних., Data-Driven Documents для динамічної візуалізації інформації, включаючи вершини графів, PostgreSQL як базу даних для зберігання інформації про цікаві місця та запити користувача, та Open-Elevation API для надання необхідних даних про висоту вершин графів.

Ці компоненти спільно забезпечують взаємодію між користувачем, сервером та зовнішніми API, дозволяючи ефективно працювати з даними маршрутизації.

### 2.4 Діаграма станів

Діаграма станів, зображена на рисунку 2.4, побудована за допомогою UML, ілюструє зміни станів системи у процесі її роботи, включаючи події, що викликають переходи між станами.

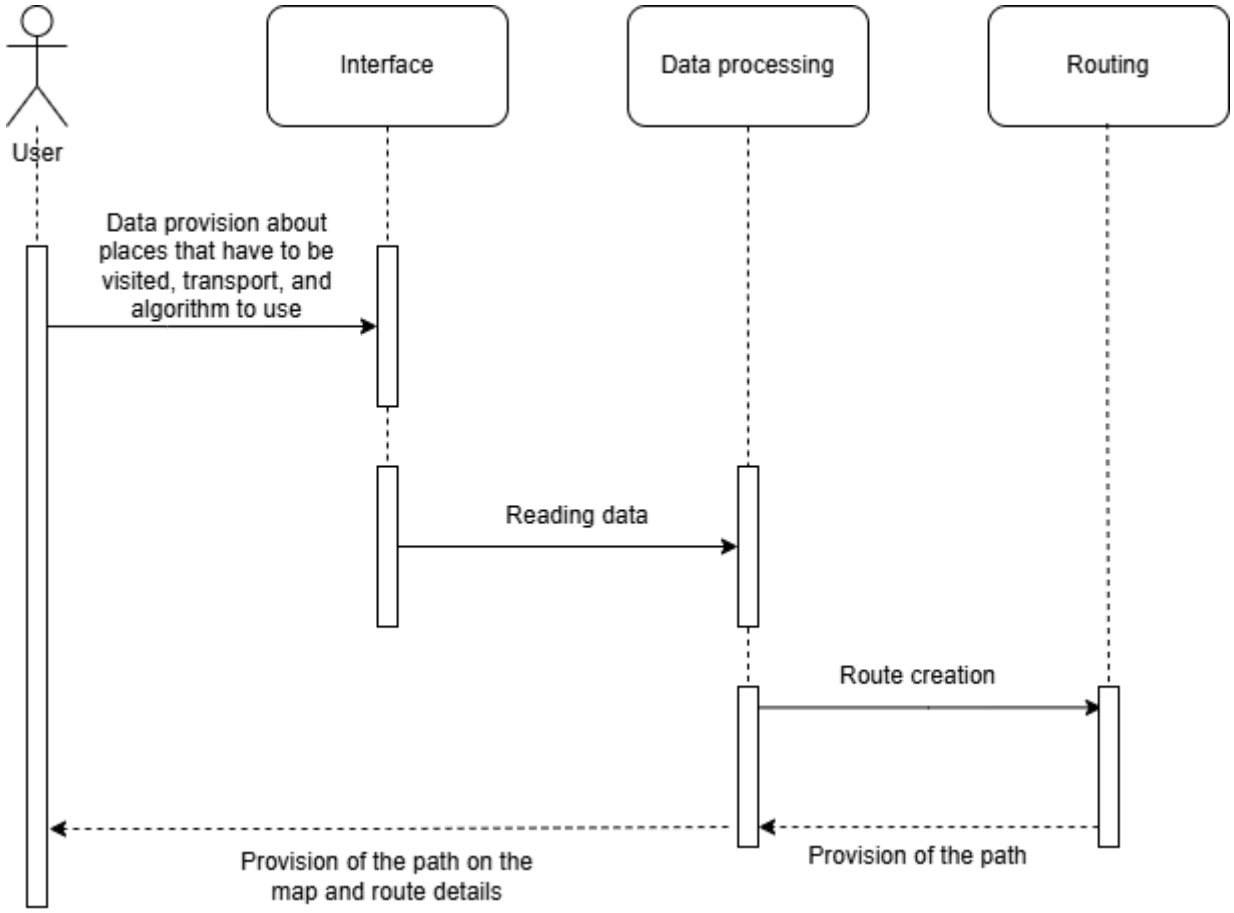


Рисунок 2.4 – Діаграма станів

Діаграма відображає очікування введення даних користувачем, обробку даних користувача та запитів до зовнішніх сервісів, і надання результатів роботи користувачеві. Завдяки діаграмі станів можна простежити основні етапи роботи системи та зрозуміти взаємозв'язки між подіями, які викликають зміну стану.

### 2.5 Діаграма послідовності

Діаграма послідовності у форматі UML надає покроковий огляд дій системи, починаючи з введення даних користувачем і закінчуючи отриманням результатів.

Діаграму для системи побудови маршрутів наведено на рисунку 2.5.

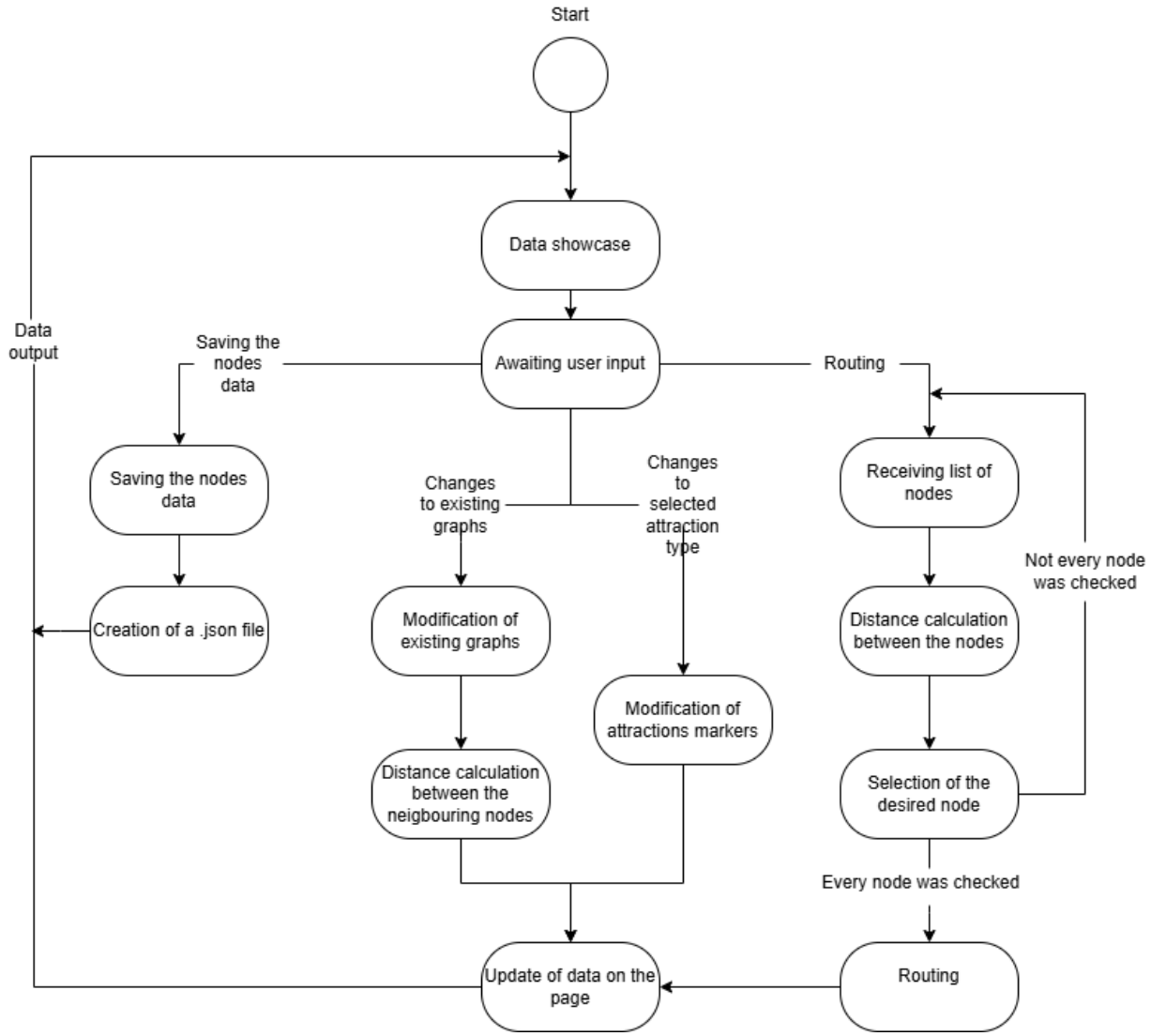


Рисунок 2.5 – Діаграма послідовності

На діаграмі зображено передачу даних із веб-сторінки до серверної частини, форматування отриманих даних у придатний для обробки вигляд, застосування алгоритмів для обчислення маршруту, та повернення результату роботи користувачеві через інтерфейс. Ця діаграма спрощує розуміння роботи системи та забезпечує покрокове уявлення про взаємодію компонентів і виконання запитів.

### 2.6. Вхідні дані

У цьому проєкті було використано відкриту бібліотеку OpenStreetMap, яка використовує open collaboration для підтримки. Інформація отримується з різних відкритих джерел, опитувань, а також аерофотозйомок. Зображення

місцевостей було введено в систему за допомогою бібліотеки Leaflet завдяки відсутності обмежень доступу до цих ресурсів.

При додаванні нових точок на карту, їхні дані, включаючи назву вершини та її координати, записуються до масиву allnodes. У разі зміни розташування існуючих вершин, координати оновлюються до нових значень. Зі створенням взаємозв'язків між точками, інформація про ці зв'язки додається до масиву paths, який містить дані про дорогу між вершинами, такі як ідентифікатор зв'язку, початкову та кінцеву вершини.

Система підтримує імпорт даних у через файли .json. Коли файл містить дані у потрібному форматі, система автоматично зчитує інформацію про масиви allnodes та paths, забезпечуючи їх доступність для подальшого використання або калькуляцій.

Аналогічно, завдяки підтримки бази даних PostgreSQL, користувач має нагоду отримати дані з прикладу який вже наявний у базах даних, разом з отриманням інформації про цікаві локації чи пам'ятки.

Після отримання даних надається можливість обрати алгоритм пошуку, транспортний засіб, та точки, по яким потрібно пройти. Ці дані передаються в систему.

### 2.7 Вихідні дані

На основі отриманих даних про точки та зв'язки між ними, система забезпечує відображення актуальної інформації на мапі. Всі графи представлені на карті разом із візуалізацією їхніх зв'язків та дистанції. Система надає список доданих вершин графів, завдяки якому можна обрати список початкової так кінцевої точок, разом з проміжними.

На початку маршрутизації, система прораховує відстані між з'єднаними вершинами графів. Ці дані використовуються в обраному методі для знаходження найближчої точки від початкової. Під час створення шляху, система відображає на карті шлях, який відповідає заданим критеріям користувача, використовуючи Data-Driven Documents. Водночас, надаються дані про маршрут, його дистанція та час витрачений на його прорахунок.

Для подальшого використання та зручності, існує можливість експортувати побудовані точки та дороги між ними у .json файл та збережені на пристрої. За рахунок цього, отримані дані можна імплементувати як готовий приклад як інші вже наявні у базі даних.

### 2.8 Структура даних

Дані про точки та зв’язки між ними можуть створюватися користувачем вручну, завантажуватися з готового файлу, отримуватися з бази даних або зчитуватися з раніше експортованого .json файлу. Всі джерела зберігають в собі інформацію про вершини графів і зв’язки між парами у вигляді двох масивів. Приклад масивів зображено у таблицях 2.1 і 2.2

Таблиця 2.1 – Масив точок nodes

| ID | Name | X | Y |
|---|---|---|---|
| 1 | 0 | 41.005050 | 28.974914 |
| 2 | 1 | 41.006350 | 28.976255 |
| 3 | 2 | 41.007212 | 28.977060 |
| 4 | 3 | 41.006718 | 28.977955 |
| 5 | 4 | 41.005791 | 28.976995 |
| 6 | 5 | 41.005710 | 28.974839 |
| 7 | 6 | 41.005309 | 28.974356 |
| 8 | 7 | 41.006447 | 28.973417 |

Таблиця 2.2 – Масив зв’язків між точками paths

| ID | from | to | type |
|---|---|---|---|
| 0 | 0 | 6 | Automobile |
| 1 | 0 | 1 | Automobile |
| 2 | 0 | 27 | Automobile |
| 3 | 27 | 26 | Automobile |

Кінець таблиці 2.2.

| ID | from | to | type |
|---|---|---|---|
| 4 | 26 | 25 | Automobile |
| 5 | 25 | 24 | Automobile |
| 6 | 61 | 62 | Pedestrian |

Для відображення точок на карті місцевості потрібні координати розташування вершин. За ними, дані будуть показані на інтерфейсі користувача та будуть проводитися прорахунки відстаней між вершинами. Для створення шляхів вершин потрібні дані про взаємозв’язки між ними.

При імпортуванні та додавання даних маршрутизації а також вершин, викликається функція збагачення вершин даними про висоту по їх координатам для покращення майбутньої маршрутизації. Приклад оновлених даних вершин графів наведено у таблиці 2.3.

Таблиця 2.3 – Оновлений зміст масиву вершин графів nodes

| ID | Name | X | Y | Elevation |
|---|---|---|---|---|
| 1 | 0 | 41.005050 | 28.974914 | 748 |
| 2 | 1 | 41.006350 | 28.976255 | 762 |
| 3 | 2 | 41.007212 | 28.977060 | 762 |
| 4 | 3 | 41.006718 | 28.977955 | 778 |
| 5 | 4 | 41.005791 | 28.976995 | 771 |
| 6 | 5 | 41.005710 | 28.974839 | 748 |
| 7 | 6 | 41.005309 | 28.974356 | 748 |
| 8 | 7 | 41.006447 | 28.973417 | 750 |

Завдяки використанню двох масивів можна забезпечити повну інформацію, необхідну для прокладання шляхів на карті та подальшого їх зображення та виведення інформації прорахунку.

Щоб відобразити маркери місць, система використовує інформацію збережену в PostgreSQL з масивом даних, які містять координати, додаткову

інформацію про місце і його рейтинг. Модифікація цих маркерів користувачем недоступна, оскільки вони додаються до системи через серверну частину.

Зміст масиву з маркерами локацій можна знайти в таблиці 2.4.

Таблиця 2.4 – Зміст масиву локацій places

| ID | coordinates | category | title | rating |
|---|---|---|---|---|
| 0 | [50.465780, 30.514260] | Restaurants | O'Connor's Irish Pub | 3.5 |
| 1 | [50.465900, 30.515470] | Restaurants | Lvivski Plyatski | 4.5 |
| 2 | [50.4649562, 30.514688] | Hotels | Podol Plaza Hotel | 4.1 |
| 3 | [50.469664, 30.520292] | Hotels | Mackintosh Hotel | 4.4 |
| 4 | [50.4689, 30.5075] | Attractions | St. Nicholas Church | 4.3 |
| 5 | [50.4657, 30.5159] | Attractions | Sofiivska Square | 4.6 |

Для додавання нових локацій для іншим міст чи районів чи роботу через інші зовнішні джерела, інформацію з бази даних можна оновлювати для забезпечення її актуальності

### 2.9 Опис структури таблиць бази даних

База даних використовується для збереження таблиць з інформацією про приклади вершин графів які можуть бути завантажені та розглянуті, зв’язок між парами вершин з типом транспортного засобу, та координати та описи маркерів для зображення на карті. При відсутності доступу до бази даних, інформація з цих таблиць стає недосяжною, але систему можна використовувати надалі шляхом ручного створення вершин графів або їх імпорту з файлу формату .json.

Таблиця example містить в собі дані про вершини графів з наступною структурою:

- id (SERIAL) – унікальний ідентифікатор вузла, первинний ключ;
- name (VARCHAR(50)) – назва вузла;
- x (DOUBLE PRECISION) – координата по осі X;
- y (DOUBLE PRECISION) – координата по осі Y.

Таблиця paths описує шляхи між графами для відповідних прикладів за наступною структурою:

- id (SERIAL): Унікальний ідентифікатор шляху, первинний ключ;
- from (INTEGER): Ідентифікатор початкового вузла (посилання на Example1.id);
- to (INTEGER): Ідентифікатор кінцевого вузла (посилання на Example1.id);
- type (VARCHAR(50)): Тип шляху (наприклад, "дорога", "пішохідна доріжка").

Таблиця places описує локації для зображення на маркерах за наступною структурою:

- id (SERIAL) – унікальний ідентифікатор, первинний ключ;
- coordinates (TEXT) – координати місця у форматі тексту;
- category (VARCHAR(50)) – категорія місця – готель, ресторан, туристична місцина, інше;
- title (VARCHAR(255)) – назва місця;
- rating (NUMERIC) – рейтинг місця.

### 2.10 Специфікація модулей системи

Серверна частина системи забезпечує обробку HTTP-запитів та надає доступ до бази даних шляхом підключення до неї та виконання відповідних запитів API з вказаним ідентифікатором прикладу, дані якого необхідно отримати. Після отримання інформації про вершини графів з бази даних чи іншого джерела, викликається зовнішнє API для отримання актуальної інформації щодо значення висоти у відповідних координатах.

Клієнтська частина пропонує Web-сторінку з можливістю огляду карти, додавання нових вершин графів, їх редагування, створення зв’язку між ними, та обрання точок для маршрутизації, засобу пересуванню, та алгоритму маршрутизації. По завершенню пошуку маршруту, виводиться шлях, його дистанція, та час, витрачений на його прорахунок. Для реалізації сторінки було використано бібліотеку Leaflet для отримання актуальних карт та Data-Driven Documents для можливості отримання даних про вершини графів та їх зображення. Додатково існує можливість огляду місць на карті за фільтрами.

Маршрутизація проводиться після виклику з Web-сторінки, під час якої отримується інформація про обраний алгоритм, засіб пересування, та обрані точки. З серверної сторони отримується інформація про координати графа та проводиться маршрутизація по обраним вершинам, враховуючи обмеження на їх досяжність транспортним засобом. В залежності від обраного алгоритму маршрутизації, різні рішення використовуватимуть різні функції для обчислення даних. Після проведення маршрутизації, на клієнтську частину повертається інформація про створений маршрут, його дистанцію та час, який було витрачено на прорахунок.

У випадку обрання з клієнтської сторони локацій для огляду, система додасть бажані маркери з їх інформацією з бази даних на карту.

### 2.11 Специфікація функцій застосунку

Для забезпечення роботи системи було реалізовано низку функцій, специфікацію яких наведено у таблиці 2.5.

Таблиця. 2.5 – Специфікації функцій застосунку

| Функція | Опис |
|---|---|
| dragNode | Забезпечує зміни на мапі відповідно до переміщення точок графу. |
| redrawNodes | Зображає вершини графів на карті після змін. |
| redrawLines | Зображає шлях між точками на карті після змін. |
| calculateDistancesbetweennodes | Надає відстань між пов’язаними точками. |
| Getshortestroute | Надає найкоротший маршрут використовуючи метод розв’язку, представлений у пункті 1.5. |
| getSelectedTravelType | Отримує тип транспортного засобу який використовується для врахування по яким маршрутам вести маршрутизацію. |
| enrichNodesWithElevation | Назначає імпортованим вершинам значення висоти через використання API для врахування під час маршрутизації. |
| populateMarkers | Відзначає чи прибирає на мапі локації, відфільтровані за вибором користувача |

Кінець таблиці 2.5.

| Функція | Опис |
|---|---|
| Dijkstra | Проводить знаходження оптимального маршруту між точками через метод Дейкстри. |
| aStar | Проводить знаходження оптимального маршруту між точками через метод A*. |

Реалізовані функції забезпечують відображення на інтерфейсі користувача карти місцевості з інтерактивною можливістю змінювати її елементи, маркери цікавих місць, обраних користувачем, вершини графів із заданими координатами, шляхи між вершинами графів, які оновлюються відповідно до змін, деталі про маршрут, включаючи тип транспортного засобу та результат обчисленого найкоротшого шляху.

Ці функції забезпечують інтерактивність застосунку та дозволяють користувачеві не лише отримувати інформацію про маршрут, а й активно впливати на його параметри.

Реалізація мікросервісної архітектури дозволяє забезпечити використання актуальної інформації під час маршрутизації алгоритмом A* з урахуванням в обмеженні по руху через пари вершин графу та поєднувати декілька бажаних вершин графів через які потрібно знайти шлях на додачу з однією фінальною вершиною, до якої вестиметься прорахунок після знаходження маршруту.

### 2.12 Керівництво користувача

Для використання системи на одному пристрої, база даних запускається як окремий Docker контейнер для симуляції розгортки на окремому середовищі. Запуск контейнеру може відбуватися як і через

відповідну програму на пристрої, або через скрипт для запуску файлу docker-compose.yml – “docker-compose up –build”.

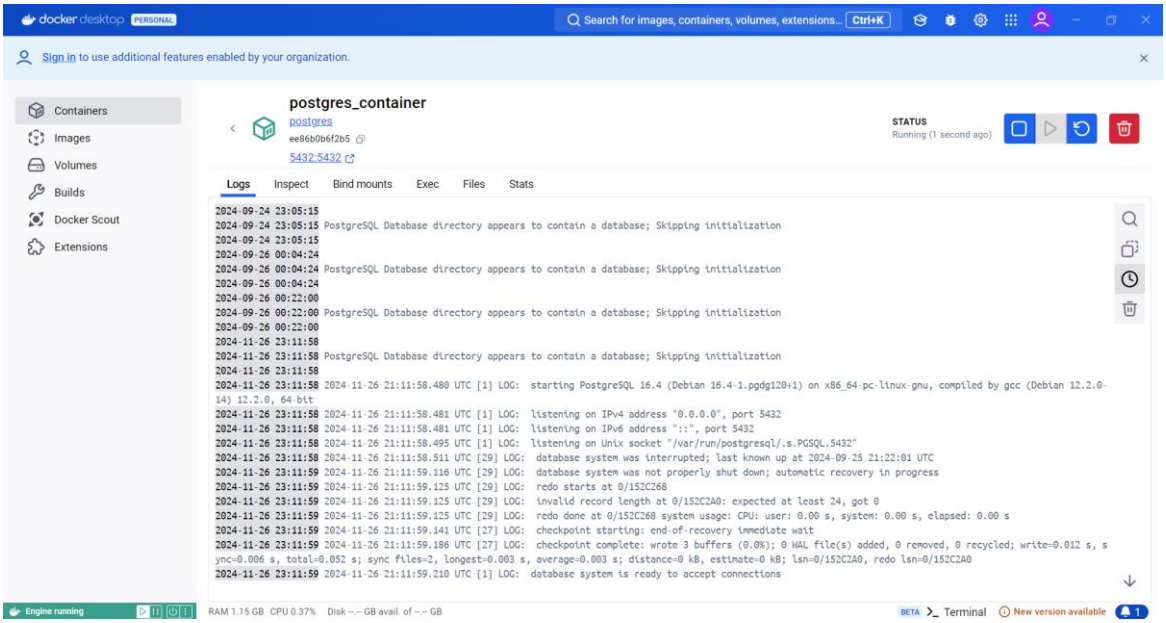


Рисунок 2.6 – Запуск Docker-контейнеру з PostgreSQL

Завдяки Node.js, розгортання самого застосунку на сервері чи на порті виконується завдяки запуску окремого скрипта app.js. Після запуску контейнера, система стає доступною за адресою http://localhost:3000, коли база даних досяжна через порт 5432.

У випадку відсутності необхідності в розгортанні додатку на сервері з базою даних та потреби запуску його локально, спершу для запуску додатку потрібно відкрити сторінку системи через файл index.html використовуючи або файлову систему, або середовище чи редактор для програмування. Такий підхід дозволяє не тільки запустити застосунок, а й проводити спостереження за його роботою, здійснювати аналіз функцій програми та відслідковувати роботу коду під час виконання.

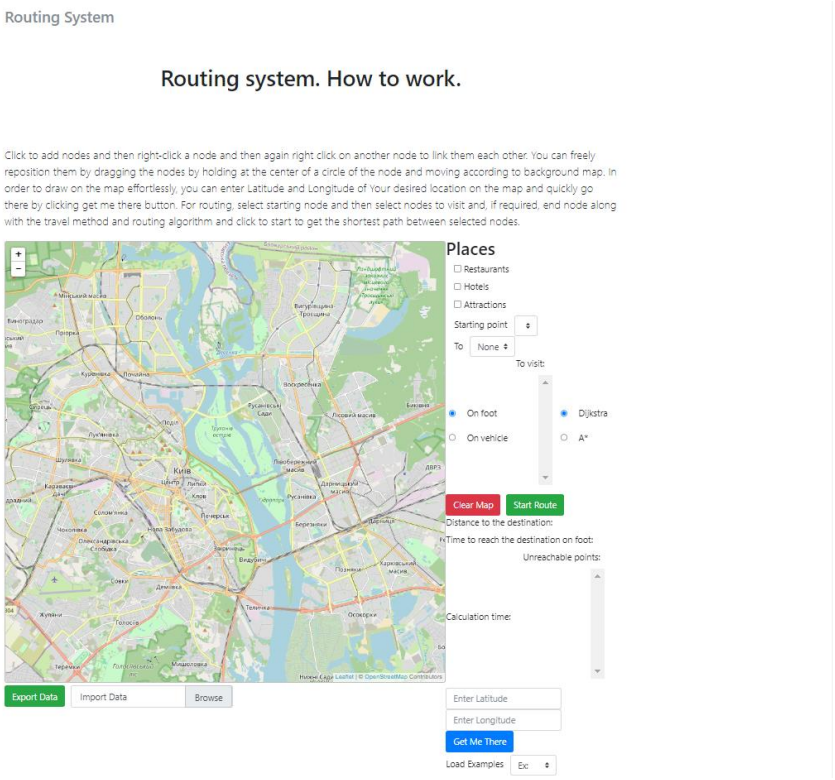


Рисунок 2.7 – Головна сторінка системи

На початку система представлена користувачеві разом з інструкціями користування та мапою, яка не містить доданих графів. Користувач може відфільтрувати типи місцевостей, які його цікавлять, з доступного переліку. Після вибору на карту додаються відповідні мітки, при наведенні на які відображаються їх назва та рейтинг у системі. Якщо користувач прибирає певний тип місцевостей зі списку обраних, відповідні маркери зникають із карти.

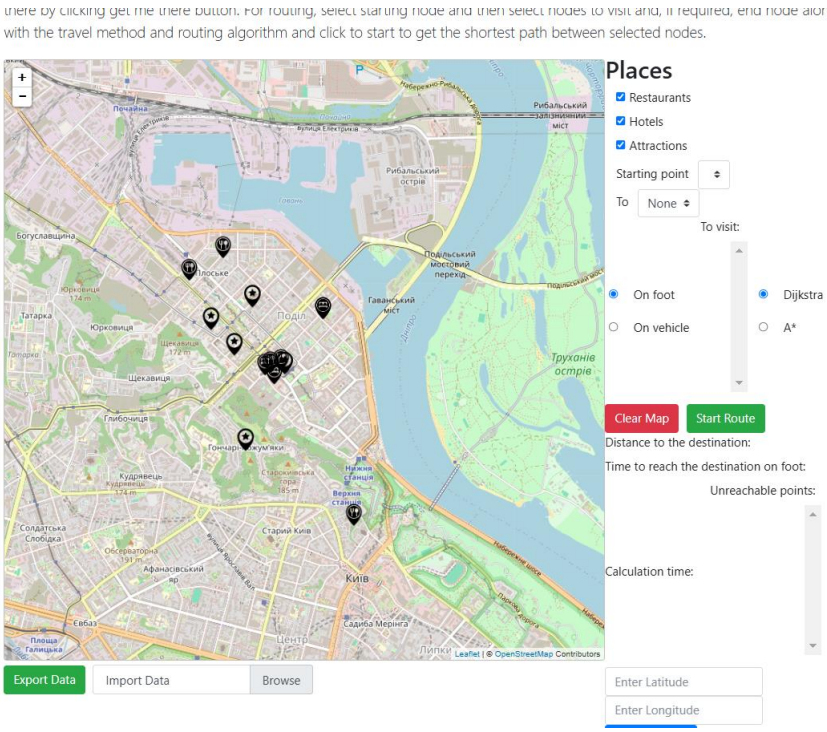


Рисунок 2.8 – Додані маркери місцевостей на мапі

Користувач також має можливість завантажити файл .json із набором вершин графів та шляхів між ними у двох масивах, вибрати приклад із бази даних або створити власні точки для навігації. Для цього достатньо натиснути ліву кнопку миші на бажаному місці на мапі, щоб додати вершину, а правою кнопкою миші — створити зв’язок між парою вершин. У результаті на карті з’являється з’єднання між обраними точками, що дозволяє будувати маршрути. Додані вершини автоматично потрапляють до списків початкових, кінцевих та проміжних місць, які користувач може обрати для побудови маршруту. Тип маршруту визначається залежно від обраного транспортного засобу. При цьому повторне натискання правої кнопки миші між двома вершинами дозволяє змінити тип маршруту.

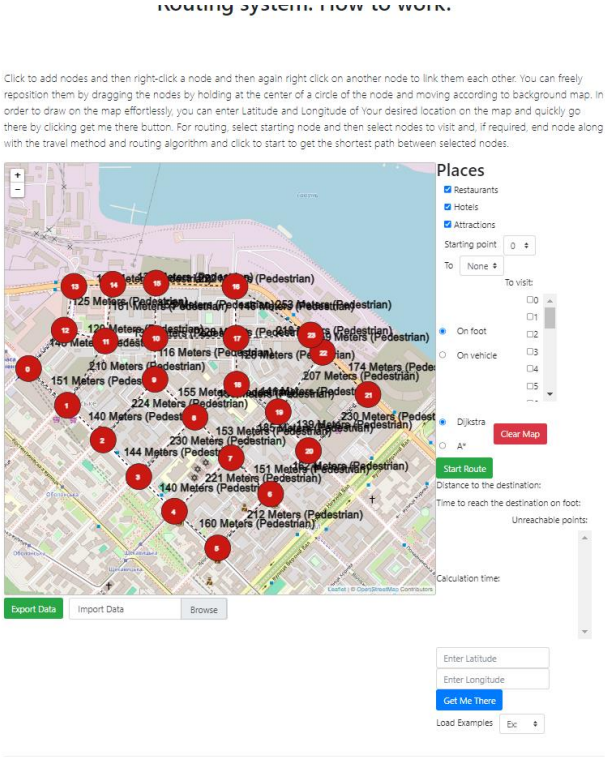


Рисунок 2.9 – Сторінка системи після введення прикладу

Після додавання всіх необхідних точок, користувач може вибрати алгоритм маршрутизації, вказати стартову вершину графу, проміжні точки, через які має проходити маршрут, та, за бажанням, кінцеву точку. Якщо кінцева вершина графа не вказана, маршрут автоматично завершується на останній пройденій вершині графу.

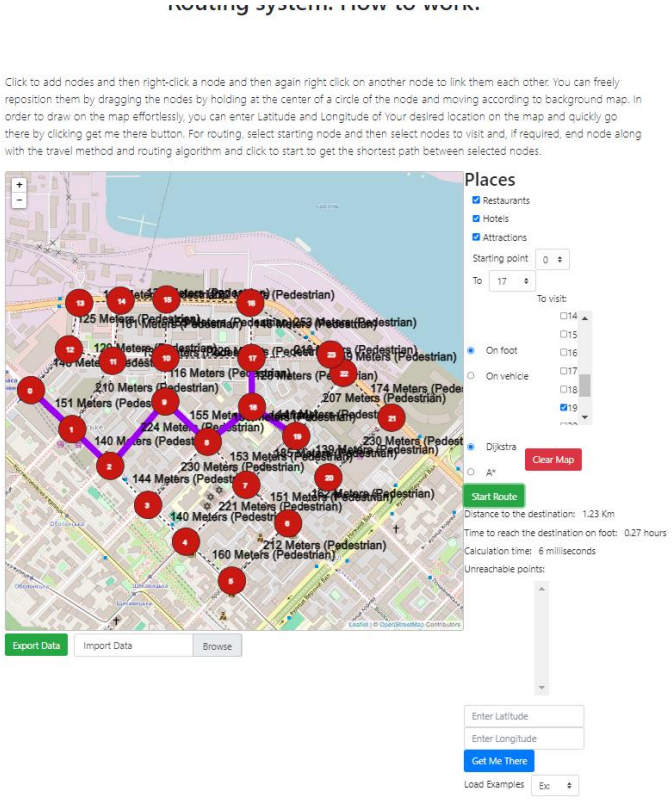


Рисунок 2.10 – Результат прокладання шляху від стартової точки до кінцевої через введені проміжні

Коли система запускає обчислення маршруту, вона опрацьовує введену інформацію за допомогою обраного алгоритму, будує шлях через обрані вершини графу та відображає його на карті. Наприкінці користувач отримує бажаний маршрут, який проходить через вершини графа, та інформацією про витрачений час на його обчислення.

Якщо користувач потребує побудувати новий маршрут, він може задати нові початкові та кінцеві точки разом з проміжними, запустити обчислення заново, та отримати оновлений результат. Для скидання стану систему до початкового можна скористатися кнопкою “Clear Map”, яка очищає карту. За необхідності користувач може ввести координати, щоб перенести карту місцевості до бажаної локації.

## 2.13 Висновок до другого розділу

У розділі було проаналізовано завдання, визначено вимоги до системи та інструменти, необхідні для її реалізації. Обґрунтовано вибір мови

програмування JavaScript, для забезпечення зручної взаємодії між серверною та клієнтською частинами системи. Також було досліджено бібліотеки та фреймворки, які дозволили реалізувати функціонал проєкту, та обґрунтоване їх використання.

Було розглянуто роботу з наявними даними, їх створення, додавання до системи, і збереження як прикладів для подальщого використання. Було розглянуто введення даних про бажаний шлях користувачем, експортування у форматі .json для подальшого використання, та імплементації Open-Elevation для збагачення вершин шрафів інформацією про висоту координат для використання при маршрутизації

З використанням Leaflet і Data-Driven Documents було створено інтерактивну карту місцевості з можливістю додавання вершин графів. Використання відкритої платформи OpenStreetMap забезпечило отримання актуальних картографічних даних. База даних PostgreSQL дозволила додати приклади вершин графів і маршрутів для майбутнього використання, а фреймворк Bootstrap забезпечив створення інтерфейсу для взаємодії із системою та отримання даних.

У розділі описано процес передачі інформації від користувача до системи, введені вручну чи імпортовані з файлу .json, а також інтеграцію сторонніх бібліотек і бази даних. Для пояснення роботи системи та складових було побудовано UML-діаграми та структурні схеми, які дозволяють детально дослідити процеси, що відбуваються в системі, описати функціонал і методику обробки даних на кожному етапі.

# 3. АНАЛІЗ РЕЗУЛЬТАТІВ ЗНАХОДЖЕННЯ ОПТИМАЛЬНОГО МАРШРУТУ

## 3.1 Метод аналізу

Для аналізу ефективності маршрутизації та порівняння реалізованих алгоритмів Дейкстри та A*, було створено ряд різних за розміром прикладів які включають в себе вершини графів з різними типами зв’язків. Для кожного транспортного засобу та алгоритму будуть задані ряд вершин які мають бути пройдені, для яких маршрутизація кожним методом буде проведена тричі для визначення середнього часу необхідного системі для проведення шляху між заданими вершинами. Після отримання тестових даних, вони будуть порівняні між собою для визначення найбільш ефективного методу маршрутизації.

За рахунок проходження алгоритмом Дейкстри кожного можливого шляху між вершинами графів та його гарантоване знаходження мінімального шляху до кожного з них, результати проведення пошуку маршруту цим алгоритмом є опорним для порівняння з іншими алгоритмами пошуку маршрутів.

## 3.2 Сценарії аналізу

Для аналізу маршрутизації було обрано три сценарії в різних умовах маршрутизації, два з яких які складаються з комплексних ділянок міста з різними типами зв’язків між вершинами графів і один з яких складається зі спрощеної системи зв’язків з метою перевірки ефективності методів маршрутизації при малих розмірах системи графів.

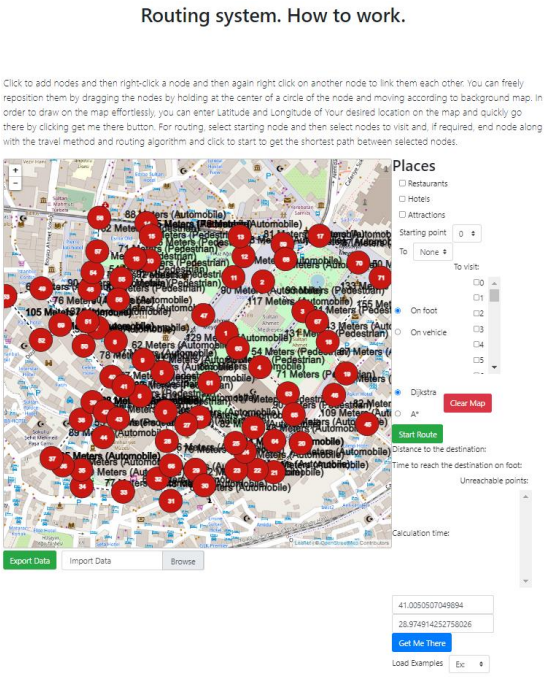


Рисунок 3.1 – Перший сценарій маршрутизації для аналізу

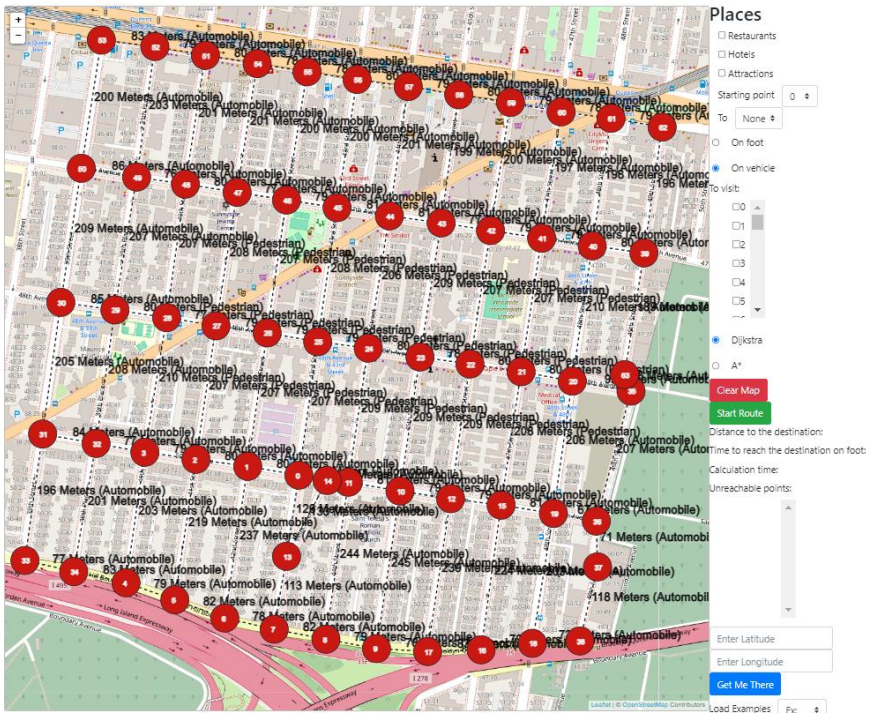


Рисунок 3.2 –Другий сценарій маршрутизації для аналізу

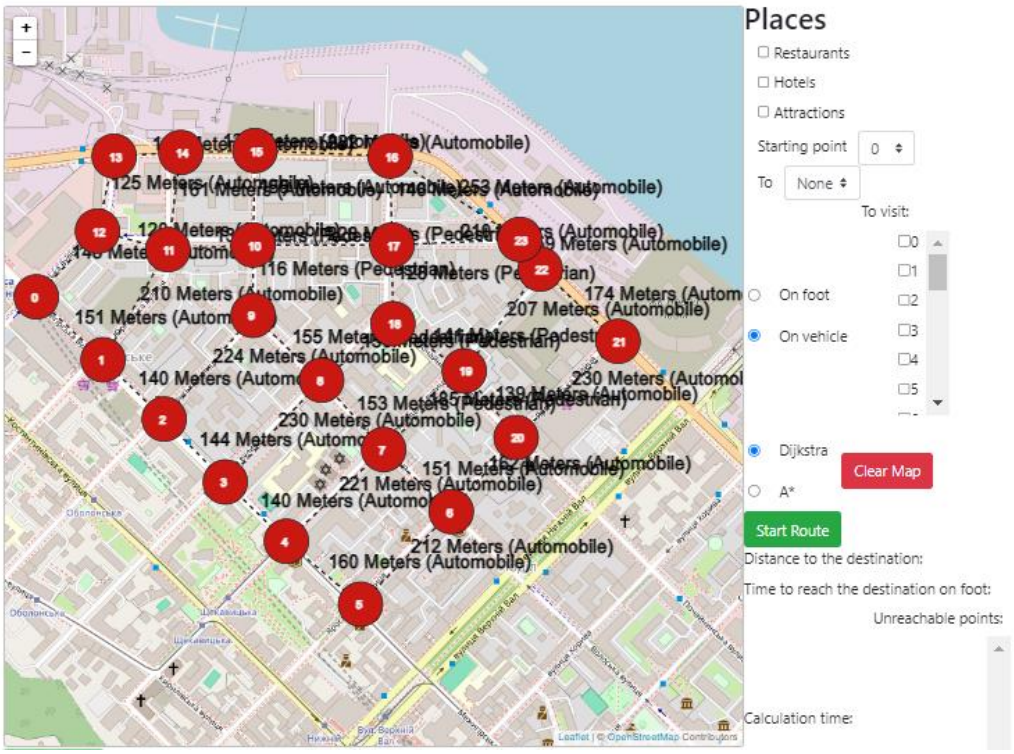


Рисунок 3.3 –Третій сценарій маршрутизації для аналізу

### 3.2.1 Аналіз ефективності алгоритмів маршрутизації при пішохідному засобу пересуванню

#### 3.2.1.1 Перший випадок

Для першого випадку було задано початкову вершину графу як 47 і наступні проміжні точки: 8, 13, 21, 27, 40, 45, 53, 68.

Таблиця. 3.1 – Результати та час маршрутизації для алгоритму Дейкстри при пішохідному засобу пересуванню в першому сценарії

| № | Отримана відстань, км | Час розрахунку маршруту, мс |
|---|---|---|
| 1 | 2.31 | 41 |
| 2 | 2.31 | 40 |
| 3 | 2.31 | 45 |

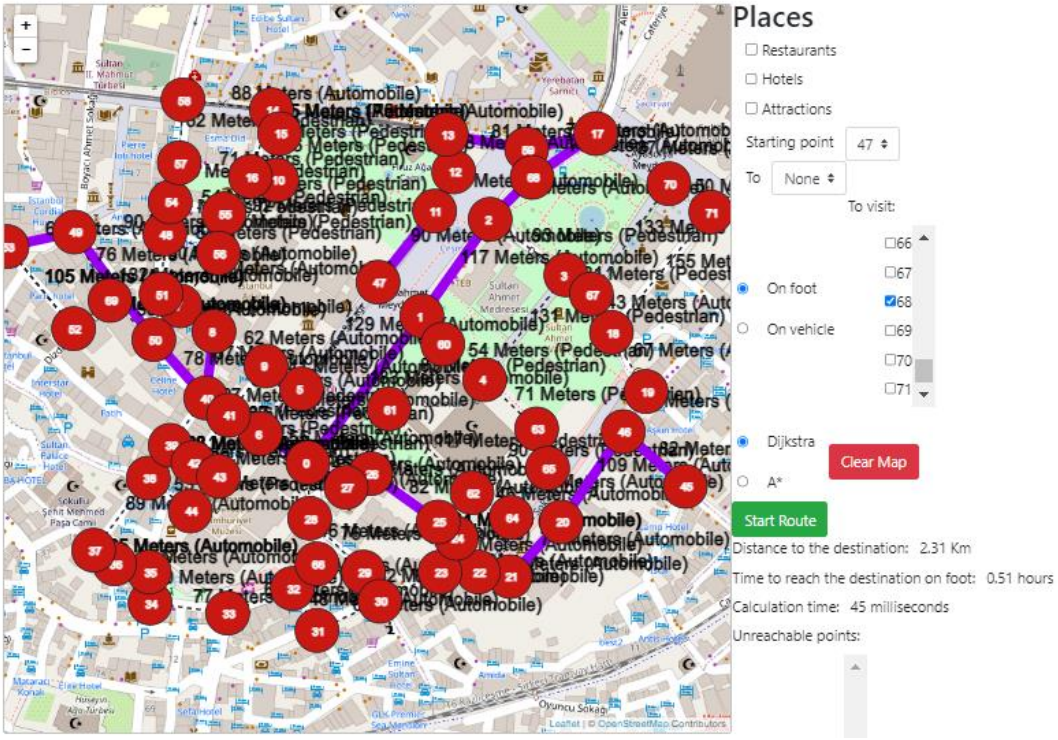


Рисунок 3.4 – Результат маршрутизації алгоритмом Дейкстри при пішохідному засобу пересування в першому випадку

Таблиця. 3.2 – Результати та час маршрутизації для алгоритму A* при пішохідному засобу пересуванню в першому сценарії

| № | Отримана відстань, км | Час розрахунку маршруту, мс |
|---|---|---|
| 1 | 2.33 | 259 |
| 2 | 2.33 | 285 |
| 3 | 2.33 | 244 |

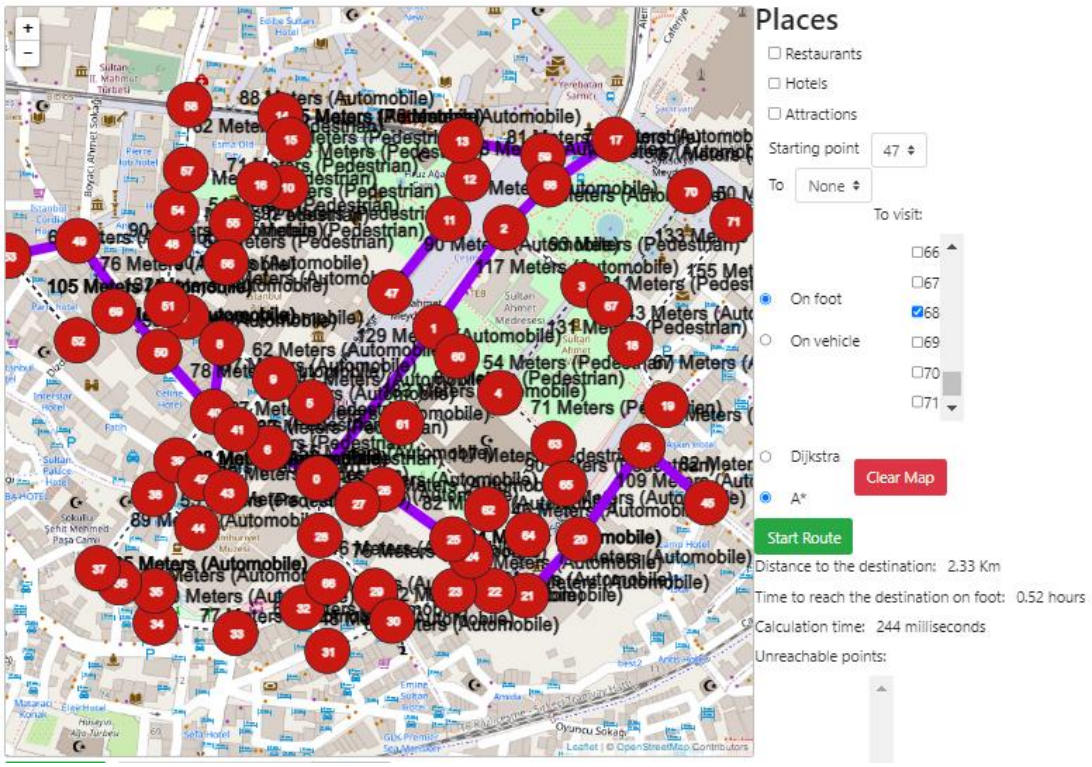


Рисунок 3.5 – Результат маршрутизації алгоритмом A* при пішохідному засобу пересування в першому випадку

### 3.2.1.2 Другий випадок

Для першого випадку було задано початкову вершину графу як 55, кінцеву вершину як 38 і наступні проміжні точки: 3, 7, 11, 20, 44.

Таблиця. 3.7 – Результати та час маршрутизації для алгоритму Дейкстри при пішохідному засобу пересуванню в другому сценарії

| № | Отримана відстань, км | Час розрахунку маршруту, мс |
|---|---|---|
| 1 | 3 | 23 |
| 2 | 3 | 76 |
| 3 | 3 | 28 |

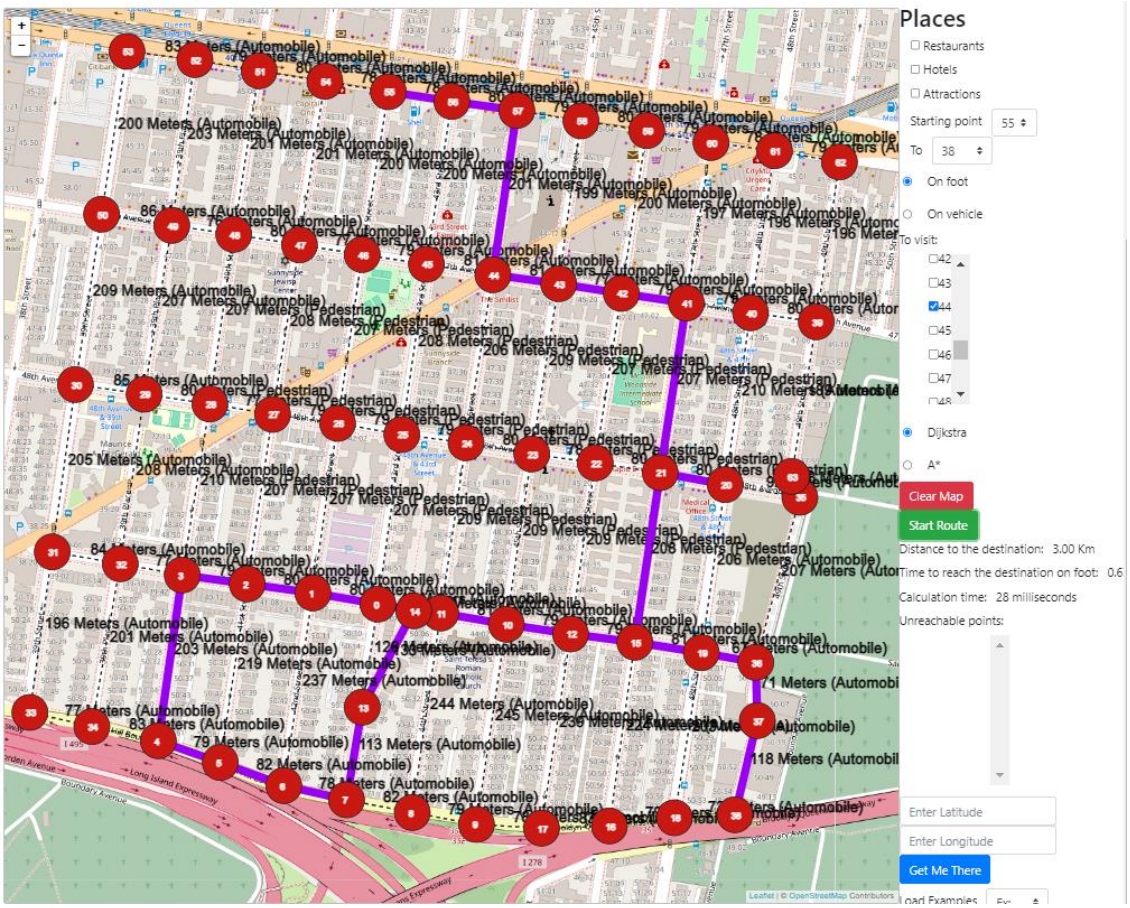


Рисунок 3.6 – Результат маршрутизації алгоритмом Дейкстри при пішохідному засобу пересування в другому випадку

Таблиця. 3.8 – Результати та час маршрутизації для алгоритму A* при пішохідному засобу пересуванню в другому сценарії

| № | Отримана відстань, км | Час розрахунку маршруту, мс |
|---|---|---|
| 1 | 3.02 | 290 |
| 2 | 3.02 | 175 |
| 3 | 3.02 | 224 |

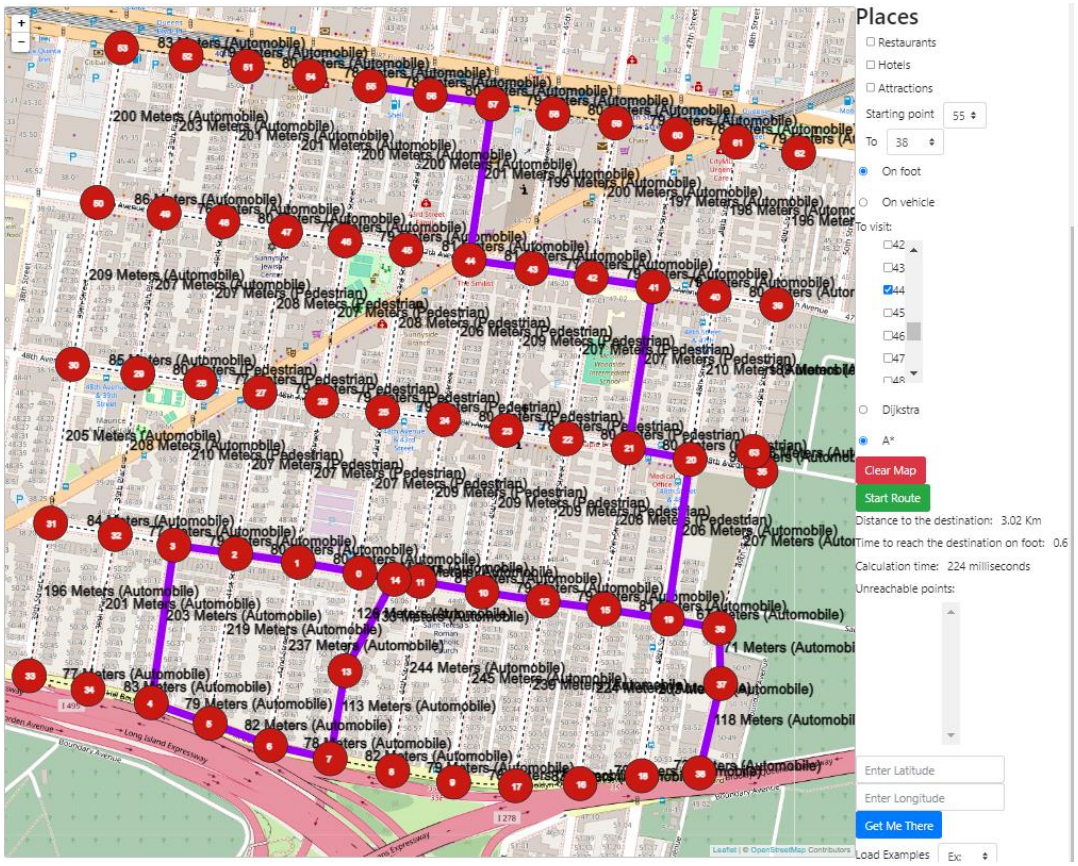


Рисунок 3.7 – Результат маршрутизації алгоритмом A* при пішохідному засобу пересування в другому випадку

### 3.2.1.3 Третій випадок

Для третього випадку було задано початкову вершину графу як 3 і наступні проміжні точки: 0, 8, 17, 10.

Таблиця. 3.9 – Результати та час маршрутизації для алгоритму Дейкстри при пішохідному засобу пересуванню в третьому сценарії

| № | Отримана відстань, км | Час розрахунку маршруту, мс |
|---|---|---|
| 1 | 1.36 | 10 |
| 2 | 1.36 | 5 |
| 3 | 1.36 | 7 |

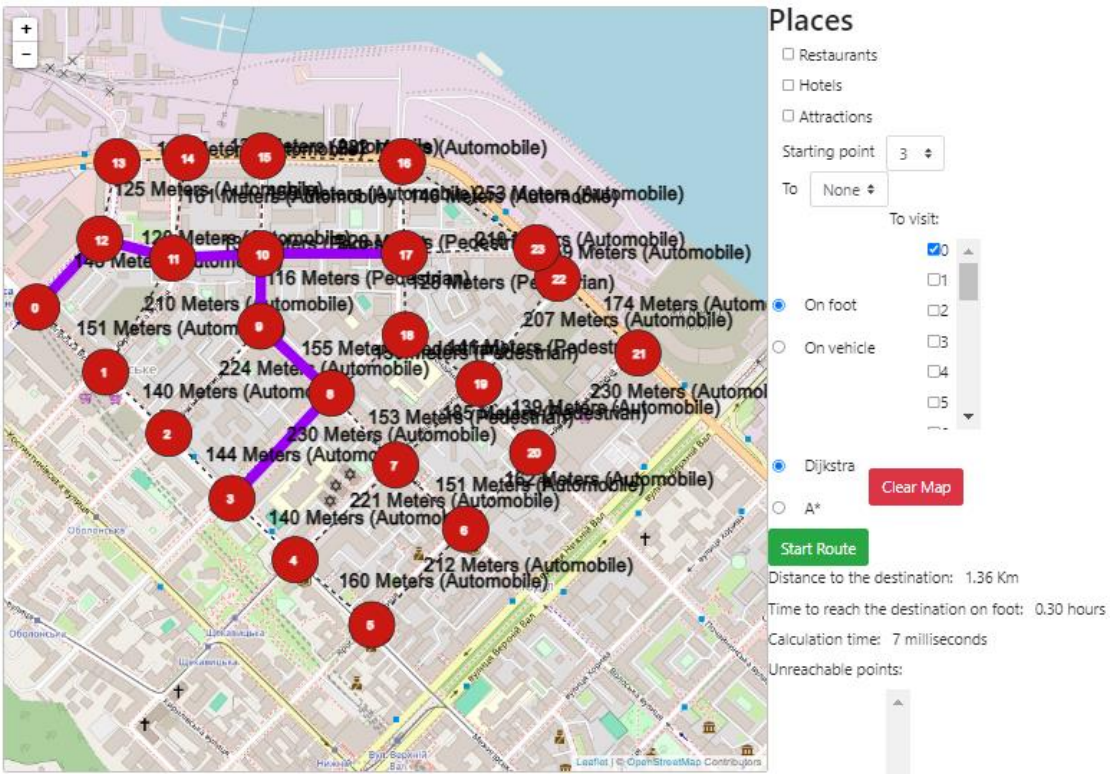


Рисунок 3.8 – Результат маршрутизації алгоритмом Дейкстри при пішохідному засобу пересування в третьому випадку

Таблиця. 3.10 – Результати та час маршрутизації для алгоритму A* при пішохідному засобу пересуванню в третьому сценарії

| № | Отримана відстань, км | Час розрахунку маршруту, мс |
|---|---|---|
| 1 | 1.36 | 31 |
| 2 | 1.36 | 31 |
| 3 | 1.36 | 27 |

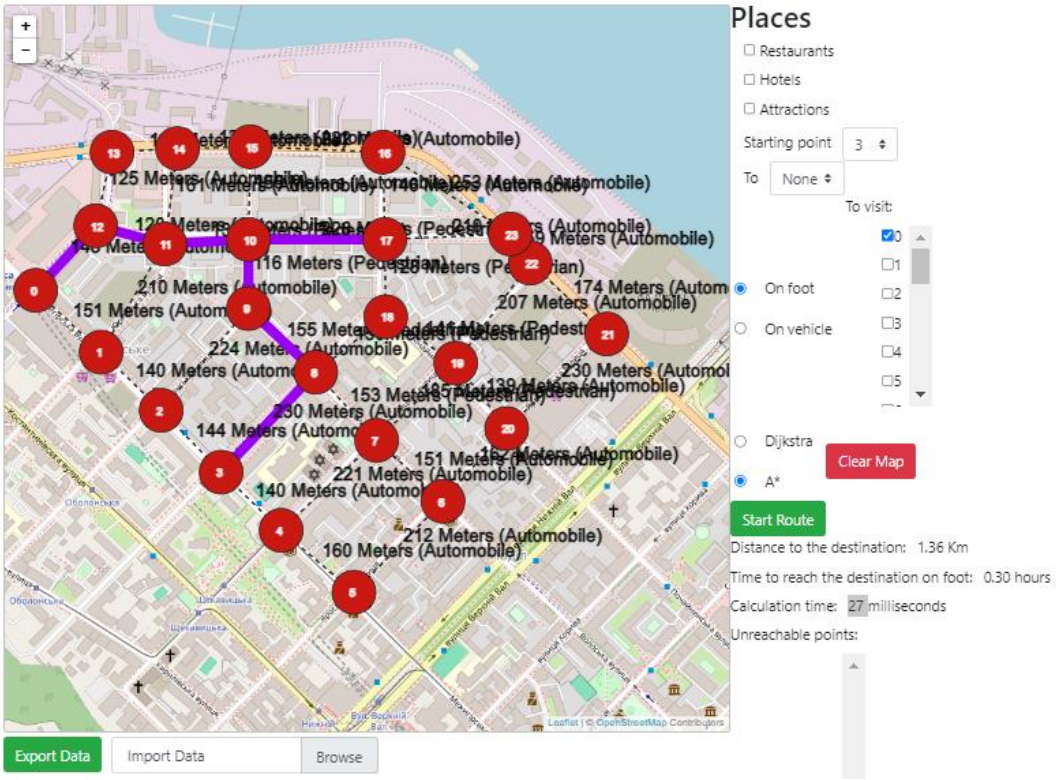


Рисунок 3.9 – Результат маршрутизації алгоритмом A* при пішохідному засобу пересування в третьому випадку

### 3.2.2 Підсумок даних при пересуванні пішки

Під час проведення аналізу було виявлено, що алгоритму Дейкстри необхідна менша кількість часу для знаходження оптимального маршруту ніж для методу A*. Водночас, при другому сценарії, алгоритм A* прорахував інший, трохи довший маршрут, що могло бути результатом вплину евристики та значення висоти на координатах вершин графів.

### 3.2.3 Аналіз ефективності алгоритмів маршрутизації при пересуванні автомобілем

#### 3.2.3.1 Перший випадок

Для першого випадку було задано початкову вершину графу як 47 і наступні проміжні точки: 8, 13, 21, 27, 40, 45, 53, 68.

Таблиця. 3.11 – Результати та час маршрутизації для алгоритму Дейкстри при пересування автомобілем в першому сценарії

| № | Отримана відстань, км | Час розрахунку маршруту, мс |
|---|---|---|
| 1 | 2.40 | 57 |
| 2 | 2.40 | 41 |
| 3 | 2.40 | 39 |

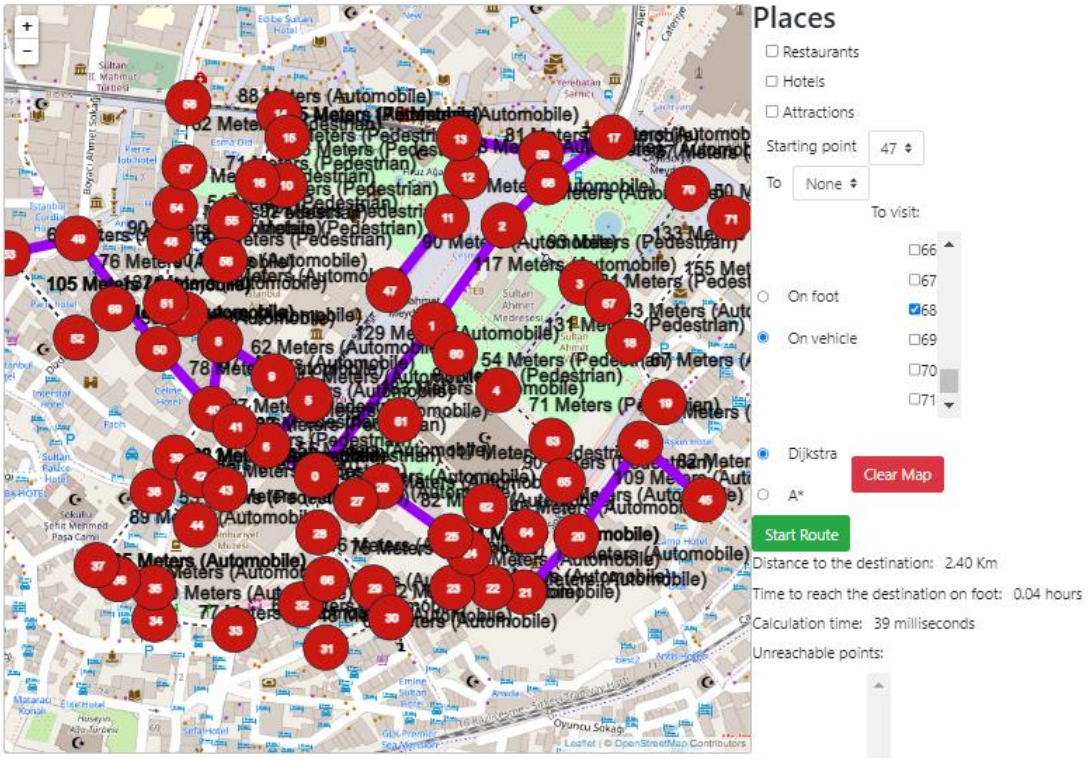


Рисунок 3.10 – Результат маршрутизації алгоритмом Дейкстри при пересуванні автомобілем в першому випадку

Таблиця. 3.12 – Результати та час маршрутизації для алгоритму A* при пересування автомобілем в першому сценарії

| № | Отримана відстань, км | Час розрахунку маршруту, мс |
|---|---|---|
| 1 | 2.42 | 173 |
| 2 | 2.42 | 209 |
| 3 | 2.42 | 174 |

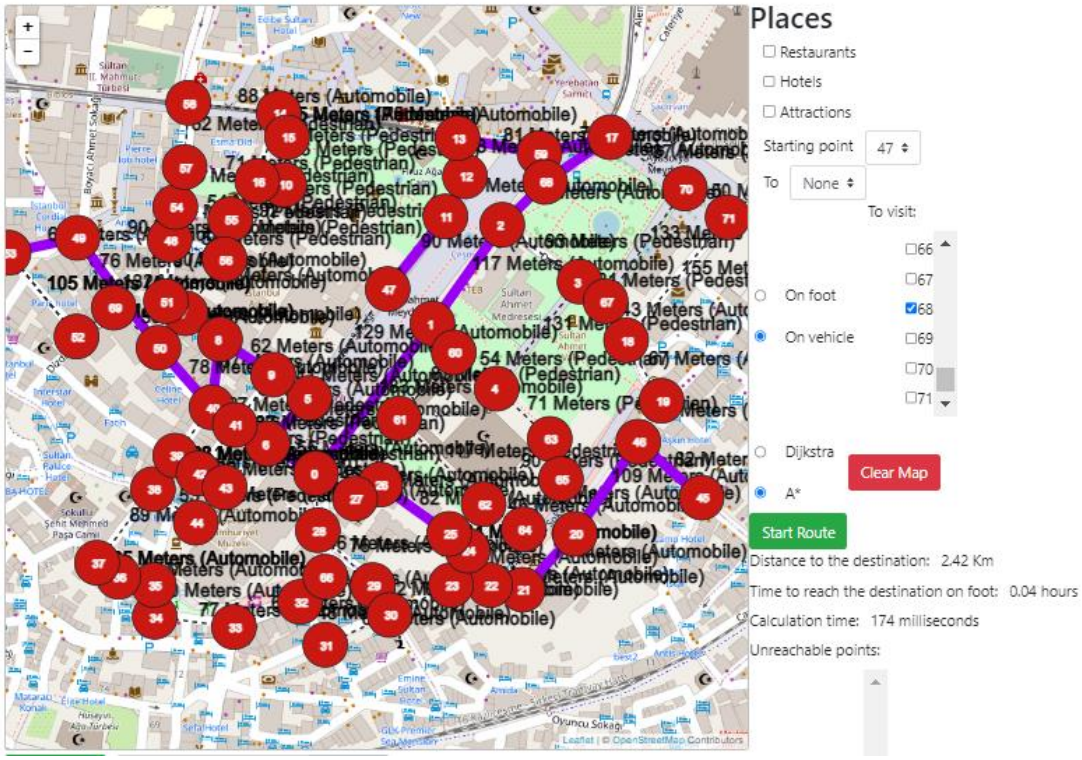


Рисунок 3.11 – Результат маршрутизації алгоритмом A* при пересуванні автомобілем в першому випадку

### 3.2.3.2 Другий випадок

Для першого випадку було задано початкову вершину графу як 55, кінцеву вершину як 38 і наступні проміжні точки: 3, 7, 11, 20, 44.

Таблиця. 3.13 – Результати та час маршрутизації для алгоритму Дейкстри при пересування автомобілем в другому сценарії

| № | Отримана відстань, км | Час розрахунку маршруту, мс |
|---|---|---|
| 1 | 3.01 | 24 |
| 2 | 3.01 | 30 |
| 3 | 3.01 | 28 |

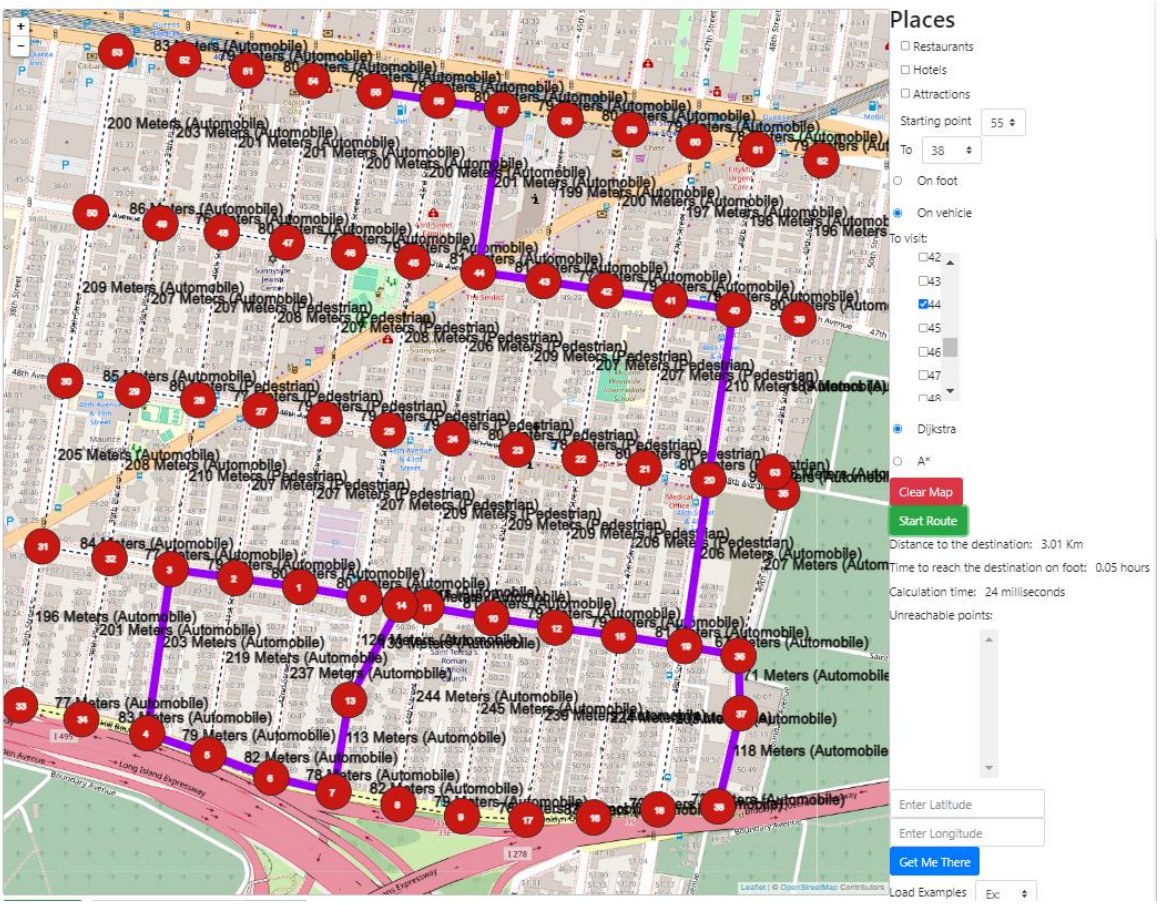


Рисунок 3.12 – Результат маршрутизації алгоритмом Дейкстри при пересуванні автомобілем в другому випадку

Таблиця. 3.14 – Результати та час маршрутизації для алгоритму A* при пересування автомобілем в першому сценарії

| № | Отримана відстань, км | Час розрахунку маршруту, мс |
|---|---|---|
| 1 | 3.02 | 168 |
| 2 | 3.02 | 141 |
| 3 | 3.02 | 140 |

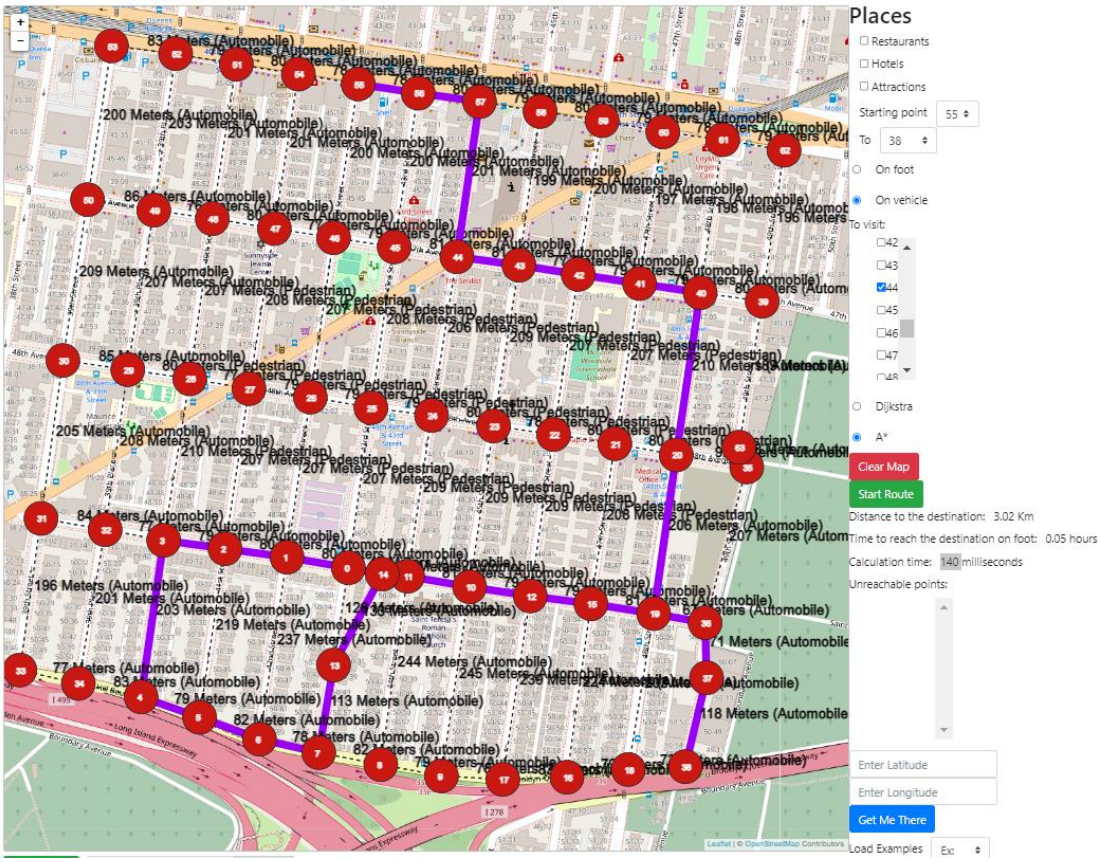


Рисунок 3.13 – Результат маршрутизації алгоритмом A* при пересуванні автомобілем в другому випадку

### 3.2.3.3 Третій випадок

Для третього випадку було задано початкову вершину графу як 3 і наступні проміжні точки: 0, 8, 17, 10.

Таблиця. 3.15 – Результати та час маршрутизації для алгоритму Дейкстри при пересування автомобілем в третьому сценарії

| № | Отримана відстань, км | Час розрахунку маршруту, мс |
|---|---|---|
| 1 | 2.07 | 8 |
| 2 | 2.07 | 8 |
| 3 | 2.07 | 7 |

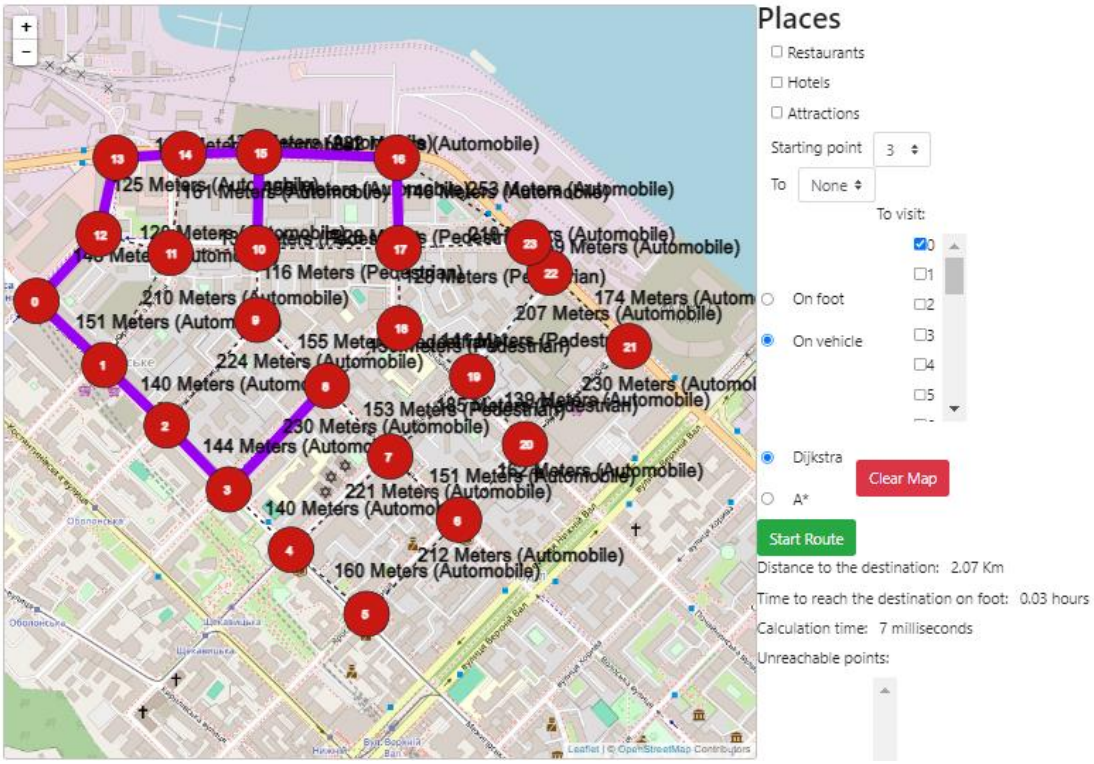


Рисунок 3.14 – Результат маршрутизації алгоритмом Дейкстри при пересуванні автомобілем в третьому випадку

Таблиця. 3.16 – Результати та час маршрутизації для алгоритму A* при пересування автомобілем в третьому сценарії

| № | Отримана відстань, км | Час розрахунку маршруту, мс |
|---|---|---|
| 1 | 2.08 | 27 |
| 2 | 2.08 | 26 |
| 3 | 2.08 | 39 |

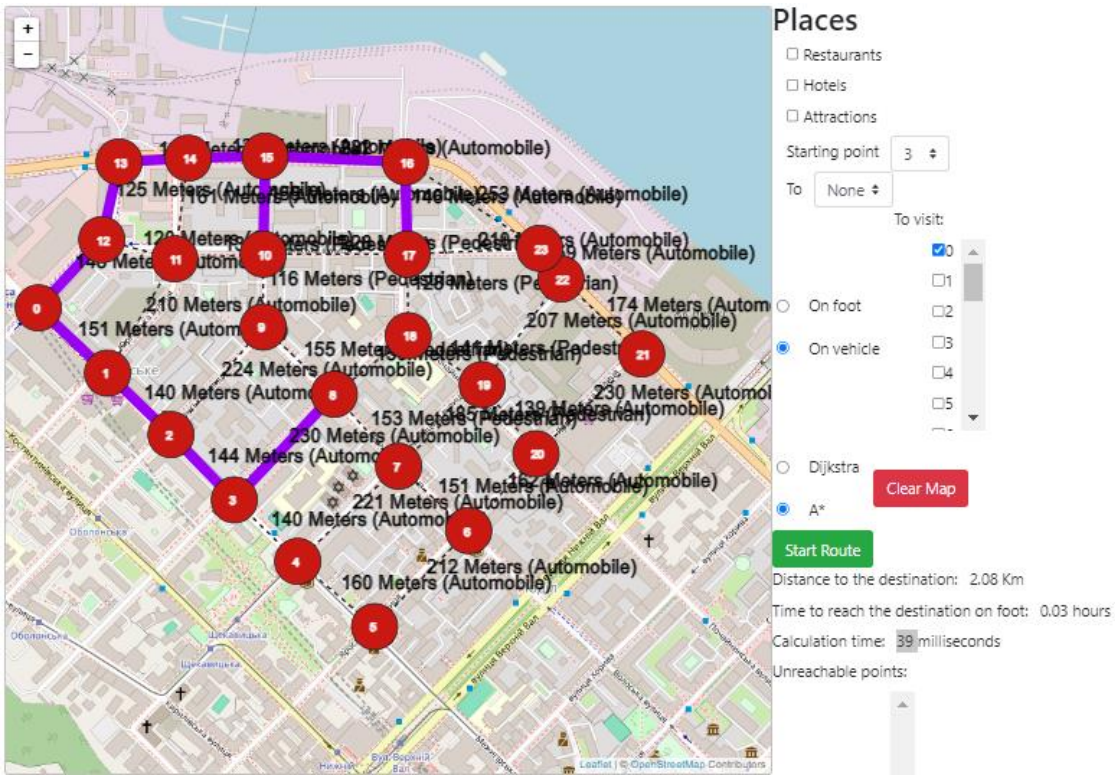


Рисунок 3.15 – Результат маршрутизації алгоритмом A* при пересуванні автомобілем в третьому випадку

### 3.3 Підсумок даних при пересуванні автомобілем

Аналогічно із проведенням аналізу для випадків маршрутизації з використанням пішохідного методу пересування, алгоритму Дейкстри знадобилася значно менше часу для знаходження оптимального маршруту ніж для методу A*. Різниці між маршрутами не спостерігається.

### 3.4 Висновок до третього розділу

В розділі було проведено аналіз ефективності алгоритмів маршрутизації через порівняння результатів прорахунку шляху та час, витрачений на це у ряді різних прикладів.

В таблиці 6.13 наведено результати та середній час маршрутизації кожного сценарію.

Таблиця. 3.17 – Результати та середній час маршрутизації кожного сценарію

| Алгоритм | Сценарій | Тип транспорту | Отримана відстань, км | Середній час розрахунку маршруту, мс |
|---|---|---|---|---|
| Дейкстри | 1 | Пішки | 2.31 | 42 |
| A* | 1 | Пішки | 2.33 | 262,66 |
| Дейкстри | 2 | Пішки | 3 | 42,33 |
| A* | 2 | Пішки | 3.02 | 229,66 |
| Дейкстри | 3 | Пішки | 1.36 | 7,33 |
| A* | 3 | Пішки | 1.36 | 30 |
| Дейкстри | 1 | Автомобіль | 2.40 | 45,66 |
| A* | 1 | Автомобіль | 2.42 | 185,33 |
| Дейкстри | 2 | Автомобіль | 3.01 | 27,33 |
| A* | 2 | Автомобіль | 3.02 | 449 |
| Дейкстри | 3 | Автомобіль | 2.07 | 7,66 |
| A* | 3 | Автомобіль | 2.08 | 30,66 |

Коли використання алгоритму Дейкстри здатне знайти гарантований короткий маршрут, метод A* разом з іншими евристичними методами маршрутизації здатний як і врахувати потреби користувача (такі як тип дороги, АЗС, тощо), чи надати оптимальний для них маршрут у великих системах при правильному використанні евристичної функції.

Під час проведення аналізу, в розділі було визначено що в створеній системі алгоритм Дейкстри надає бажаний результат витрачаючи значно меншу часу, коли метод A* витрачав на знаходження того-ж маршруту більше часу. Однак, за рахунок використання іншого методу прорахунку маршруту, враховуються і інші змінні, такі як висота між вершинами графів, внаслідок

чого результат може відрізнятися від отриманих даних від аналогічного сценарію для алгоритму Дейкстри.

Таким чином ми можемо визначити що кожний алгоритм варто використовувати в залежності від різних потреб та умов використання, а ефективність евристичних залежить від правильної імплементації допоміжної функції та від бажаного результату. Кожний метод може використовуватися паралельно або комбіновано в залежності від комплексності маршруту, дистанції, та необхідності враховувати зовнішні фактори.

# ВИСНОВКИ

Під час виконання атестаційної роботи було створено систему маршрутизації та проведено дослідження популярних алгоритмів маршрутизації.

Створена система утилізує мікросервісну архітектуру для забезпечення створення, редагування, та надання вершин графів для майбутньої маршрутизації та отримання актуальних даних щодо маршрутів для використання у маршрутизації. Для прорахунків було використано алгоритми Дейкстри через гарантоване надання короткого маршруту та A* через використання евристичної функції для обрання оптимальних вершин графів для навігації.

В результаті проведення дослідження було визначено, що алгоритм Дейкстри витрачає значно менше часу для надання маршруту аніж алгоритм A*. Це вказує, що для пришвидшення пошуку та уникнення аналізу зайвих вершин графу евристичні алгоритми вимагають ускладненої і чіткої реалізації. За рахунок використання іншого методу прорахунку маршруту, в алгоритмі A* враховуються і інші змінні, такі як висота між вершинами графів, через що отриманий маршрут може відрізнятися від аналогічної інформації від алгоритму Дейкстри.

Кожний алгоритм може використовуватися в залежності від різних потреб та умов використання. Для систем малих масштабів, алгоритму Дейкстри достатньо для визначення найкоротшого маршруту, коли у великих системах, евристичні алгоритми здатні пришвидшити пошуки та зменшити час прорахунку маршруту, враховуючи при цьому побажання користувача та інші фактори.

Результати виконання атестаційної роботи можуть використовуватися для імплементації алгоритмів маршрутизації, покращення їх утилізації, порівняння їх ефективності у розробленій системі та її подальший розвиток.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Наконечний С. І., Савіна С. С. Математичне програмування: Навч. посіб. Київ: КНЕУ, 2003. 425 с;

2. T. Agarwal. Shortest Routes vs. Fast Routes: Delivery Dilemma. Отримано з TrackoBit. URL: https://trackobit.com/blog/shortest-routes-vs-fastest-routes-delivery (дата звернення: 25.10.2024);

3. D. S. Hochbaum. Lecture Notes for IEOR 266: Graph Algorithms and Network Flows. IEOR 266, 2014. pp 37;

4. E. Dijkstra. A note on two problems in connexion with graphs. Numerische Mathematik. Numerische Mathematik 1, 1959. pp 269–271;

5. D. E. Knuth. The Art of Computer Programming: Volume 1: Fundamental Algorithms. Addison-Wesley Professional, 1997;

6. A. Patel. Amit’s A* Pages. Отримано з Red Blob Games. URL: http://theory.stanford.edu/~amitp/GameProgramming/ (дата звернення: 20.02.2023);

7. Рудик О. Визначення найкоротшого шляху у зваженому графі, алгоритми Дейкстри і Флойда-Уоршелла. Київські учнівські олімпіади з інформаційних технологій і вивчення інформатики, 2016. URL: http://www.kievoit.ippo.kubg.edu.ua/kievoit/2016/75_C++/index.html (дата звернення: 30.09.2024);

8. G. David Forney, J. The Viterbi Algorithm: A Personal History. Cambridge, MA: MIT, 2005;

9. Панасенко І.О. Дослідження алгоритмів знаходження оптимальних маршрутів у складі Web-додатку на базі мікросервісної архітектури. «Молодь: наука та інновації» 2024: матеріали XII Міжнародної науково-технічної конференції студентів, аспірантів та молодих вчених, Дніпро, 13–15 листопада 2024 року (у 3-х томах) / Національний технічний університет «Дніпровська політехніка» – Дніпро : НТУ «ДП», 2024. Том 2. С. 127-128;

10. Зеленський О. В., Дармосюк В. М., Лобач Р.В. Відновлення матриць відстаней та їх застосування. Математичне та комп'ютерне моделювання. Серія: Фізико-математичні науки. Випуск 22. Кам’янець-Подільський національний університет імені Івана Огієнка, Інститут кібернетики імені В.М. Глушкова Національної академії наук України, 2021. С. 75-80;

11. C. Rayner, N. Sturtevant, N. Sturtevant. Euclidean Heuristic Optimization. Euclidean Heuristic Optimization. Association for the Advancement of Artificial Intelligence, 2011. pp 82;

12. Leica Geosystems. Quantum GIS - еталон успіху некомерційних проєктів. Отримано з Leica Geosystems, URL: https://ngc.com.ua/ua/info/qgis.html (дата звернення: 01.10.2024);

13. A. Oleś. What is the algorithm used by ORS API for QGIS. Отримано з openrouteservice: URL: https://ask.openrouteservice.org/t/what-is-the-algorithm-used-by-ors-api-for-qgis/2476 (дата звернення: 01.10.2024);

14. Environmental Systems Research Institute, Inc. Introduction to ArcGIS. Отримано з ArcGIS Architecture Center: https://architecture.arcgis.com/en/overview/introduction-to-arcgis/introduction.html (дата звернення: 01.10.2024);

15. Environmental Systems Research Institute, Inc. (2024). Algorithms used by the ArcGIS Network Analyst extension. Отримано з ArcGIS Desktop: https://desktop.arcgis.com/en/arcmap/latest/extensions/network-analyst/algorithms-used-by-network-analyst.htm (дата звернення: 01.10.2024);

16. Google. Google Maps. Отримано з Google. URL: www.google.com/maps;

17. AllBlackBerry. Real time traffic information with Google Maps. Отримано з CrackBerry, 2007. URL: https://crackberry.com/real-time-traffic-information-google-maps (дата звернення: 27.09.2024);

18. ganga4518. The Algorithms Behind The Working Of Google Maps. Отримано з CodeChef. URL: https://blog.codechef.com/2021/08/30/the-

algorithms-behind-the-working-of-google-maps-dijkstras-and-a-star-algorithm/ (дата звернення: 27.09.2024);

19. Apple. Apple Maps. Отримано з Apple. URL: https://www.apple.com/maps/;

20. J. Dove, K. Parrish. Apple Maps vs. Google Maps: Which one is best for you? Отримано з Digital Trends, 2021. URL: https://www.digitaltrends.com/mobile/apple-maps-vs-google-maps/ (дата звернення: 27.09.2024);

21. Microsoft. Bing Maps. Отримано з Bing. URL: www.bing.com/maps/;

22. Waze Mobile. Waze. Отримано з Waze. URL: https://www.waze.com/;

23. Techstars. Routific. Отримано з Routific. URL: https://routific.com/l;

24. Routific. Route optimization API for last mile logistics & couriers. Отримано з Routific: https://dev.routific.com/ (дата звернення: 27.09.2024);

25. Roadtrippers. Roadtrippers. Отримано з Roadtrippers: https://roadtrippers.com/;

26. T. Tran. Advantages and Disadvantages of Microservices Architecture. Отримано з Orient Software. URL: https://www.orientsoftware.com/blog/advantages-and-disadvantages-of-microservices/ (дата звернення: 08.11.2024).

27. Capital One Tech (2024). 10 microservices design patterns for better architecture. Отримано з Medium. URL: https://medium.com/capital-one-tech/10-microservices-design-patterns-for-better-architecture-befa810ca44e (дата звернення: 01.12.2024).

28. P. Wayner. JavaScript compared to C, Java, C#, Python, Ruby, and PHP. Отримано з TechBeacon. URL: https://techbeacon.com/app-dev-testing/javascript-vs-other-languages-live-long-prosper (дата звернення: 01.10.2024);

29. Logos IT Academy. Мови програмування: для чого потрібні, які популярні, яку обрати і з чого розпочати вивчення? Отримано з Logos IT Academy. URL: https://lviv.logos-academy.com/movy-programuvannya-yaku-obraty-i-z-chogo-rozpochaty-vyvchennya# (дата звернення: 01.10.2024);

30. Amazon Web Services, Inc. What is PostgreSQL? Отримано з Amazon Web Services. URL: https://aws.amazon.com/rds/postgresql/what-is-postgresql/ (дата звернення: 01.10.2024);

31. FoxmindEd. Що таке PostgreSQL і для чого використовується? Отримано з FoxmindEd. URL: https://foxminded.ua/postgresql-shcho-tse/ (дата звернення: 01.10.2024);

32. J. R. Lourenço. Open-Elevation. Отримано з Open-Elevation. URL: https://open-elevation.com/ (дата звернення: 01.10.2024);

33. A. Ali. Visual Studio Code Vs. Visual Studio. Отримано з Medium. URL: https://medium.com/codex/visual-studio-code-vs-visual-studio-299cf7b16f67 (дата звернення: 09.07.2022).

# ДОДАТОК А. Код програми

## Файл app.js

```
const express = require("express");
const app = express();
const path = require('path');
const { Client } = require('pg');
const { exec } = require('child_process');

exec('npx mocha', (error, stdout, stderr) => {
  if (error) {
      console.error(`Error executing tests: ${error}`);
      return;
  }
  if (stderr) {
      console.error(`Error during execution: ${stderr}`);
  }
  console.log(`Test results:\n${stdout}`);
});

const client = new Client({
  host: 'localhost',
  port: 5432,
  user: 'postgres',
  password: '1111',
  database: 'postgres'
});

client.connect().then(() => {
  console.log('Connected to PostgreSQL');
}).catch(err => {
  console.error('Error connecting to PostgreSQL:', err.stack);
});
;

app.get('/places', async (req, res) => {
  try {
    const result = await client.query('SELECT coordinates, category, title, rating FROM places');
    res.json(result.rows);
  } catch (err) {
    console.error(err);
    res.status(500).send('Error retrieving data from database');
  }
});

// gathering nodes data
app.get('/api/nodes/:example', async (req, res) => {
  const example = req.params.example;

  try {
    const nodesTable = `Example${example}`;
    const nodesQuery = `SELECT * FROM ${nodesTable}`;

    const result = await client.query(nodesQuery);
    const nodes = result.rows.map(row => ({
      name: row.name,
      x: row.x,
      y: row.y
    }));

    res.json(nodes);
  } catch (err) {
    console.error('Error retrieving nodes:', err.stack);
    res.status(500).send('Error retrieving nodes from database');
  }
});
```

```
// gathering paths data
app.get('/api/paths/:example', async (req, res) => {
  const example = req.params.example;

  try {
    const pathsTable = `ExamplePaths${example}`;
    const pathsQuery = `SELECT * FROM ${pathsTable}`;

    const result = await client.query(pathsQuery);
    const paths = result.rows.map(row => ({
      id: row.id,
      from: row.from,
      to: row.to
    }));

    res.json(paths);
  } catch (err) {
    console.error('Error retrieving paths:', err.stack);
    res.status(500).send('Error retrieving paths from database');
  }
});


app.listen(3000, () => {
  console.log("Application started and Listening on port 3000");
});

// serve your css as static
app.use(express.static(path.join(__dirname, 'public')));

app.get("/", (req, res) => {
  var parentDir = path.join(__dirname, 'public');
  console.log("Serving file from:", parentDir);
  res.sendFile(path.join(parentDir, "index.html"));
});
```

## Файл docker-compose.yml

```
version: '3.8'

services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "3000:3000"
    depends_on:
      - postgres
    environment:
      POSTGRES_HOST: postgres
      POSTGRES_PORT: 5432
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: "1111"
      POSTGRES_DB: postgres
    volumes:
      - .:/usr/src/app
    command: node app.js
    networks:
      - bridge

  postgres:
    image: postgres:16.4
    container_name: ee86b0b6f2b5
    environment:
      POSTGRES_PASSWORD: "1111"
      PATH: "/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/lib/postgresql/16/bin"
      GOSU_VERSION: "1.17"
      LANG: "en_US.utf8"
```

```
      PG_MAJOR: "16"
      PG_VERSION: "16.4-1.pgdg120+1"
      PGDATA: "/var/lib/postgresql/data"
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - /var/lib/postgresql/data:/var/lib/postgresql/data
    ports:
      - "5432:5432"
    restart: "no"
    runtime: "runc"
    networks:
      - bridge

volumes:
  postgres_data:

networks:
  bridge:
    driver: bridge
```

## Файл Dockefile

```
FROM node:18

WORKDIR /usr/src/app

COPY package*.json ./
RUN npm install

COPY . .

EXPOSE 3000

CMD ["node", "app.js"]
```

## Файл main.js

```
var graphmap, svg, maps, g;

var mapdata = {
    allnodes: [],
    paths: [],
    distances: [],
    getui: {
        htmlSelectStartingNode: "#from-starting",
        htmlSelectEndNode: "#to-end",
        htmlListNode: "#to-visit-list",
        htmlDistance: "#distance",
        htmlTimeToReach: "#timeToReach"
    },
    getstate: {
        selectedNode: null,
        fromNode: null,
        toNode: null
    },
    places: {
        coordinates: [],
        category: null,
        title: null,
        rating: null
    },
    checkboxes: []
};

let markers = [];

let places = [];
```

```
const fetchPlaces = async () => {
  try {
    const response = await fetch('/places');

    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    const data = await response.json();
       places = data.map(row => {
      let coordinates = row.coordinates;
      let longitude, latitude;

      if (coordinates) {
        coordinates = coordinates.replace('POINT(', '').replace(')', '');
        [longitude, latitude] = coordinates.split(' ').map(Number);
      }

      return {
        coordinates: [latitude, longitude],
        category: row.category,
        title: row.title,
        rating: parseFloat(row.rating)
      };
    });

    console.log('Places:', places);
       renderPlacesOnMap(places);

  } catch (error) {
    console.error('Error fetching places:', error);
  }
};
fetchPlaces();

maps = L.map('svg-map').setView([50.4657, 30.5166], 10);
mapLink = '<a href="http://openstreetmap.org">OpenStreetMap</a>';
L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
    attribution: '&copy; ' + mapLink + ' Contributors', maxZoom: 18,
}).addTo(maps);
maps._initPathRoot()
svg = d3.select("#svg-map").select("svg")
    .attr("class", "svgmap")
    .on("contextmenu", function () { d3.event.preventDefault(); })



maps.on("viewreset", redrawmapwhenviewchanges);

maps.on('click', function (e) {
    var nodeName = mapdata.allnodes.length;
    console.log(e.latlng.lat + ", " + e.latlng.lng);

    var newNode = {
        name: nodeName,
        x: e.latlng.lat,
        y: e.latlng.lng
    };
    getElevationWithRetry(newNode.x, newNode.y).then(elevation => {
        newNode.elevation = elevation !== null ? elevation : 0;
        mapdata.allnodes.push(newNode);
        redrawNodes();
        addNodeToSelect(nodeName);
    }).catch(error => {
        console.error("Failed to retrieve elevation:", error);
    });
});

function redrawmapwhenviewchanges() {
```

```
    redrawLines();
    redrawNodes();
}

function dragNode() {
    return function (d, i) {
        var d = d;
        var golf = true;
        var newLatLng;

        maps.on('mousemove', function (e) {
            if (golf) {
                newLatLng = {
                    lat: e.latlng.lat,
                    lng: e.latlng.lng
                };

                var nodeDatum = {
                    name: d.name,
                    x: newLatLng.lat,
                    y: newLatLng.lng
                };

                mapdata.allnodes[i] = nodeDatum;
                redrawLines();
                redrawNodes();
            } else {
                return;
            }
        });

        maps.on('mouseup', function () {
            golf = false;

            getElevationWithRetry(newLatLng.lat, newLatLng.lng).then(elevation => {
                mapdata.allnodes[i].elevation = elevation !== null ? elevation : 0;
                calculateDistancesbetweennodes();
                redrawLines();
                redrawNodes();
            }).catch(error => {
                console.error("Failed to retrieve elevation:", error);
            });

            return;
        });
    }
};

function redrawNodes() {

    svg.selectAll("g.nodes").data([]).exit().remove();

    var elements = svg.selectAll("g.nodes").data(mapdata.allnodes, function (d, i) { return d.name; });

    var nodesEnter = elements.enter().append("g")
        .attr("class", "nodes");

    elements.attr("transform", function (d, i) {

        return "translate(" +
            maps.latLngToLayerPoint(new L.LatLng(d.x, d.y)).x + "," +
            maps.latLngToLayerPoint(new L.LatLng(d.x, d.y)).y + ")";
    });

    nodesEnter.append("circle")
        .attr("nodeId", function (d, i) { return i; })
        .attr("r", '24')
        .attr("class", "node")
        .style("cursor", "pointer")
```

```
        .on('click', nodeClick)
        .on("mouseenter", function () { maps.dragging.disable(); })
        .on("mouseout", function () { maps.dragging.enable(); })
        .on('contextmenu', function (d, i) { startEndPath(i); })
        .call(dragManager)

    nodesEnter
        .append("text")
        .attr("nodeLabelId", function (d, i) { return i; })
        .attr("dx", "-5")
        .attr("dy", "5")
        .attr("class", "label")
        .on('contextmenu', function (d, i) { startEndPath(i); })
        .call(dragManager)
        .text(function (d, i) { return d.name });

    elements.exit().remove();
};

function redrawLines() {

    svg.selectAll("g.line").data([]).exit().remove();

    var elements = svg
        .selectAll("g.line")
        .data(mapdata.paths, function (d) { return d.id });

    var newElements = elements.enter();

    var group = newElements
        .append("g")
        .attr("class", "line");

    var line = group.append("line")
        .attr("class", function (d) {
            return "from" + mapdata.allnodes[d.from].name + "to" + mapdata.allnodes[d.to].name;
        })
                .attr("x1", function (d) { return maps.latLngToLayerPoint(new L.LatLng(mapdata.allnodes[d.from].x,
mapdata.allnodes[d.from].y)).x; })
                .attr("y1", function (d) { return maps.latLngToLayerPoint(new L.LatLng(mapdata.allnodes[d.from].x,
mapdata.allnodes[d.from].y)).y; })
                 .attr("x2", function (d) { return maps.latLngToLayerPoint(new L.LatLng(mapdata.allnodes[d.to].x,
mapdata.allnodes[d.to].y)).x; })
                 .attr("y2", function (d) { return maps.latLngToLayerPoint(new L.LatLng(mapdata.allnodes[d.to].x,
mapdata.allnodes[d.to].y)).y; });

    var text = group.append("text")
              .attr("x", function (d) { return parseInt((maps.latLngToLayerPoint(new L.LatLng(mapdata.allnodes[d.from].x,
mapdata.allnodes[d.from].y)).x + maps.latLngToLayerPoint(new L.LatLng(mapdata.allnodes[d.to].x,
mapdata.allnodes[d.to].y)).x) / 2) + 5; })
              .attr("y", function (d) { return parseInt((maps.latLngToLayerPoint(new L.LatLng(mapdata.allnodes[d.from].x,
mapdata.allnodes[d.from].y)).y + maps.latLngToLayerPoint(new L.LatLng(mapdata.allnodes[d.to].x,
mapdata.allnodes[d.to].y)).y) / 2) - 5; })
        .attr("class", "line-label")
        .style("font-size", "20px");;

    elements.selectAll("text")
        .text(function (d) {
            var pathType = d.type === 'Automobile' ? ' (Automobile)' : ' (Pedestrian)';
            return Math.round(mapdata.distances[d.from][d.to]) + " Meters" + pathType;
        });

    elements.exit().remove();
}

function LatLon(lat, lon) {
    this.lat = Number(lat);
    this.lon = Number(lon);
}
```

```
LatLon.prototype.distanceTo = function (point, radius) {
   if (!(point instanceof LatLon)) throw new TypeError('point is not LatLon object');
   radius = (radius === undefined) ? 6378137 : Number(radius);
   if (Number.prototype.toRadians === undefined) {
      Number.prototype.toRadians = function () { return this * Math.PI / 180; };
   }
   if (Number.prototype.toDegrees === undefined) {
      Number.prototype.toDegrees = function () { return this * 180 / Math.PI; };
   }
   var R = radius;
   var φ1 = this.lat.toRadians(), λ1 = this.lon.toRadians();
   var φ2 = point.lat.toRadians(), λ2 = point.lon.toRadians();
   var Δφ = φ2 - φ1;
   var Δλ = λ2 - λ1;
   var a = Math.sin(Δφ / 2) * Math.sin(Δφ / 2)
      + Math.cos(φ1) * Math.cos(φ2)
      * Math.sin(Δλ / 2) * Math.sin(Δλ / 2);
   var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
   var d = R * c;
   return d;
};


$('#exampleModal').on('show.bs.modal', function (event) {
   var button = $(event.relatedTarget)
   var recipient = button.data('whatever')
   var modal = $(this)
   modal.find('.modal-title').text('New message to ' + recipient)
   modal.find('.modal-body input').val("")

});

var dragManager = d3.behavior.drag()
   .on('dragstart', dragNodeStart())
   .on('drag', dragNode())
   .on('dragend', dragNodeEnd());

$('#setexample').on('change', function () {
   var value = $(this).val();
   if (value >= 1 && value <=4 ) {
      clearGraph();
      switch(value>=3){
         case true:
            maps.setView(new L.LatLng(41.0050507049894,28.974914252758026, 18));
            break;
         case false:
            maps.setView(new L.LatLng(50.4657, 30.5166), 18);
            break;
      }

      $.getJSON("/api/nodes/" + value, function (nodes) {
         if (nodes === undefined || nodes.length === 0) {
            console.log("** JSON format error or no nodes data:");
            console.log(nodes);
            return;
         }
         mapdata.allnodes = nodes;
         enrichNodesWithElevation();

         $.getJSON("/api/paths/" + value, function (paths) {
            if (paths === undefined || paths.length === 0) {
               console.log("** JSON format error or no paths data:");
               console.log(paths);
               return;
            }
            mapdata.paths = paths;

            mapdata.distances = [];
```

```
            mapdata.getstate.selectedNode = null;
            mapdata.getstate.fromNode = null;
            mapdata.getstate.toNode = null;
            $(mapdata.getui.htmlListNode).empty();

            mapdata.allnodes.forEach(function (node) {
               addNodeToSelect(node.name);
            });

            calculateDistancesbetweennodes();
            redrawLines();
            redrawNodes();
         });
      });
   }
});


$("#data-export").click(function (e) {

   e.stopPropagation()

   var exportData = JSON.stringify({
      nodes: mapdata.allnodes,
      paths: mapdata.paths
   });

   var target = $(this);

   var link = $("<a></a>")
      .addClass("exportLink")
      .click(function (e) { e.stopPropagation(); })
      .attr('target', '_self')
      .attr("download", "nodesandpaths.json")
      .attr("href", "data:application/json," + exportData)

   link.appendTo(target).get(0).click();

   $(".exportLink").remove();

});

$("#getmethere").on('click',function () {

   var valuelat = $("#latitude").val();
   var valuelong = $("#longitude").val();
   clearGraph();

   if (valuelat == '' || valuelong == '' ){
     alert("Please Enter Lat. and Long.");
   }
   else {
      maps.setView(new L.LatLng(valuelat, valuelong), 10);
   }
});

async function enrichNodesWithElevation() {
   const batchSize = 50;
   const delayBetweenBatches = 2000;

   for (let i = 0; i < mapdata.allnodes.length; i += batchSize) {
      const batch = mapdata.allnodes.slice(i, i + batchSize);
      console.log(`Processing batch ${Math.floor(i / batchSize) + 1}`);
      const elevations = await getElevations(batch);
      for (let j = 0; j < batch.length; j++) {
         mapdata.allnodes[i + j].elevation = elevations[j];
      }

      if (i + batchSize < mapdata.allnodes.length) {
```

```
            console.log(`Waiting ${delayBetweenBatches}ms before next batch...`);
            await new Promise(res => setTimeout(res, delayBetweenBatches));
        }
    }


    mapdata.allnodes.forEach((node, index) => {
        if (node.elevation === undefined || node.elevation === null) {
            console.warn(`Node ${index} does not have elevation data. Assigning default value 0.`);
            node.elevation = 0;
        }
    });
    console.log(`Fetching elevation completed.`);
}

$("#data-import").change(function (e) {
    e.stopPropagation();
    var files = e.target.files;
    var file = files[0];
    if (file === undefined) return;
    var reader = new FileReader();
    reader.onload = function () {
        try {
            var importedData = JSON.parse(this.result);
        }
        catch (exception) {
            console.log("** Error importing JSON: %s", exception);
            return;
        }
        if (importedData.nodes === undefined
            || importedData.paths === undefined
            || Object.keys(importedData).length !== 2) {
            console.log("** JSON format error:");
            console.log(importedData);
            return;
        }

        mapdata.allnodes = importedData.nodes;
        mapdata.paths = importedData.paths;
        mapdata.distances = [];
        mapdata.getstate.selectedNode = null;
        mapdata.getstate.fromNode = null;
        mapdata.getstate.toNode = null;
        $(mapdata.getui.htmlSelectStartingNode).empty();
        $(mapdata.getui.htmlSelectEndNode).empty();
        $(mapdata.getui.htmlSelectEndNode).append($("<option></option>").attr("value", null).text("None"));
        $(mapdata.getui.htmlListNode).empty();

        mapdata.allnodes.forEach(function (node) {
            addNodeToSelect(node.name);
        });

        enrichNodesWithElevation();

        calculateDistancesbetweennodes();
        redrawLines();
        redrawNodes();
    }
    reader.readAsText(file);
});


$('#getshortestroute').on('click', function () {
    console.log("Calculating Shortest Route");

    d3.selectAll("line").classed({ "shortest": false });
    calculateDistancesbetweennodes();

    const targetNodeArray = $(mapdata.getui.htmlListNode + " input:checkbox:checked")
```

```
    .map(function() {
        return $(this).val();
    }).get();

if (!$(mapdata.getui.htmlSelectStartingNode).val() || targetNodeArray.length === 0) {
    console.log("Either no starting node selected or targetNodeArray is empty. Exiting function.");
    return;
}

let sourceNode = $(mapdata.getui.htmlSelectStartingNode).val();
const targetNode = $(mapdata.getui.htmlSelectEndNode).val();
const selectedAlg = $("input[type='radio'][name='routingAlg']:checked").val();

$('#unreachable-list').empty();
const unreachablePoints = new Set();

let Distance = 0;
const startTime = Date.now();

while (targetNodeArray.length !== 0) {
    let results = null;
    let foundReachablePath = false;
    const unreachableThisRound = new Set();

    targetNodeArray.forEach(element => {
        console.log("Calculating route to element:", element);
        let temp;

        if (selectedAlg === 'Dijkstra') {
            temp = dijkstra(sourceNode, element);
        } else if (selectedAlg === 'AStar') {
            temp = aStar(sourceNode, element);
        }

        if (temp && temp.path) {
            foundReachablePath = true;
            if (results == null || results.distance > temp.distance) {
                results = temp;
            }
        } else {
            unreachableThisRound.add(element);
        }
    });

    if (foundReachablePath && results && results.path) {
        results.path.forEach(function (step) {
            const stepLine = d3.select(
                "line.from" + step.source + "to" + step.target + "," +
                "line.from" + step.target + "to" + step.source
            );
            stepLine.classed({ "shortest": true });
        });

        targetNodeArray.splice(targetNodeArray.indexOf(String(results.target)), 1);
        sourceNode = results.target;
        Distance += results.distance;
    } else {
        unreachableThisRound.forEach(node => unreachablePoints.add(node));
        targetNodeArray.splice(targetNodeArray.indexOf(sourceNode), 1);
    }
}

if (targetNode !== 'None') {
    let finalResults;

    if (selectedAlg === 'Dijkstra') {
        finalResults = dijkstra(sourceNode, targetNode);
    } else if (selectedAlg === 'AStar') {
        finalResults = aStar(sourceNode, targetNode);
```

```
        }

        if (finalResults && finalResults.path) {
            finalResults.path.forEach(function (step) {
                const stepLine = d3.select(
                    "line.from" + step.source + "to" + step.target + "," +
                    "line.from" + step.target + "to" + step.source
                );
                stepLine.classed({ "shortest": true });
            });
            Distance += finalResults.distance;
        } else {
            unreachablePoints.add(targetNode);
        }
    }

    unreachablePoints.forEach(node => {
        $('#unreachable-list').append(`<li>${node}</li>`);
    });

    const calculationTime = Date.now() - startTime;
    document.getElementById("calculationTime").innerText = calculationTime + " milliseconds";

    Distance = Distance / 1000;
    const travelSpeed = parseFloat($("input[type='radio'][name='travelType']:checked").val());
    $(mapdata.getui.htmlDistance).attr("value", Distance).text((Distance.toFixed(2)) + " Km");
    $(mapdata.getui.htmlTimeToReach).attr("value", (Distance / travelSpeed).toFixed(2)).text((Distance / travelSpeed).toFixed(2)
+ " hours");
});

$('#clearmap').on('click', function () {
    clearGraph();

});


function addNodeToSelect(nodeName) {
    $(mapdata.getui.htmlSelectStartingNode).append($("<option></option>").attr("value", nodeName).text(nodeName));
    $(mapdata.getui.htmlSelectEndNode).append($("<option></option>").attr("value", nodeName).text(nodeName));
    $(mapdata.getui.htmlListNode).append($("<div></div>").attr("id", "div"+String(nodeName)));
                                $("#div"+String(nodeName)).append($("<input></input>").attr("value",                nodeName).attr("id",
"check"+String(nodeName)).attr("type", "checkbox"));
    $("#div"+String(nodeName)).append($("<label></label>").attr("value", nodeName).text(nodeName));
};

function clearGraph() {
    mapdata.allnodes = [];
    mapdata.paths = [];
    $(mapdata.getui.htmlSelectStartingNode).empty();
    $(mapdata.getui.htmlSelectEndNode).empty();
    $(mapdata.getui.htmlSelectEndNode).append($("<option></option>").attr("value", null).text("None"));
    $(mapdata.getui.htmlListNode).empty();
    $(mapdata.getui.htmlSelectStartingNode).empty();
    $(mapdata.getui.htmlTimeToReach).empty();
    $("#results").empty();
    $('#svg-map').css({
        'background-image': 'url(' + null + ')'

    });
    redrawLines();
    redrawNodes();
};


function nodeClick(d, i) {
    console.log("node:click %s", i);
    console.log(d);

    d3.event.preventDefault();
```

```
    d3.event.stopPropagation();
};

function dragNodeStart() {
    return function (d, i) {
        console.log("dragging node " + i);

    }
};

function dragNodeEnd() {
    return function (d, i) {
        console.log("node " + i + " repositioned");
    }
};

function killEvent() {
    if (d3.event.preventDefault) {
        d3.event.preventDefault();
        d3.event.stopPropagation();
    }
};

function startEndPath(index) {
    d3.event.stopPropagation();
    d3.event.preventDefault();
    if (mapdata.getstate.fromNode === null) {

        mapdata.getstate.fromNode = index;
    }
    else {
        if (mapdata.getstate.fromNode === index) {

            return;
        }

        mapdata.getstate.toNode = index;
        console.log(index + " Node lar");
        var selectedType = getSelectedTravelType();

        var existingPath = mapdata.paths.find(path =>
            (path.from === mapdata.getstate.fromNode && path.to === index) ||
            (path.from === index && path.to === mapdata.getstate.fromNode)
        );

        if (existingPath) {
            existingPath.type = selectedType;
        } else {
            var pathDatum = {
                id: mapdata.paths.length,
                from: mapdata.getstate.fromNode,
                to: index,
                type: selectedType
            };
            mapdata.paths.push(pathDatum);
        }
        calculateDistancesbetweennodes();
        redrawLines();
        redrawNodes();
        mapdata.getstate.fromNode = null;
        mapdata.getstate.toNode = null;
    }
};

function getSelectedTravelType() {
    var travelTypeRadios = document.getElementsByName('travelType');
    for (var i = 0; i < travelTypeRadios.length; i++) {
        if (travelTypeRadios[i].checked) {
            return travelTypeRadios[i].value == '4.5' ? 'Pedestrian' : 'Automobile';
```

```
        }
    }
    return null;
}

function calculateDistancesbetweennodes() {
    mapdata.distances = [];

    for (var i = 0; i < mapdata.allnodes.length; i++) {
        mapdata.distances[i] = [];
        for (var j = 0; j < mapdata.allnodes.length; j++) {
            mapdata.distances[i][j] = 'x';
        }
    }

    for (var i = 0; i < mapdata.paths.length; i++) {
        var sourceNodeId = parseInt(mapdata.paths[i].from);
        var targetNodeId = parseInt(mapdata.paths[i].to);
        var sourceNode = mapdata.allnodes[sourceNodeId];
        var targetNode = mapdata.allnodes[targetNodeId];
        var p1 = new LatLon(sourceNode.x, sourceNode.y);
        var p2 = new LatLon(targetNode.x, targetNode.y);
        var d = p1.distanceTo(p2);

        mapdata.distances[sourceNodeId][targetNodeId] = d;
        mapdata.distances[targetNodeId][sourceNodeId] = d;

        if (!mapdata.paths[i].type) {
            mapdata.paths[i].type = getSelectedTravelType();
        }

        console.log(mapdata.paths[i]);
    }
}

function dijkstra(start, end) {
    var nodeCount = mapdata.distances.length,
        infinity = 99999, // infinity
        shortestPath = new Array(nodeCount),
        nodeChecked = new Array(nodeCount),
        pred = new Array(nodeCount);

    var selectedTravelType = getSelectedTravelType();

    for (var i = 0; i < nodeCount; i++) {
        shortestPath[i] = infinity;
        pred[i] = null;
        nodeChecked[i] = false;
    }

    shortestPath[start] = 0;

    for (var i = 0; i < nodeCount; i++) {
        var minDist = infinity;
        var closestNode = null;

        for (var j = 0; j < nodeCount; j++) {
            if (!nodeChecked[j]) {
                if (shortestPath[j] <= minDist) {
                    minDist = shortestPath[j];
                    closestNode = j;
                }
            }
        }

        nodeChecked[closestNode] = true;

        for (var k = 0; k < nodeCount; k++) {
            if (!nodeChecked[k]) {
```

```
            var nextDistance = distanceBetween(closestNode, k, mapdata.distances, selectedTravelType);
            if ((parseInt(shortestPath[closestNode]) + parseInt(nextDistance)) < parseInt(shortestPath[k])) {
               soFar = parseInt(shortestPath[closestNode]);
               extra = parseInt(nextDistance);
               shortestPath[k] = soFar + extra;
               pred[k] = closestNode;
            }
         }
      }
   }

   if (shortestPath[end] < infinity) {
      var newPath = [];
      var step = {
         target: parseInt(end)
      };

      var v = parseInt(end);

      while (v >= 0) {
         v = pred[v];
         if (v !== null && v >= 0) {
            step.source = v;
            newPath.unshift(step);
            step = {
               target: v
            };
         }
      }

      totalDistance = shortestPath[end];

      return {
         mesg: 'Status: OK',
         path: newPath,
         source: start,
         target: end,
         distance: totalDistance
      };
   } else {
      return {
         mesg: 'Sorry No path found',
         path: null,
         source: start,
         target: end,
         distance: 0
      };
   }

   function distanceBetween(fromNode, toNode, distances, travelType) {
      var path = mapdata.paths.find(p => p.from == fromNode && p.to == toNode);
      if (!path) {
         path = mapdata.paths.find(p => p.from == toNode && p.to == fromNode);
      }

      if (!path || (travelType === 'Automobile' && path.type !== 'Automobile')) {
         return infinity;
      }

      var dist = distances[fromNode][toNode];
      if (dist === 'x') dist = infinity;
      return dist;
   }
}

function aStar(start, end) {
   var nodeCount = mapdata.distances.length,
      infinity = 99999,
      shortestPath = new Array(nodeCount),
```

```
    nodeChecked = new Array(nodeCount),
    pred = new Array(nodeCount),
    gScore = new Array(nodeCount),
    fScore = new Array(nodeCount);

var selectedTravelType = getSelectedTravelType();

for (var i = 0; i < nodeCount; i++) {
    shortestPath[i] = infinity;
    gScore[i] = infinity;
    fScore[i] = infinity;
    pred[i] = null;
    nodeChecked[i] = false;
}

gScore[start] = 0;
fScore[start] = heuristic(start, end);

while (true) {
    var closestNode = null;
    var minFScore = infinity;

    for (var i = 0; i < nodeCount; i++) {
        if (!nodeChecked[i] && fScore[i] < minFScore) {
            minFScore = fScore[i];
            closestNode = i;
        }
    }

    if (closestNode === null) {
        return { mesg: 'Sorry, no path found', path: null, source: start, target: end, distance: 0 };
    }

    if (closestNode === parseInt(end)) {
        var newPath = [];
        var v = end;

        while (v !== null) {
            var step = { target: v };
            v = pred[v];
            if (v !== null && v >= 0) {
                step.source = v;
                newPath.unshift(step);
            }
        }

        var totalDistance = gScore[end];
        return {
            mesg: 'Status: OK',
            path: newPath,
            source: start,
            target: end,
            distance: totalDistance
        };
    }

    nodeChecked[closestNode] = true;

    for (var i = 0; i < nodeCount; i++) {
        if (!nodeChecked[i]) {
            var tentativeGScore = gScore[closestNode] + distanceBetween(closestNode, i, mapdata.distances, selectedTravelType);

            if (tentativeGScore < gScore[i]) {
                pred[i] = closestNode;
                gScore[i] = tentativeGScore;
                fScore[i] = gScore[i] + heuristic(i, end);
            }
        }
    }
```

```
    }

    function distanceBetween(fromNode, toNode, distances, travelType) {
        var path = mapdata.paths.find(p => p.from == fromNode && p.to == toNode);
        if (!path) {
            path = mapdata.paths.find(p => p.from == toNode && p.to == fromNode);
        }
        if (!path || (travelType === 'Automobile' && path.type !== 'Automobile')) {
            return infinity;
        }

        var dist = distances[fromNode][toNode];
        if (dist === 'x') dist = infinity;
        return dist;
    }

    function heuristic(nodeA, nodeB) {
        var sourceNode = mapdata.allnodes[nodeA];
        var targetNode = mapdata.allnodes[nodeB];
        if (!sourceNode || !targetNode) return Infinity;

        var dx = sourceNode.x - targetNode.x;
        var dy = sourceNode.y - targetNode.y;
        var dz = sourceNode.elevation - targetNode.elevation;

        var euclideanDistance = Math.sqrt(dx * dx + dy * dy + dz * dz);
        console.log(`Heuristic between ${nodeA} and ${nodeB}: ${euclideanDistance}`);

        return euclideanDistance;
    }
}

async function getElevations(locations, retries = 3, delayMs = 1000) {
    const locString = locations.map(loc => `${loc.y},${loc.x}`).join('|');
    const url = `https://api.open-elevation.com/api/v1/lookup?locations=${locString}`;

    for (let attempt = 1; attempt <= retries; attempt++) {
        try {
            const response = await fetch(url);
            if (!response.ok) {
                throw new Error(`Error fetching elevations: ${response.statusText}`);
            }
            const data = await response.json();
            if (data && data.results) {
                return data.results.map(result => result.elevation);
            } else {
                throw new Error('Invalid elevation data format');
            }
        } catch (error) {
            console.error(`Attempt ${attempt} failed: ${error.message}`);
            if (attempt < retries) {
                console.log(`Retrying in ${delayMs}ms...`);
                await new Promise(res => setTimeout(res, delayMs));
            } else {
                console.error('All retry attempts failed.');
                return locations.map(() => 0);
            }
        }
    }
}

async function getElevationWithRetry(lat, lon, retries = 3, delayMs = 1000) {
    for (let attempt = 1; attempt <= retries; attempt++) {
        try {
            const elevation = await getElevation(lat, lon);
            return elevation;
        } catch (error) {
            console.error(`Attempt ${attempt} failed: ${error.message}`);
            if (attempt < retries) {
```

```
            console.log(`Retrying in ${delayMs}ms...`);
            await new Promise(res => setTimeout(res, delayMs));
          } else {
            console.error('All retry attempts failed. Assigning default elevation.');
            return null;
          }
        }
      }
    }

    async function getElevation(lat, lon, retries = 3, delayMs = 1000) {
      const url = `https://api.open-elevation.com/api/v1/lookup?locations=${lat},${lon}`;

      for (let attempt = 1; attempt <= retries; attempt++) {
        try {
          const response = await fetch(url);
          if (!response.ok) {
            throw new Error(`Error fetching elevation: ${response.statusText}`);
          }
          const data = await response.json();
          if (data && data.results && data.results[0]) {
            return data.results[0].elevation;
          } else {
            throw new Error('Invalid elevation data format');
          }
        } catch (error) {
          console.error(`Attempt ${attempt} failed: ${error.message}`);
          if (attempt < retries) {
            console.log(`Retrying in ${delayMs}ms...`);
            await new Promise(res => setTimeout(res, delayMs));
          } else {
            console.error('All retry attempts failed. Assigning default elevation.');
            return 0;
          }
        }
      }
    }
    // ==== PLACES DATABASE ====

    function populateMarkers(selectedCategories) {
      markers.forEach(marker => {
        maps.removeLayer(marker);
      });
      markers = [];

      function createMarker(place) {
        let markerIcon;

        switch (place.category) {
          case 'Restaurants':
            markerIcon = L.icon({
              iconUrl: 'images/restaurant-icon.png',
              iconSize: [32, 32],
            });
            break;
          case 'Hotels':
            markerIcon = L.icon({
              iconUrl: 'images/hotel-icon.png',
              iconSize: [32, 32],
            });
            break;
          case 'Attractions':
            markerIcon = L.icon({
              iconUrl: 'images/attraction-icon.png',
              iconSize: [32, 32],
            });
            break;
        }
```

```
    const marker = L.marker(place.coordinates, { icon: markerIcon }).addTo(maps);
    marker.bindPopup(`<b>${place.title}</b><br>Rating: ${place.rating}`);
    marker.on('mouseover', function (e) {
      this.openPopup(e.latlng);
      });
      marker.on('mouseout', function (e) {
         this.closePopup();
      });
    return marker;
   }

   checkboxes.forEach(checkbox => {
    if (checkbox.checked) {
      const category = checkbox.value;
      const categoryMarkers = places.filter(place => place.category === category);
      markers.push(...categoryMarkers.map(createMarker));
    }
   });
  }

$('input[name="placeCheckbox"]').change(function () {
   checkboxes = document.querySelectorAll('#placesBlock input[type="checkbox"]');

   var selectedCategories = [];
   checkboxes.forEach(checkbox => {
    if (checkbox.checked) {
      selectedCategories.push(checkbox.value);
    }
   });

   populateMarkers(selectedCategories);

});
```

## Файл index.html

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <meta name="description" content="Routing system">

  <title>Routing system</title>


  <!-- Bootstrap core CSS -->
   <link  rel="stylesheet"  href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta/css/bootstrap.min.css"  integrity="sha384-
/Y6pD6FV/Vv2HJnA6t+vslU6fwYXjCFtcEpHbNJ0lyAFsXTsjBbfaDjzALeQsN6M"
   crossorigin="anonymous">
   <link
   rel="stylesheet"
   href="https://cdn.jsdelivr.net/npm/leaflet@0.7.7/dist/leaflet.css"
   />
  <!-- Custom styles for this template -->
  <link href="style/justified-nav.css" rel="stylesheet">
</head>

<body>

  <div class="container">

    <div class="masthead">
      <h3 class="text-muted">Routing System</h3>
    </div>

    <div class="jumbotron">
      <h1>Routing system. How to work.</h1>
```

```
  </div>
  <div class="row">
    <div class="col-lg-12">
      <p class="lead">Click to add nodes and then right-click a node and then again right click on another node to link them each other. You can freely reposition them by dragging the nodes by holding at the center of a circle of the node and moving according to background map. In order to draw on the map effortlessly, you can enter Latitude and Longitude of Your desired location on the map and quickly go there by clicking get me there button. For routing, select starting node and then select nodes to visit and, if required, end node along with the travel method and routing algorithm and click to start to get the shortest path between selected nodes. </p>
    </div>
  </div>

  <!-- Main Map Canvas -->
  <div style="display: flex;">
    <div class="row">
    <div class="col-lg-12">

      <div id="svg-map" style="width: 1200px; height: 1200px" class="card">

      </div>
      <form style="margin-top:5px;">

      <div class="form-row align-items-center">

        <div class="col-auto">
        <button type="button" class="btn btn-success" id="data-export"> Export Data</button>
        </div>
        <div class="col-auto">
        <label class="custom-file">
          <input type="file" id="data-import" class="custom-file-input">
          <span class="custom-file-control">Import Data</span>
        </label>
        </div>
      </div>
      </form>
    </div>
    </div>
    <div class="row">

    <div class="col-lg-12">
      <form>
      <div class="form-row align-items-center">
       <div class="col-auto" id="placesBlock">
        <h2>Places</h2>
        <div class="col-auto">
            <label for="restaurants-checkbox"><input name="placeCheckbox" type="checkbox" id="restaurants-checkbox" value="Restaurants" /> Restaurants</label>
        </div>
        <div class="col-auto">
          <label for="hotels-checkbox"><input name="placeCheckbox" type="checkbox" id="hotels-checkbox" value="Hotels" /> Hotels</label>
        </div>
        <div class="col-auto">
            <label for="attractions-checkbox"><input name="placeCheckbox" type="checkbox" id="attractions-checkbox" value="Attractions" /> Attractions</label>
        </div>
       </div>

       <div class="col-auto">
        <div class="col-auto">
          <label class="mr-sm-2" for="from">Starting point</label>
          <select id="from-starting" class="custom-select mb-2 mr-sm-2 mb-sm-0"></select>
        </div>

        <div class="col-auto">
          <label class="mr-sm-2" for="to">To</label>
          <select id="to-end" class="custom-select mb-2 mr-sm-2 mb-sm-0">
            <option value="">None</option>
          </select>
```

```
          </div>
        </div>
        <div class="col-auto">
          <div class="form-row align-items-center">
            <input type="radio" style="margin: 10px;" name="travelType" id="travelType" value="4.5" checked>
            <label style="margin: 10px;">On foot</label>
          </div>
          <div class="form-row align-items-center">
            <input type="radio" style="margin: 10px;" name="travelType" id="travelType" value="60">
            <label style="margin: 10px;">On vehicle</label>
          </div>
        </div>
        <div class="col-auto">
          <label class="mr-sm-2" for="to">To visit:</label>
        <ul id="to-visit-list" style="height: 200px; overflow:hidden; overflow-y:scroll;">
          <div>

          </div>
        </ul>
        </div>

        <div class="col-auto">
          <div class="form-row align-items-center">
            <input type="radio" style="margin: 10px;" name="routingAlg" id="routingAlg" value="Dijkstra" checked>
            <label style="margin: 10px;">Dijkstra</label>
          </div>
          <div class="form-row align-items-center">
            <input type="radio" style="margin: 10px;" name="routingAlg" id="routingAlg" value="AStar">
            <label style="margin: 10px;">A*</label>
          </div>
        </div>

        <div class="col-auto">
        <button type="button" id="clearmap" class="btn btn-danger">Clear Map</button>
        </div>
        <div class="col-auto">
          <button type="button" id="getshortestroute" class="btn btn-success" title="find shortest path between nodes"> Start
Route</button>
        </div>

        <div class="col-auto">
          <label class="mr-sm-2" for="from">Distance to the destination: </label>
          <label id="distance"></label>
        </div>
        <div class="col-auto">
          <label class="mr-sm-2" for="from">Time to reach the destination on foot: </label>
          <label id="timeToReach"></label>
        </div>
        <div class="col-auto">
          <label class="mr-sm-2" for="from">Calculation time: </label>
          <label id="calculationTime"></label>
        </div>
        <div class="col-auto">
          <label class="mr-sm-2" for="from">Unreachable points: </label>
          <ul id="unreachable-list" style="height: 200px; overflow:hidden; overflow-y:scroll;">
        </div>


        <div class="col-auto">
        <label class="sr-only" for="inlineFormInput">Latitude</label>
        <input type="text" class="form-control mb-2 mb-sm-0" id="latitude" placeholder="Enter Latitude">
        </div>
        <div class="col-auto">
        <label class="sr-only" for="inlineFormInput">Longitude</label>
        <input type="text" class="form-control mb-2 mb-sm-0" id="longitude" placeholder="Enter Longitude">
        </div>

        <div class="col-auto">
        <button type="button" id= "getmethere"class="btn btn-primary">Get Me There</button>
```

```
            </div>

            <div class="col-auto" style="margin-top: 5px;">

            <label class="mr-sm-2" for="inlineFormCustomSelect">Load Examples</label>
            <select class="custom-select mb-2 mr-sm-2 mb-sm-0" id="setexample">
               <option value="0">Ex:</option>
               <option value="1">One</option>
               <option value="2">Two</option>
               <option value="3">Three</option>
               <option value="4">Four</option>


            </select>
            </div>
          </div>
          </form>
        </div>
        </div>

    </div>

    <!-- Site footer -->
    <footer class="footer">
    </footer>

  </div>
  <!-- /container -->

  <!-- Bootstrap core JavaScript
    ================================================== -->

  <!-- <script type="text/javascript" src="js/d3.v3.min.js"></script> -->
  <script src="https://code.jquery.com/jquery-3.2.1.min.js"></script>
          <script          src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.11.0/umd/popper.min.js"          integrity="sha384-
b/U6ypiBEHpOf/4+1nzFpr53nxSS+GLCkfwBdFNTxtclqqenISfwAzpKaMNFNmj4"
    crossorigin="anonymous"></script>
          <script          src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta/js/bootstrap.min.js"          integrity="sha384-
h0AbiXch4ZDo7tp9hKZ4TsHbi047NrKGLO3SEJAg45jXxnGIfYzk4Si90RDIqNm1"
    crossorigin="anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/d3@3.3.0/d3.min.js"></script>


  <script
  src="https://cdn.jsdelivr.net/npm/leaflet@0.7.7/dist/leaflet.js">
</script>

  <script src="js/main.js"></script>
  <!-- IE10 viewport hack for Surface/desktop Windows 8 bug -->
  <script src="js/ie10-viewport-bug-workaround.js"></script>
</body>

</html>
```

**Додаток** 5

# ВІДГУК

на кваліфікаційну роботу магістра

**“Дослідження алгоритмів знаходження оптимальних маршрутів у складі Web-додатку на базі мікросервісної архітектури”**

студента групи 126м-23-1

Панасенка Івана Олеговича

1. Метою кваліфікаційної роботи є дослідження алгоритмів знаходження оптимальних маршрутів з подальшою вдосконаленою реалізацією у складі Web-додатку на базі мікросервісної архітектури.

2. Завдання та зміст кваліфікаційної роботи відповідає основній меті – оцінці знань та ступеня підготовленості студента за спеціальністю 126 "Інформаційні системи та технології".

3. Тема роботи представляється актуальною, оскільки в інформаційних системах маршрутизації існують певні обмеження в алгоритмах рішення задачі. В своїй роботі студент спробував подолати такі обмеження, використовуючи додатковий обчислювальний інструментарій із залученням технології контейнеризації. Побудова додатку на базі мікросервісної архітектури дозволила б використовувати математичний апарат надання оптимального маршруту через певний API, а також призвела до більш гнучкого адаптування до змін при оновлені частин системи. В роботі було запропоноване рішення сучасним інструментарієм на базі платформи Node.js.

4. У процесі виконання роботи дипломник опрацював певний обсяг теоретичного матеріалу з тематики роботи.

5. У якості зауважень до роботи слід відзначити невдосконалену реалізацію мікросервісної архітектури, а також не в повній мірі прораховані варіанти маршрутів з урахуванням географічних особливостей місцевості.

6. Серед переваг виконаної роботи слід відзначити підготовлену на високому фаховому рівні практичну складову.

7. Відзначу, що студентом були підготовлені тези доповіді на XII Міжнародну науково-технічну конференцію студентів, аспірантів та молодих вчених “Молодь: наука та інновації”, що проходила у місті Дніпро 13–15 листопада 2024 року. Студент зайняв почесне друге місце з наданням сертифікату учасника.

8. Незважаючи на недоліки, кваліфікаційна робота заслуговує оцінку "відмінно" (90 балів), а студент Панасенко Іван Олегович – присвоєння йому відповідної кваліфікації за спеціальністю 126 "Інформаційні системи та технології".

Керівник кваліфікаційної роботи,
доцент кафедри ІТКІ, канд. техн. наук  І.М. Гаркуша

# ДОДАТОК В

## РЕЦЕНЗІЯ

**на кваліфікаційну роботу магістра**

**студента групи 126м-23-1**

**Панасенка Івана Олеговича.**

**Тема роботи: «Дослідження алгоритмів знаходження оптимальних маршрутів у складі Web-додатку на базі мікросервісної архітектури»**

Кваліфікаційна робота присвячена актуальній темі, в ході виконання якої виконується розробка Web-додатку на базі мікросервісної архітектури для задач знаходження оптимальних маршрутів. Обрана тема є актуальною, а розглянуті у кваліфікаційній роботі технології можуть бути успішно використані при створені, наприклад, геоінформаційних систем оперування просторовими даними та картами. Тому вміст роботи відповідає основній меті – перевірці знань та ступеня підготовленості студента за спеціальністю 126 «Інформаційні системи та технології».

Повнота та глибина рішення задач в роботі є недостатньо розкритою.

Оформлення пояснювальної записки кваліфікаційної роботи виконано з деякими відхиленнями від нормативних вимог.

В роботі магістр представив практичну реалізацію дослідження, використовуючи мову програмування JavaScript, платформу Node.js, вільне API – Open-Elevation API та технологію контейнеризації Docker, що є безумовним плюсом при виконанні роботи за цією спеціальністю.

До зауважень по роботі слід віднести наступні.

1. В пояснювальній записці відсутні схеми алгоритмів маршрутизації, які реалізовані в програмному продукті.

2. Не проведено порівняльний аналіз результатів побудови маршрутів із відомими картографічними системами (наприклад, Google Maps або подібними).

3. Розбиття на мікросервіси та їх API в пояснювальній записці нечітко окреслено.

Вважаю, що незважаючи на недоліки, кваліфікаційна робота заслуговує на оцінку «відмінно» (90 балів), а студент Панасенко Іван Олегович присудження йому відповідної кваліфікації за спеціальністю 126 «Інформаційні системи та технології».

**Рецензент,**
**доцент кафедри ПЗКС НТУ "ДП",**
**канд. техн. наук** **А.Л. Ширін**

**Міністерство освіти і науки України**
**Національний технічний університет "Дніпровська політехніка"**

# ПОДАННЯ
# ГОЛОВІ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ
# ВІДОМОСТЕЙ ДО ЗАХИСТУ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Направляється студент(ка) Панасенко І.О. до захисту кваліфікаційної роботи
за спеціальністю 126 Інформаційні системи та технології
на тему: *Дослідження алгоритмів знаходження оптимальних маршрутів у складі Web-додатку на базі мікросервісної архітектури*

Кваліфікаційна робота і рецензія додаються.
Декан факультету (директор інституту) Удовик І.М. _____
(підпис)

## Довідка про успішність

Панасенко І.О. за період навчання в університеті, на факультеті Інформаційних технологій з 2023 р. до 2025 р. повністю виконав (ла) освітню програму за спеціальністю з таким розподілом оцінок за
**інституційною шкалою: відмінно - 64.71 %, добре - 23.53 %, задовільно - 11.76 %.**

Секретар факультету _____

## Висновок керівника

(зазначається відповідність змісту роботи, вимогам до рівня вищої освіти за НРК та компетентностям освітньої програми, оцінка виконання завдання)

Студент(ка) Панасенко І.О. виконав кваліфікаційну роботу відповідно до поставленого завдання. Зміст роботи відповідає вимогам 2-го рівня вищої освіти за НРК та компетентностям освітньої програми.
Вважаю, що кваліфікаційну роботу студента Панасенко І.О. можна оцінити на відмінно, 90, А при відповідному захисту.

Керівник_проекту (роботи) _____ Гаркуша І. М., доц.
"20" грудня 2024 року

## Висновок кафедри про кваліфікаційну роботу

Кваліфікаційну роботу розглянуто. Студент(ка) Панасенко І.О.
допускається до захисту цієї роботи в екзаменаційній комісії.

Завідувач кафедри Інформаційних технологій та комп'ютерної інженерії

_____ д.т.н., професор Гнатушенко В.В.
(підпис)
"20" 12 2024 року

Сформовано в ІАС "Деканат" 09.12.2024р. 12:19:37