АНОТАЦІЯ

Актуальність теми роботи полягає в тому, що стрімке впровадження цифрових технологій задля оптимізації процесу рішень задач сучасного бізнесу, створило великий попит на високопродуктивні спеціалізовані програмні системи. Одним із видів таких систем є географічні інформаційні системи. Висока вартість, складність в інтеграції та закритість вихідного коду більшості доступних на ринку ГІС-систем та сервісів, роблять їх недоступними для середнього та малого бізнесу. Цю ситуацію можна покращити шляхом дослідження, розширення та розробки оптимальних процесів інтеграції існуючих рішень з відкритим вихідним кодом.
Об’єктом дослідження є можливість створення сервісу побудови маршрутів з підтримкою нестандартного профілю маршрутизації, а саме розрахунком маршруту для трамваю. Також цей сервіс повинен підтримувати операції прямого та зворотнього геокодування та бути гнучким в розгортанні на широкому спектрі серверного обладнання.
Мета роботи полягає у створенні прототипу географічної інформаційної системи яка вирішує задачі маршрутизації (для легкового авто та трамваїв), здійснює операції геокодування та орієнтована на взаємодію через Web API.
Рішення базується на проектах з відкритим вихідним кодом, зокрема GraphHopper Routing Engine - система прокладення маршрутів, Nominatim - система прямого та зворотнього геокодування, OpenStreetMap - картографічний сервіс [14], який виступатиме в якості джерела геолокаційних даних та Docker Engine - платформа ізоляції, конфігурації та запуску додатків
Пояснювальна записка містить: __с., __рис., __табл., __дод.
Ключові слова:

SUMMARY

The urgency of the topic is that the rapid introduction of digital technologies to optimize the process of solving the problems of modern business, has created a great demand for high-performance specialized software systems. One type of such systems is geographical information systems. The high cost, complexity of integration and closure of the source code of most commercially available GIS systems and services make them inaccessible to medium and small businesses. This situation can be improved by researching, extending, and developing optimal processes for integrating existing open source solutions.
The object of the study is the ability to create a route-building service that supports a non-standard routing profile, namely the calculation of a route for a tram. This service should also support forward and reverse geocoding operations and be flexible to deploy across a wide range of server hardware.
The purpose of the work is to create a prototype geographical information system that solves routing problems (for cars and trams), performs geocoding operations and is oriented towards interaction through the Web API.

The solution is based on open source projects, including GraphHopper Routing Engine, a routing system, Nominatim, a direct and reverse geocoding system, OpenStreetMap, a mapping service, and a Docker Engine isolation platform, configuration and isolation platform launching applications

Explanatory note contains: __p., __fig., __tab., __app.
Keywords:

ЗМІСТ

Вступ

В дипломній роботі буде досліджуватися та розроблятися сервіс побудови маршрутів та прямо-зворотнього геокодування, який являє собою дві окремі програмних системи, які дозволять здійснювати прокладку маршруту через вказані географічні точки та виконувати операції геокодування.
Щоб реалізувати подані системи перш за все необхідно виділити основні дії які вони повинні виконувати. Це:
1. отримувати вхідні дані від користувача;
2. прокладати маршрути;
3. виконувати операції прямого та зворотнього геокодування;
4. повертати результат обчислень користувачу.
Метою дипломної роботи є дослідження та розробка географічної інформаційної системи яка вирішує задачі маршрутизації (для легкового авто та трамваїв), здійснює операції геокодування та орієнтована на взаємодію через Web API.
Для досягнення поставленої мети необхідно розробити систему, яка дозволить:
1. здійснювати прокладку маршруту для легкового авто та трамваю через Web API ;
2. здійснювати перетворення опису об’єкта в його відображення на земній поверхні (пряме геокодування) через Web API;
3. здійснювати витяг описової інформації з вказаного об’єкта карти (зворотне геокодування) через Web API;
Для розробки системи необхідно реалізувати серверну частину для кожного з сервісів. В більшості випадків, для вирішення задач подібного рівня складності доцільно брати за основу вже готові рішення або фреймворки. Саме тому реалізація сервісів полягатиме в розширенні та конфігурації готових рішень з відритим вихідним кодом та підбору оптимального способу їхнього розгортання.
Для реалізації функціональності прокладки маршрутів обрано проект з відкритим вихідним кодом GraphHopper Routing Engine, який швидко працює, невибагливий до апаратних ресурсів та написаний на Java. Вихідний код буде досліджено та розширено для імплементації підтримки профілю руху по трамвайним коліям.
Для реалізації функціональності прямого та зворотнього геокодування обрано проект Nominatim, який також має відкритий вихідний код, легкий в

конфігурації та використовується як частина інфраструктури картографічного сервісу OpenStreetMap.
Джерелом картографічних даних для обох сервісів буде дамп даних обраного фрагменту мапи сервісу OpenStreetMap.
Робота готових сервісів буде відбуватись на базі платформи ізоляції, конфігурації та запуску додатків Docker Engine у вигляді контейнерів.

1. Розробка програмної системи

2. Аналіз вимог до програмної системи

1. Аналіз предметної області

З алгоритмічної точки зору, розрахунок маршруту між точками являє собою класичну задачу пошуку оптимального шляху між вершинами зваженого графу.
1. Задача пошуку оптимального шляху.
Задача пошуку найкращого маршруту на карті може бути представлена у формі графа, як зображено на рисунку 1.1. Так вершини графа будуть відображенням таких об’єктів карти, як будівлі, підприємства, пам’ятники, тощо, а ребра - вулиці між цими об’єктами.

Рисунок 1.1 - Зважений граф

Пошук найкоротшого шляху, наприклад, з вершини ОО до ОО, потребує пошук колекції ребер, що з’єднують між собою вузли ОО та ОО, для якої сума вагів ребер буде найменшою. У цьому випадку довжина найкоротшого шляху від деякого вузла ОО до деякого вузла ОО називатися відстанню від ОО до ОО.
Розглянемо орієнтований граф ОО = (ОО, ОО), де ОО - скінченний набір вузлів і ОО - набір ребер між цими вузлами. Число вузлів |ОО| будемо позначати ОО, а число ребер |ОО| будемо позначати m. Кожне ребро ОО має вагу ОО (ОО). Шлях визначається послідовністю вузлів (ОО1, ... , ОООО), для яких (ОО1, ОООО+1) ∈ ОО для 1 < ОО < ОО. Вузол який з'єднаний деяким ребром з певним вузлом ОО, називається сусідом ОО. Вузол, який передує іншому вузлу шляху, називається його батьком в межах цього шляху. Аналогічно, дитиною називають наступника вузла на певному шляху.
Якщо початковим вузлом є вершина ОО ∈ ОО і кінцевим вузлом ОО ∈ ОО, то найкоротший шлях визначається як шлях (ОО, ... , ОО), який має мінімальну суму ваг всіх ребер на шляху. Довжина найкоротшого шляху від ОО до вузла ОО визначається як ОО(ОО) і також називається відстанню від ОО до ОО.
2. Алгоритм Дейкстри.

Розроблений голландським вченим Едсгером Дейкстри алгоритм служить для пошуку найкоротших шляхів з однієї наперед заданої вершини графа до всіх інших. З його допомогою, при наявності спеціально оформлених вхідних даних, можна вирішувати задачу пошуку найкращих шляхів з одного конкретного місця до кожного з багатьох інших.
Мінусом даного методу є неможливість роботи з графами, в яких є ребра з негативною вагою. Якщо, наприклад, деяка програмна система передбачає можливість вибору небажаних варіантів прокладання маршрутів, то для роботи з нею варто скористатися іншим алгоритмом [1].
Програмна реалізація алгоритму базується на оброці та зберіганні даних в два масиви: логічний visited - для зберігання інформації про відвідані вершини і чисельний distance, в який будуть заноситися знайдені найкоротші шляхи в чисельному еквіваленті. Отже, є граф G = (V, E). Кожна з вершин цього графу що входить в множину V, спочатку відзначається не відвіданою, тобто всім елементам масиву visited присвоюється значення false.
Оскільки на момент старту обчислень найкоротші варіанти шляхів ще не знайдено, в елементи масиву distance, які містять в собі довжини векторів між вершинами, записуються такі довжини, які свідомо більші будь-якого розміру потенційного шляху (зазвичай це число називають нескінченністю, але на практиці використовують, наприклад максимальне значення конкретного типу даних). В якості стартового пункту вибирається певна вершина s і їй присвоюється нульова довжина шляху: distance [s] = 0, оскільки немає ребра з s в s (алгоритм не передбач графові петлі).
Далі, знаходяться всі сусідні вершини (в які є ребро з s). Для прикладу такими вершинами будуть t і u. І по черзі досліджуються, а саме обчислюється вартість маршруту з s в кожну з цих вершин. Цілком ймовірно, що в ту чи іншу вершину з s існує декілька можливих варіантів побудувати шлях, тому ціну шляху в таку вершину в масиві distance доведеться переглядати з кожною ітерацією, тим самим перезаписуючи найбільше (неоптимальне) значення новим знайденим найменшим.
На рисунку 1.2 наглядно візуалізовано послідовність кроків пошуку найкоротших шляхів від першої вершини.

Рисунок 1.2 - Алгоритм Дейкстри
Після обробки суміжних з s вершин вона в масиві visited позначається як відвідана і активною стає та вершина, обчислений шлях з s в яку являється мінімальний. Припустимо, шлях з s в вершину u є коротшим, ніж з s в вершину t, отже, в такому випадку u міняє статус на активний. Далі, як і для вершини s досліджуються зв’язки з сусідами u, за винятком вершини s. Згодом, після виконання всіх обчислень, u позначається як пройдена, активною стає вершина t, і вся процедура повторюється для неї. Алгоритм Дейкстри триває до тих пір, поки всі доступні з s вершини не будуть досліджені.
3. Алгоритм пошуку A*.

Під час подорожі до певного пункту призначення, як правило, немає сенсу шукати шлях в протилежному напрямі. Тому з'явився алгоритм, який віддає перевагу вершинам, що знаходяться у напрямку до пункту призначення, і спершу відвідує їх, на відміну від алгоритму Дейкстри, який здійснює пошук у всіх напрямках простору. Харт, Нільссон і Рафаель вводять алгоритм A* [2], який додає евристику до алгоритму Дейкстри, роблячи його більш направленим до кінцевої вершини.

По суті, алгоритм A* виконує пошук по першому найкращому збігу на графі, тим самим знаходячи маршрут з найменшою вартістю від однієї вказаної вершини до іншої.

Порядок обходу вершин базується на використанні евристичної функції «відстань + вартість» (зазвичай її позначають як f (x)) [3]. Ця функція являє собою суму результатів двох інших: функції вартості на досягнення даної вершини (x) з початкової точки (позначається як g (x)) і евристичної оцінкою відстані від розглянутої вершини до кінцевої точки (позначається як h (x)).

Функція h (x) повинна базуватись на адекватному евристичному принципі, тобто не повинна переоцінювати відстані до цільової вершини. Наприклад, для завдання пошуку оптимального шляху h (x) може являти собою функцію розрахунку відстані до мети по прямій лінії, так як це фізично найменша можлива відстань між двома точками. Приклад результату задачі маршрутизації з використанням цього евристичного принципу зображено на рисунку 1.3.

Рисунок 1.3 - Візуалізація евристики алгоритму A*

Отже, A* досягає більш високої продуктивності (за часом) за допомогою евристики, проходячи меншу кількість вершин графа.

Кінцеву швидкість виконання обчислень можна додатково збільшити за рахунок використання двонаправленої версії алгоритму.

4. Геокодування.

Геокодування - це процес конвертації географічного ідентифікатора геопросторових об'єктів [4].

Для реалізації геокодування необхідний підготовлений набір метаданих для геопросторових об'єктів. До них відносяться широта і довгота, певні координати X і У, вулична адреса та інші описові характеристики об'єктів [15].

Як правило, масиви подібних геоданих зберігаються у вигляді просторових баз даних в яких виконано співставлення метаданих з координатною системою в рамках якої буде здійснюватися пошук.

У наш час у різних ГІС-платформах реалізовані функції адресного прив'язування даних з використанням файлів спеціального формату, у яких виконано формалізацію інформації вилученої з об'єктів вуличної мережі. Приклад такої мережі зображено на рисунку 1.4.

Рисунок 1.4 - Подання вуличної мережі у форматі OpenStreetMap

Щоб ефективно реалізувати роботу та збереження інформації, вуличну мережу міста розбивають на окремі квартальні відрізки, для кожного відрізка в базі даних зберігають його атрибути, такі як назва вулиці, номер будівлі біля початкової точки відрізка з правого боку, номер будівлі біля останньої точки відрізка з правого боку, номер будівлі біля початкової точки з лівого боку і номер будівлі біля кінцевої точки з лівого боку вулиці. Правий і лівий бік визначаються за парністю нумерування будівель на відрізку вулиці.

При геокодуванні адреси будинку, описаної назвою вулиці і номером будинку, знаходиться відрізок з необхідною назвою та інтервалом номерів будинків, далі на відповідній стороні знаходиться приблизне місце розташування будинку за різницею між номерами будинків на початку і кінці ділянки. Розміри будинків і можливі пропуски між ними в даному методі не враховуються.

Методами геокодування можна застосовувати для досить швидкого створення картографічних база даних з інформацією, що має текстове координатне прив'язування. Крім вуличних адресних координат, існують шаблони для створення об'єктів (точкових або площинних) за назвами міст і адміністративних одиниць, за кодами поштових округів та ін.

Процес перетворення опису об’єкта в його відображення на земній поверхні називається прямим геокодування. Результатом прямого геокодування є об’єкт або група об’єктів карти з атрибутами. Атрибути обов’язково включають в себе географічні координати. Так само часто, але не обов’язково, атрибути містять назву об’єкта, адресу, інформацію про адміністративне підпорядкування.

Зворотній процес - витяг описової інформації з об’єкта карти - називається зворотнім геокодування. Зазвичай результатом такого процесу є текстова інформація в заздалегідь обумовленому форматі.

Діапазон, для якого використовується геокодування, досить широкий. Однак часто під геокодуванням мають на увазі пошук географічних координат для заданої адреси або назви місця. Під зворотнім геокодування так само мають на увазі перетворенням географічних координат на адресу.

Як ми казали раніше, різні об’єкти можуть володіти різним набором описових характеристик. Об’єкти так само можуть бути частиною набору даних з обмеженою сферою застосування.

Машинний алгоритм, який виконує ці дії називається геокодером.

Геокодер приймає від користувача запит і повертає йому результат. В якості запиту можуть бути надані географічні координати, адреса, назва або будь-яка інша інформація. Залежно від запиту, відповідь буде представляти об’єкт карти (пряме геокодування) або текстовий опис цього об’єкта (зворотне).

Для зіставлення географічних координат і опису місцевості в процесі геокодування зазвичай використовуються бази даних. Такий підхід забезпечує зручне зберігання даних і швидкий доступ до них геокодером.

2. Постановка задачі

Виходячи з аналізу предметної області в системі буде лише одна роль користувача.

Програмний комплекс повинен мати змогу розраховувати маршрути між заданими географічними точками для легкового авто та трамваю. Також повинна бути забезпечена можливість виконувати операції геокодування.

Задачі маршрутизації та геокодування мають бути реалізовані на основі двох окремих програмних рішень, одне з яких необхідно розширити задля реалізації підтримки профілю маршрутизації для трамваїв, а для другого розробити та описати оптимальну схему конфігурації та розгортання.

Користувач має мати змогу прокладати маршрути використовуючи Web API. Аналогічним чином повинна відбуватись взаємодія з сервісом геокодування. Також повинна бути можливість взаємодії з сервісами та візуалізація результатів через простий веб інтерфейс. Результати виконання операцій через API мають бути в форматі JSON.

Інформація про конфігурація сервісу геокодування повинна бути доступна через Web API.

Прототип продукту повинен бути легким в розгортанні на серверах з операційними системами Linux та Windows.

3. Опис ключових варіантів використання

Система реалізовує функціональність розрахунку маршрутів та геокодування з допомогою взаємодії з розгорнутими сервісами через HTTP-протокол, тому серед актантів виділяється лише одна роль користувача, основними варіантами якого є розрахунок маршруту для легкового авто, розрахунок маршруту для трамваю, виконання операцій прямого та зворотнього геокодування відповідно до поставленого завдання (рисунок 1.5).

Рисунок 1.5 - Діаграма варіантів використання для основних функцій

У випадку виконання операцій через веб-інтерфейс присутня можливість візуалізувати результати. Також присутня можливість отримати інформація про конфігурація сервісу геокодування (рисунок 1.6).

Рисунок 1.6 - Діаграма варіантів використання для додаткових функцій

Таким чином визначивши варіанти використання для основних і додаткових функцій отримуємо діаграму варіантів використання для системи зображену на рисунку 1.7.

Рисунок 1.7 - Діаграма варіантів використання цілої системи

Всі варіанти використання для користувача та їх короткий опис поданий в таблиці 1.1.

Таблиця 1.1 - Опис прецедентів для актора «Користувач»

Прецедент	Короткий опис
Розрахунок маршруту для легкового авто	Дозволяє розраховувати маршрут для легкового авто
Розрахунок маршруту для трамваю	Дозволяє розраховувати маршрут для трамваю
Операція прямого геокодування	Дозволяє здійснювати перетворення опису об’єкта в його відображення на земній поверхні
Операція зворотнього геокодування	Дозволяє здійснювати витяг описової інформації з вказаного об’єкта карти
Візуалізація розрахованого маршруту	Дозволяє візуалізувати розрахований маршрут
Візуалізація результату операції геокодування	Дозволяє візуалізувати результат операції геокодування
Отримування конфігурації сервісу геокодування	Дозволяє отримувати інформацію про версію сервера, підтримувані функції та обмеження географічної області на якій відбуваються розрахунки

3. Проектування та конструювання програмної системи

4. Бібліотека маршрутизації

В якості бібліотеки маршрутизації буде використовуватися GraphHopper - це бібліотека маршрутизації та сервер із відкритим сирцевим кодом, написана на Java, який надає API маршрутизації через HTTP. Він працює на Linux сервері, настільному ПК, Android , iOS або Raspberry Pi . За замовчуванням використовуються дані OpenStreetMap як джерело даних про дорожньої мережі. Вбудований веб-інтерфейс зображено на рисунку 1.8.

Рисунок 1.8- Інтерфейс GraphHopper

GraphHopper сумісний з різними алгоритмами пошуку найкоротшого шляху, таких як Dijkstra, A * та його двонаправлені версії . Щоб зробити маршрутизацію досить швидкою для довгих шляхів (континентальний розмір) та уникнути евристичних підходів, GraphHopper використовує ієрархії скорочень за замовчуванням. У журналі журналу Java від Oracle автор Пітер Каріч описує методи, необхідні для ефективної та швидкої

оперативної пам'яті [5]. Крім того, сирцевий код GraphHopper добре покритий модульними, інтеграційними та навантажувальними тестами [6].
Щоб комп'ютер зрозумів реальний світ, використання графів як структури даних часто є найпростішим і природним вибором. Що стосується дорожніх мереж, то розв'язок задачі пошуку найкоротшого шляху - це набір вершин, з'єднаних вулицями, які є ребрами графу. На рисунку 1.9 зображено приклад фрагменту мапи, який був перетворений у граф.
Ліцензія Apache дозволяє кожному налаштувати і інтегрувати GraphHopper з власними безкоштовними або комерційними продуктами. Разом із високою швидкістю запитів і даних OpenStreetMap це робить GraphHopper можливою альтернативою існуючим службам маршрутизації і програмного забезпечення навігації GPS [7].

Рисунок 1.9 - Дорожній граф побудований з фрагменту мапи

Крім маршрутної маршрутизації для різних транспортних засобів, GraphHopper можна використовувати для обчислення матриць відстані, які потім використовуються як вхід для проблем маршрутизації транспортних засобів [8].

5. Бібліотека геокодування

За операції геокодування відповідатиме Nominatim - інструмент для пошуку даних OSM по текстовому опису об’єкту, а також для пошуку повних адрес по заданим координатам (зворотне геокодування). Nominatim також використовується як одне із джерел в пошуковій формі на головній сторінці OpenStreetMap і покращує пошук на сайті MapQuest Open Initiative. Так само деякі кампанії надають встановлені сервіси Nominatim, звернення до яких відбувається за певним API.

6. Вибір методу та середовища розгортання системи

Docker відкрита платформа для розробки, публікації, і запуску додатків. Дана платформа дає можливість відокремити програми від інфраструктури, щоб можливо було швидко зробити розгортання. З Docker можливо керувати інфраструктурою так само просто, як і додатками.
Докер забезпечує можливість упаковки і запуску додатку в ізольованому середовищі званим контейнером. Ізоляція і необхідний рівень безпеки дозволяють запускати безліч контейнерів на заданому хості. Через легкий характер контейнерів, які працюють без додаткового навантаження гіпервізора, можна запустити більше контейнерів, ніж на віртуальній машині
Сама платформа називається Docker Engine, яка представляє клієнт-серверний застосунок з трьома головними компонентами (рисунок 1.10):
1. Сервер працює у фоновому режимі (демон).
2. REST API, який використовують програми для взаємодії з сервером.
3. Інтерфейс командного рядка (CLI) клієнт.

CLI використовує Docker REST API для керування або взаємодії з демоном Докер за допомогою сценаріїв або безпосередньо команд CLI. Багато інших додатків Docker засновані на використанні API і CLI.
Демон створює і управляє об'єктами Docker, такими як образи, контейнери, мережі та сховища даних.
1. Архітектура платформи Docker Engine.
Docker використовує архітектуру клієнт-сервер (рисунок 1.10).

Рисунок 1.10 - Склад платформи Docker
Docker клієнт спілкується з демоном Docker, який бере на себе створення, запуск, розподіл контейнерів. Обидва, клієнт і сервер можуть працювати на одній системі, також можна підключити клієнт до віддаленого демона docker. Клієнт і сервер спілкуються через сокет або через RESTful API.
Як показано на рисунку 1.11, демон запускається на хост-машині. Користувач не взаємодіє з сервером на пряму, а використовує для цього клієнт. Docker-клієнт - головний інтерфейс до Docker системи. Він отримує команди від користувача і взаємодіє з docker-демоном. Демон Docker працює на хост-машині. Користувач використовує клієнт Docker для взаємодії з демоном.
2. Головні компоненти платформи Docker Engine.
Щоб розуміти, як працює docker, потрібно знати про три основні його компоненти:
1. образи (images)
2. реєстр (registries)
3. контейнери
Docker-образ - це read-only шаблон з набором інструкцій для створення контейнера. Наприклад, образ може містити операційну систему Ubuntu з веб-сервером NGINX і додатком на ній.

Рисунок 1.11 - Архітектура платформи Docker
Образи використовуються для створення контейнерів. Docker дозволяє створювати нові образи, оновлювати існуючі, або завантажувати і використовувати образи, які були створені іншими користувачами.
Реєстр - це бібліотека образів. Він може бути публічним або приватним і може розташовуватися на одному сервері з демоном, клієнтом або на окремому сервері. Реєстри - це компонента поширення.
Контейнери схожі на директорії. У контейнерах міститься все, що потрібно для роботи програми. Кожен контейнер створюється з образу. Контейнери можуть бути створені, запущені, зупинені або видалені. Кожен контейнер ізольований і є безпечною платформою для додатка. Контейнери - це компонента роботи.
Docker сервіс надає режим swarm, за допомогою якого можна обмежувати кількість примірників процесів образу. Можна вказати кількість паралельних

завдань реплік для запуску, і менеджер swarm гарантує, що навантаження розподілиться рівномірно між робочими вузлами.

3. Переваги контейнеризації.

Контейнери несуть в собі багато привабливих переваг як для розробників, так і для системних адміністраторів. В першу чергу важливим є порівняння контейнеризації з віртуальними машинами (рисунок 1.12)

Деякі з найбільш привабливих переваг перераховані нижче.

1. Абстрагування хост-системи від контейнеризованих додатків. Контейнери були задуманими повністю стандартизованими. Це означає, що контейнер з'єднується з хостом або чим-небудь зовнішнім по відношенню до нього, за допомогою певних стандартизованих інтерфейсів. Контейнеризований застосунок не повинен покладатися або якимось чином залежати від ресурсів або архітектури хоста, на якому він працює.

2. Простота масштабування. Одним з переваг абстрагування між операційною системою хоста і контейнерами є те, що при правильному проектуванні програми, масштабування може бути простим і прямолінійним. Сервіс-орієнтована архітектура в комбінації з контейнеризованими додатками забезпечує основу для легкого масштабування.

3. Простота управління залежностями і версіями додатка. Контейнери дозволяють розробнику програми або компоненту додатку, з усіма його залежностями і далі працювати з ними як з єдиним цілим. Хосту не треба турбуватися про залежності, необхідні для запуску конкретного додатку. Якщо хост може запустити Docker, він може запустити будь-який Docker-контейнер.

4. Надзвичайно легкі, ізольовані середовища виконання. Контейнери ізольовані на рівні процесів, працюючи при цьому поверх одного і того ж ядра хосту. Це означає, що контейнер не включає в себе повну операційну систему, що призводить до практично миттєвого його запуску.

Рисунок 1.12 - Схематичне порівняння архітектури віртуальних машин та контейнерів

5. Спільно використовувані шари. Контейнери легкі ще і в тому сенсі, що вони зберігаються «пошарово». Якщо кілька контейнерів засновані на одному і тому ж шарі, вони можуть спільно використовувати цей базовий шар без дублювання, що призводить до мінімального завантаження дискового простору в наступних образах.

6. Можливість компонування та передбачуваність. Docker-файли дозволяють користувачам задати конкретні дії, необхідні для створення нового образу контейнера. Це дозволяє задавати налаштування середовища виконання так, як нібито це код, при бажанні, зберігаючи ці налаштування в системі контролю версій. Однакові Docker-файл, зібрані в одному і тому ж оточенні, завжди створюють ідентичні образи контейнеру.

Docker базується на технологіях namespaces і cgroups (перша забезпечує ізоляцію, друга - угруповання процесів і обмеження ресурсів), тому в плані віртуалізації він мало чим відрізняється від звичних нам LXC / OpenVZ. Та ж швидкість роботи, ті ж методи ізоляції, засновані на механізмах ядра Linux, але він надає зручне API для взаємодії з ними, що дозволяє розгорнути повноцінне віртуальне оточення і запустити в ньому.

7. Побудова UML-діаграми класів

Архітектура GraphHopper передбачає для кожного унікального типу транспортного засобу свій окремий профіль маршрутизації, який буде використано для побудови стандартизованого зваженого графу транспортної мережі (див. рисунок 1.9).

Кожен профіль являє собою клас-спадкоємець єдиного для всіх транспортних засобів базового абстрактного класу AbstractFlagEncoder (рисунок 1.13).

Рисунок 1.13 - Структура базового абстрактного класу AbstractFlagEncoder

В цьому класі (див. рисунок 1.13) визначено усі необхідні поля та методи для імплементації унікальної логіки інтерпретації атрибутів картографічних об’єктів в сумісний з алгоритмом маршрутизації формат.

Також слід зазначити що клас AbstractFlagEncoder реалізовує інтерфейс FlagEncoder, структура якого зображена на рисунку 1.14.

Рисунок 1.14 - Структура інтерфейсу FlagEncoder

Цей інтерфейс (див. рисунок 1.14) служить для опису базових методів які необхідні для фільтрації та конвертації атрибутів вхідних географічних об’єктів для подальшого використання отриманих значень в побудові зваженого графу. Він розширює два інших інтерфейсів: TurnCostEncoder (рисунок 1.15) та EncodedValueLookup (рисунок 1.16).

Рисунок 1.15 - Структура інтерфейсу TurnCostEncoder

Інтерфейс TurnCostEncoder (див. рисунок 1.15) описує методи які необхідні для алгоритму маршрутизації, а саме частини відповідальної за логіку обробки поворотів та розворотів.

Рисунок 1.16 - Структура інтерфейсу EncodedValueLookup

Діаграма подана вище (див. рисунок 1.16) описує базові операції зчитування атрибутів вхідних картографічних об’єктів.

Загалом структура інтерфейсу FlagEncoder разом з іншими інтерфейсами які він розширює зображена на рисунку 1.17.

Рисунок 1.17 - Діаграма інтерфейсу FlagEncoder та інтерфейсів які він розширює

Склавши разом всі вищеописані діаграми отримаємо загальну діаграму ієрархії класів та інтерфейсів які задіяні в реалізації нового профілю маршрутизації.

Рисунок 1.18 - Загальна діаграма ієрархії класів та інтерфейсів

Зважаючи на ієрархію залежностей між класами та інтерфейсами (див. рисунок 1.18), можна спроектувати структуру класу TramFlagEncoder (рисунок 1.19).

Рисунок 1.19 - Структура класу TramFlagEncoder

Загалом, реалізація підтримки профілю маршрутизації для трамваю являє собою написання нового класу TramFlagEncoder (див. рисунок 1.19) який повинен реалізувати та перевизначити методи базового абстрактного класу AbstractFlagEncoder (див. рисунок 1.13).
Ці методи описують логіку необхідну для роботи функції маршрутизації яка базується на алгоритмах пошуку оптимального шляху між вузлами зваженого графу. Саме за трансформацію атрибутів вхідних географічних об’єктів в вагові коефіцієнти ребер зваженого графу відповідає більша частина реалізованих та перевизначених методів.
Для коректної роботи трамвайної маршрутизації необхідно описати логіку відбору геометрії по якій дозволено прокладати маршрут (трамвайні колії), обробку швидкісних обмежень, прийняття рішень на поворотах та розворотах, односторонній рух та опис об’єктів які являються бар’єрами для руху транспорту.

8. Реалізація основних класів та методів

При першому старті сервісу GraphHopper відбувається конвертація вхідних картографічних даних в спеціалізований формат графового представлення транспортної мережі.
Трамвай може рухатись лише по трамвайних коліях. Саме тому зважений граф трамвайної транспортної мережі буде складись лише з геометрії трамвайних колій.
Фільтрація та відбір об’єктів з підтримуваними атрибутами відбувається методом який на вході отримує декодовані картографічні об’єкти зі всіма атрибутами та зв’язками (лістинг 1.1).

Лістинг 1.1 - Метод відбору об’єктів трамвайних колій

```
@Override
public EncodingManager.Access getAccess(ReaderWay way) {
   String railwayValue = way.getTag("railway");
   if((railwayValue != null) && ("tram".equals(railwayValue))) {
         return EncodingManager.Access.WAY;
      }
   return EncodingManager.Access.CAN_SKIP;
}
```

Для визначення присутності одностороннього руху на певному вхідному проміжку шляху використовуються методи описані в лістингу 1.2.

Лістинг 1.2 - Односторонній рух

```
protected boolean isBackwardOneway(ReaderWay way) {
   return way.hasTag("oneway", "-1");
}
protected boolean isForwardOneway(ReaderWay way) {
   return !way.hasTag("oneway", "-1");
}
protected boolean isOneway(ReaderWay way) {
   return way.hasTag("oneway", oneways);
}
```

Для визначення швидкісного обмеження написано метод наведений в лістингу 1.3.

Лістинг 1.3 - Метод визначення швидкісного обмеження

```
protected double getSpeed(ReaderWay way) {
   String railwayValue = way.getTag("railway");
   Integer speed = defaultSpeedMap.get(railwayValue);
   return speed;
}
```

Окремі картографічні об’єкти об’єднані між собою за рахунок атрибутів зв’язків. Логіку роботи з цими зв’язками обов’язково потрібно описати реалізувавши спеціальний метод (лістинг 1.4).

Лістинг 1.4 - Метод обробки зв’язків між картографічними об’єктами

```
@Override
public long handleRelationTags(long oldRelationFlags,
                        ReaderRelation relation) {
   return oldRelationFlags;
}
```

На цьому етапі стає доступним реалізація методу генерації об’єктів дуг зваженого графу транспортної мережі (лістинг 1.5).

Лістинг 1.5 - handleWayTags

```
@Override
public IntsRef handleWayTags(IntsRef edgeFlags, ReaderWay way,
                             EncodingManager.Access accept,
                             long relationFlags) {
   if (accept.canSkip())
      return edgeFlags;

   double speed = getSpeed(way);

   setSpeed(false, edgeFlags, speed);
   setSpeed(true, edgeFlags, speed);

   boolean isRoundabout = roundaboutEnc
         .getBool(false, edgeFlags);
   if (isOneway(way) || isRoundabout) {
      if (isForwardOneway(way))
         accessEnc.setBool(false, edgeFlags, true);
      if (isBackwardOneway(way))
         accessEnc.setBool(true, edgeFlags, true);
   } else {
      accessEnc.setBool(false, edgeFlags, true);
      accessEnc.setBool(true, edgeFlags, true);
   }

   return edgeFlags;
}
```

Наступним кроком є реалізація методу який описує формат кодування швидкісних обмежень (лістинг 1.6).

Лістинг 1.6 - Опис формату кодування швидкісних обмежень

```
@Override
public void createEncodedValues(
      List<EncodedValue> registerNewEncodedValue,
      String prefix, int index) {
   super.createEncodedValues(registerNewEncodedValue,
         prefix, index);
   registerNewEncodedValue.add(speedEncoder = new
         FactorizedDecimalEncodedValue(prefix
         + "average_speed", speedBits,
         speedFactor, false));
```

```
}
```

За ідентифікацію версії реалізації, назви профілю, формату деталізації кодування даних відповідають методи описані в лістингу 1.7.

Лістинг 1.7 - Реалізація службових методів TramFlagEncoder

```
@Override
public int defineTurnBits(int index, int shift) {
   return shift;
}
@Override
public boolean isTurnRestricted(long flags) {
   return false;
}
@Override
public long getTurnFlags(boolean restricted, double costs) {
   return 0;
}
@Override
public int getVersion() {
   return 1;
}
@Override
public String toString() {
   return "tram";
}
```

Завершальним етапом написання класу TramFlagEncoder є опис його конструкторів. При створенні екземпляру класу в першу чергу викликається конструктор описаний в лістингу 1.8.

Лістинг 1.8 - Головний конструктор TramFlagEncoder

```
public TramFlagEncoder(PMap properties) {
this((int) properties.getLong("speed_bits", 5),
       properties.getDouble("speed_factor", 5),
       properties.getBool("turn_costs", false) ? 1 : 0);
this.properties = properties;
this.setBlockFords(properties.getBool("block_fords", true));
this.setBlockByDefault(properties.getBool("block_barriers", true));
}
```

Конфігурація деяких додаткових параметрів та виклик конструктора описаного в базовому абстрактному класі описано через допоміжний конструктор (лістинг 1.9).

Лістинг 1.9 - Допоміжний конструктор TramFlagEncoder

```
public TramFlagEncoder(int speedBits,
                       double speedFactor,
                       int maxTurnCosts) {
   super(speedBits, speedFactor, maxTurnCosts);
   maxPossibleSpeed = 16;
   defaultSpeedMap.put("tram", 16);
   init();
}
```

Щоб новий клас став доступний для використання його ім’я необхідно додати до інтерфейсу фабрики створення об’єктів (лістинг 1.10).

Лістинг 1.10 - FlagEncoderFactory

```
public interface FlagEncoderFactory {
   /**
    * Ініціалізація констант з найменуваннями інших
    * профілів маршрутизації (CAR, FOOT, BIKE ...)
    */
   final String TRAM = "tram";

   final FlagEncoderFactory DEFAULT = new DefaultFlagEncoderFactory();
   FlagEncoder createFlagEncoder(String name, PMap configuration);
}
```

Фінальним етапом буде розширення коду класу-фабрики яка фактично створює об’єкти профілів при старті сервісу маршрутизації (лістинг 1.11).

Лістинг 1.11 - DefaultFlagEncoderFactory

```
public class DefaultFlagEncoderFactory implements FlagEncoderFactory {
   @Override
   public FlagEncoder createFlagEncoder(String name, PMap configuration) {
      // інші профілі маршрутизації (CAR, FOOT, BIKE ...)

      if (name.equals(TRAM))
         return new TramFlagEncoder(configuration);

      throw new IllegalArgumentException(
            "entry in encoder list not supported " + name);
   }
}
```

4. Використання програмної системи

9. Розгортання програмної системи та системні вимоги

Для роботи сервісів необхідно спочатку їх розгорнути. Кожен сервер запускатиметься у вигляді окремого Docker-контейнера. Діаграма розгортання зображена на рисунку 1.20.

Рисунок 1.20 - Діаграма розгортання

В першу чергу необхідно в директорії з проектами помістити картографічні дані необхідного фрагменту мапи. Завантаження актуальних даних можна здійснити з допомогою сервісу Overpass API. Зона для експорту являє собою прямокутник описаний двома координатами (див. рисунок 1.21).

Рисунок 1.21 - Експорт даних фрагменту мапи

Для завантаження можна скористатись утилітою cURL, приклад використання якої зображено в лістингу 1.12.

Лістинг 1.12 - Завантаження фрагмента мапи утилітою cURL

```
curl -o ./map_fragment.osm \
overpass-api.de/api/map?bbox=23.8657,49.7391,24.1933,49.9512
```

Для запуску серверу маршрутизації GraphHopper необхідно перейти в директорію з проектом, виконати побудову Docker-образу та запустити контейнер на його основі. Приклад запуску з профілем маршрутизації для легкового авто зображено в лістингу 1.13.

Лістинг 1.13 - Збірка та запуск GraphHopper для легкового авто

```
docker build -t graphhopper-fork:car .
docker run -d --name gh-car -p 9090:8989 \
        graphhopper-fork:car \
        /graphhopper/map_fragment.osm
```

Перед запуском версії сервера з профілем маршрутизації для трамваю доцільно виконати фільтрацію картографічних даних. Такий підхід обумовлений специфікою роботи профілю для функціонування якого достатньо даних пов’язаних з геометрією трамвайних колій. Саме тому немає потреби збільшувати розмір кінцевого образу за рахунок надлишкових даних. Відокремлення необхідних географічних об’єктів можна здійснити утилітою osmfilter, використання якої зображено в лістингу 1.14.

Лістинг 1.14 - Фільтрація даних пов’язаних із трамвайними коліями

```
osmfilter map_fragment.osm \
        --keep="railway=tram" \
        -o=tram_network.osm
```

Так як задача розрахунку маршруту для рейкового транспорту є досить вузькоспеціалізованою, експортовані географічні дані зазвичай містять в собі певний набір недоліків. Найбільш розповсюдженими є відсутність атрибутів наявності одностороннього руху та грубі невідповідності в геометрії.

Виправити помилки можна шляхом безпосереднього редагування та публікації правок в картографічному сервісі OpenStreetMap та повторним експортом уже коректних даних. Другим варіантом є модифікація локального файлу утилітою JOSM (див. рисунок 1.22). Цей варіант є більш надійнішим за рахунок унеможливлення внесення правок в дані іншими учасниками спільноти.

Рисунок 1.22 - Редагування даних утилітою JOSM

Маючи коректний файл з даними лишилось лиш виконати збірку образу та запустити на його основі Docker-контейнер. Приклад послідовності команд для запуску сервісу зображено в лістингу 1.14.

Лістинг 1.14 - GraphHopper

```
docker build -t graphhopper-fork:tram .
docker run -d --name gh-tram -p 9090:8989 \
        graphhopper-fork:tram \
        /graphhopper/tram_network.osm
```

Наступним кроком є запуск серверу геокодування. З огляду на рекомендації описані в документації Nominatim для розгортання буде використано готовий образ останньої доступної версії проекту. Образ опублікований та доступний в онлайн сервісі Docker Hub.

Картографічні дані, які будуть використовуватися для імпорту, повинні відповідати підтримуваному формату. Приклад конвертації з допомогою утиліти osmconvert зображено в лістингу 1.15.

Лістинг 1.15 - Конвертування даних фрагменту простору

```
osmconvert map_fragment.osm \
        -o=map_fragment.pbf
```

Перший запуск серверу геокодування відбувається в два кроки. Спочатку необхідно виконати імпорт картографічних даних в реляційну базу даних PostgreSQL. Приклад виконання цієї процедури зображено в лістингу 1.16.

Лістинг 1.16 - Імпорт просторових даних в базу даних

```
docker run -t -v ~/nominatimdata:/data \
 mediagis/nominatim:latest \
        sh /app/init.sh /data/map_fragment.pbf postgresdata 4
```

Після успішного імпорту лишилось виконати запуск контейнеру з сервісом (див. лістинг 1.17)
Лістинг 1.17 - Запуск Nominatim
```
docker run --restart=always -p 7070:8080 -d --name nominatim \
-v ~/nominatimdata/postgresdata:/var/lib/postgresql/11/main \
mediagis/nominatim:latest bash /app/start.sh
```

10. Опис використання системи

Основним способом роботи з розгорнутими сервісами є взаємодія через Web API. Кожен сервер доступний через окремий унікальний виділений порт (див. рисунок 1.20).
Функціональність прямого та зворотнього геокодування доступна через кінцеві точки API серверу Nominatim. Присутня можливість налаштування формату виводу даних (xml, json, jsonv, geojson, geocodejson). Список доступних дій та їхні основі параметри наведено в таблиці 1.1.

Таблиця 1.1 - Кінцеві точки для запиту даних сервісу геокодування

Метод	Шлях	Дія
GET	/search?format=json&q=<адреса>	Пошук об'єктів OSM за назвою чи типом
GET	/reverse?format=json&lat=<широта>&lon=<довгота>	Пошук об’єкта OSM за їх місцезнаходженням (зворотне геокодування)
GET	/lookup?osm_ids=<унікальний ідентифікатор>&format=json	Пошук деталей об’єктів OSM за їх унікальним ідентифікатором
GET	/status	Повертає статус роботи сервера

Типовий результат виконання операції геокодування являє собою масив знайдених картографічних об’єктів (див. рисунок 1.23).

Рисунок 1.23 - Результат виконання операції зворотнього геокодування

Також є можливість здійснювати операції через веб-інтерфейс. Пряме геокодування передбачає ввід адреси або назви об’єкту в поле пошуку. В свою чергу для зворотнього геокодування необхідно задати географічні координати та бажану глибину пошуку. Приклад пошуку адреси за вказаним місцезнаходження зображено на рисунку 1.24.

Рисунок 1.24- Зворотне геокодування через веб-інтерфейс

Зліва від мапи відображаються результати пошуку. Для перегляду розширеного опису знайденого об’єкту необхідно натиснути на копку «details» (див. рисунок 1.25).

Рисунок 1.25 - Редактор JOSM

Аналогічним способом відбувається взаємодія з сервісами маршрутизації. Якщо виникає потреба прокласти маршрут через API, необхідно виконати запит через кінцеву точку сервісу GraphHopper, опис параметрів якої наведено в таблиці 1.2.

Таблиця 1.2 - Кінцеві точки сервісу маршрутизації

Метод	Шлях	Дія
GET	/route?point=<маршрутна точка>&point=<маршрутна точка>&type=json&locale=uk&vehicle=<профіль>&points_encoded=false	Прокладання маршруту через довільну кількість маршрутних точок

Приклад успішного API запиту прокладання маршруту для трамваю зображено на рисунку 1.26

Рисунок 1.26 - Розрахунок маршруту для трамваю через API

Також підтримується взаємодія через веб-інтерфейс, приклад використання якого зображено на рисунку 1.27.

Рисунок 1.27 - Розрахунок маршруту для трамваю через веб-інтерфейс

11. Верифікація програмної системи

Верифікація програмної системи - це визначення відповідності програми до висунутих їй вимог.

Наш програмна система може розраховувати маршрути між заданими географічними точками для легкового авто та трамваю. Також доступна можливість виконувати операції геокодування.

Задачі маршрутизації та геокодування реалізовані на основі двох окремих програмних рішень, одне з яких детально досліджено та розширено задля підтримки профілю маршрутизації для трамваїв, а для другого розроблено та описано оптимальну схему конфігурації та розгортання.

Користувач має можливість прокладати маршрути використовуючи Web API. Аналогічним чином відбувається взаємодія з сервісом геокодування. Також доступна можливість взаємодії з сервісами через простий веб інтерфейс. Результати виконання операцій через API представляються в форматі JSON.

Інформація про конфігурація сервісу геокодування через Web API.

Реалізована та описана процедура розгортання сервісів на серверах з операційними системами Linux та Windows.

Отже, програмна система повністю відповідає поставленим їй вимогам.

1. Тестування програмної системи

5. План тестування

Тестуватися буде розроблюваний сервіс побудови маршрутів та прямо-зворотнього геокодування, , який являє собою дві окремі програмних системи, які дозволять здійснювати прокладку маршруту через вказані географічні точки та виконувати операції геокодування.

Для перевірки коректності роботи програмного забезпечення будуть використовуватися функціональні тести, які будуть перевіряти відповідність роботи поставленим завданням.
Таким чином тестуватися будуть такі функції:
1. розрахунок маршруту для трамваю через Web API;
2. розрахунок маршруту для легкового авто через Web API;
3. операція прямого геокодування через Web API;
4. операція зворотнього геокодування через Web API;
5. розрахунок маршруту та візуалізація результату для трамваю через веб-інтерфейс;
6. розрахунок маршруту та візуалізація результату для легкового авто через веб-інтерфейс;
7. операція прямого геокодування та візуалізація результату через веб-інтерфейс;
8. операція зворотнього геокодування та візуалізація результату через веб-інтерфейс;
9. отримування конфігурації сервісу геокодування через Web API.
Тестування буде проводитися для двох способів взаємодії з сервісом: за допомогою Web API та Web інтерфейсу. Для тестування Web API буде використовуватися Postman - це інструмент для тестування API.

6. Розробка тестів

Розроблені тести з результатами виконання зображені в таблиці 2.1.

Таблиця 2.1 - Розроблені тести з результатами виконання

Назва тесту	Передумови	Кроки виконання	Очікуваний результат	Пройдено
Розрахунок маршруту для трамваю через Web API	Відкрито Postman	Змінити значення перемикача типу запиту на «GET» В адресному рядку вказати адресу сервісу з трамвайним профілем маршрутизації Доповнити адресу найменуванням кінцевої точки «/route» Вказати географічними координатами початкової точки маршруту в		

форматі WGS 84 через параметр «point» Вказати географічними координатами кінцевої точки маршруту в форматі WGS 84 через параметр «point» Вказати «json» як значення параметру «type» Вказати «uk» як значення параметру «locale» Вказати «tram» як значення параметру «vehicle» Вказати «false» як значення параметру «points_encoded» Натиснути кнопку Send Повернуто json об’єкт в якому міститься масив «path» +

Продовження таблиці 2.1

Назва тесту	Передумови	Кроки виконання	Очікуваний результат	Пройдено
Розрахунок маршруту для легкового авто через Web API	Відкрито Postman	Змінити значення перемикача типу запиту на «GET» В адресному рядку вказати адресу сервісу з трамвайним профілем маршрутизації Доповнити адресу найменуванням кінцевої точки «/route» Вказати географічними координатами початкової точки маршруту в форматі WGS 84 через параметр «point» Вказати географічними координатами кінцевої точки маршруту в форматі WGS 84 через параметр «point» Вказати «json» як значення параметру «type» Вказати «uk» як значення параметру «locale» Вказати "tram" як значення параметру «car» Вказати «false» як значення параметру «points_encoded» Натиснути кнопку Send	Повернуто json об’єкт в якому міститься масив «path»	+
Отримання конфігурації сервісу геокодува-ння через Web API	Відкрито Postman	Змінити значення перемикача типу запиту на «GET» В адресному рядку вказати адресу сервісу геокодування Доповнити адресу найменуванням кінцевої точки «/status» Вказати «json» як значення параметру «format» Натиснути кнопку Send	Повернуто json об’єкт з деталями конфігурації сервера	+

Продовження таблиці 2.1

Назва тесту	Передумови	Кроки виконання	Очікуваний результат	Пройдено
Розрахунок маршруту та візуалізація результату для трамваю через Web інтерфейс	Відкрита головна веб сторінка сервісу з трамвайним профілем маршрутизації	Натиснути праву кнопку миші в межах області мапи в початковій точці маршруту Обрати пункт «Встановити початок» Натиснути праву кнопку миші в межах області мапи в кінцевій точці маршруту Обрати пункт «Встановити кінець»	Між заданими точками розраховано та візуалізовано на мапі маршрут у вигляді полілінії	+

Розрахунок маршруту та візуалізація результату для легкового авто через Web інтерфейс Відкрита головна веб сторінка сервісу з профілем маршрутизації для легкового авто Натиснути праву кнопку миші в межах області мапи в початковій точці маршруту Обрати пункт «Встановити початок» Натиснути праву кнопку миші в межах області мапи в кінцевій точці маршруту Обрати пункт «Встановити кінець»Між заданими точками розраховано та візуалізовано на мапі маршрут у вигляді полілінії +
Операція прямого геокодування та візуалізація результату через Web інтерфейс Відкрита головна веб сторінка сервісу геокодування В пошуковий рядок ввести коректну поштову адресу шуканого географічного об'єкту Натиснути кнопку «Search» Серед списку знайдених географічних об'єктів виділіть найбільш релевантний натисканням лівої кнопки миші На виділеному результаті пошуку натисніть кнопку «details» Результат пошуку містить шуканий географічний об'єкт з детальним описом його OSM атрибутів +

Продовження таблиці 2.1

Назва тесту	Передумови	Кроки виконання	Очікуваний результат	Пройдено
Операція прямого геокодування через Web API	Відкрито Postman	Змінити значення перемикача типу запиту на «GET» В адресному рядку вказати адресу сервісу геокодування Доповнити адресу найменуванням кінцевої точки «/search» Вказати адресу або опис шуканого географічного об'єкту через параметр «q» Вказати «json» як значення параметру «format» Натиснути кнопку Send	Повернуто json об’єкт в якому міститься не пустий масив з географічними об'єктами в форматі OSM	+
Операція зворотнього геокодування через Web API	Відкрито Postman	Змінити значення перемикача типу запиту на «GET» В адресному рядку вказати адресу сервісу геокодування Доповнити адресу найменуванням кінцевої точки «/reverse» Вказати широту шуканої географічної точки в форматі WGS 84 через параметр «lat»Вказати довготу шуканої географічної точки в форматі WGS 84 через параметр "lon" Вказати «json» як значення параметру «format»Натиснути кнопку Send	Повернуто json об’єкт, в якому міститься не пустий масив з географічними об'єктами в форматі OSM	+

Список використаних джерел

1. A Hub-Based Labeling Algorithm for Shortest Paths on Road Networks - [Electronic resource] - Access date: 10.12.2019. - Mode of access: https://www.microsoft.com/en-us/research/publication/a-hub-based-labeling-algorithm-for-shortest-paths-on-road-networks/?from=http%3A%2F%2Fresearch.microsoft.com%2Fpubs%2F142356%2Fhl-tr.pdf.
2. Рассел С. Дж., Норвиг, П. Искусственный интеллект: современный подход = Artificial Intelligence: A Modern Approach [Текст]. / Пер. с англ. и ред. К. А. Птицына. - 2-е изд. - М.: Вильямс, 2006. - с. 157-162.
3. Лорьер Ж.-Л. Системы искусственного интеллекта [Текст]. / Пер. с фр. и ред. В. Л. Стефанюка. - М.: Мир, 1991. - с. 238-244.
4. Что такое геокодирование? [Электронный ресурс]. - Дата доступа: 10.12.2019. - Режим доступа: https://pro.arcgis.com/ru/pro-app/help/data/geocoding/introduction-to-finding-places-on-a-map.htm.
5. Java Magazine - January/February 2014, GraphHopper Maps: Fast Road Routing in 100% Java - [Electronic resource] - Access date: 10.12.2019. - Mode of access: http://ictblog.luisalbertogh.net/docus/javamagazine/javamagazine20140102-dl.pdf.
6. Public Travic CI: showing large test suite of GraphHopper - [Electronic resource] - Access date: 10.12.2019. - Mode of access: https://travis-ci.org/graphhopper/graphhopper.
7. Методичні рекомендації по виконанню розділу техніко-економічного обґрунтування дипломних робіт студентами технічних спеціальностей напряму підготовки 8.05010302 «Інженерія програмного забезпечення» освітньо-кваліфікаційного рівня «Магістр» / Укладачі: Петрик М.Р., Михалик Д.М., Кінах Я.І., Гладьо С.В., Цуприк Г.Б. - Тернопіль: Вид-во ТНТУ імені Івана Пулюя, 2016 - 28 с.
8. Закон України «Про збір та облік єдиного внеску на загальнообов'язкове державне соціальне страхування» №2464-VI від 08.07.2010.
9. Податковий кодекс України, п.164.6 ст.164.
10. Закон України «Про розповсюдження примірників аудіовізуальних творів фонограм, відеограм, комп'ютерних програм, баз даних» №1587 від 01.01.2013.
11. Постанова Кабінету Міністрів України «Про затвердження Порядку визначення класу професійного ризику виробництва за видами економічної діяльності» № 237 від 8.02.2012.
12. Джерела, зони дії та рівні забруднень навколишнього середовища у разі аварій на АЕС і хімічно небезпечних об'єктах [Електронний ресурс]. - Дата доступу: 10.12.2019. - Режим доступу: http://buklib.net/books/30227.
13. Забруднення повітря на робочих місцях із ВДТ [Електронний ресурс]. - Дата доступу: 10.12.2019. - Режим доступу: https://www.kazedu.kz/ref/98015/12.
14. Хижняк Ю. Д. Обзор наиболее популярных картографических сервисов, предоставляющих api для разработчиков / Ю. Д. Хижняк // Научный журнал NovaInfo.Ru. - 2017.

15. Геокодування [Електронний ресурс]. - Дата доступу: 10.12.2019.. - Режим доступу: https://studfile.net/preview/7328519/page:31

Додаток А
Технічне завдання

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
Факультет комп’ютерно-інформаційних систем і програмної інженерії
Кафедра програмної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедрою
програмної інженерії
д.ф.-м.н., проф., Петрик М.Р.
“___ “_____ 2019 р.

Технічне завдання
на виконання дипломної роботи
«Сервіс побудови маршрутів та прямого-зворотнього геокодування на базі "Open street map"»

Керівник роботи:
к. т. н., доцент Михалик Дмитро Михайлович
“___” _____ 2019р.
_____

Виконавець:
студент групи СПм-61
Кузьмич Олександр Петрович
“___” _____ 2019р.
_____

м. Тернопіль - 2019

Додаток Б
Текст програмної системи

Лістинг Б.1 - Код FlagEncoderFactory FlagEncoderFactory.java
public interface FlagEncoderFactory {

```
    final String TRAM = "tram";
    final FlagEncoderFactory DEFAULT = new DefaultFlagEncoderFactory();
    FlagEncoder createFlagEncoder(String name, PMap configuration);
}
```

Лістинг Б.2 - Код createFlagEncoder DefaultFlagEncoderFactory.java

```
@Override
public FlagEncoder createFlagEncoder(String name, PMap configuration) {
    if (name.equals(TRAM))
        return new TramFlagEncoder(configuration);
    throw new IllegalArgumentException("entry in encoder list not supported " +
name);
}
```

Лістинг Б.3 - Код TramFlagEncoder TramFlagEncoder.java

```
public TramFlagEncoder(PMap properties) {
    this((int) properties.getLong("speed_bits", 5),
            properties.getDouble("speed_factor", 5),
            properties.getBool("turn_costs", false) ? 1 : 0);
    this.properties = properties;
    this.setBlockFords(properties.getBool("block_fords", true));
    this.setBlockByDefault(properties.getBool("block_barriers", true));
}
```

Лістинг Б.4 - Код TramFlagEncoder TramFlagEncoder.java

```
public TramFlagEncoder(int speedBits,
                       double speedFactor,
                       int maxTurnCosts) {
    super(speedBits, speedFactor, maxTurnCosts);
    maxPossibleSpeed = 16;
    defaultSpeedMap.put("tram", 16);
    init();
}
```

Лістинг Б.5 - Код getAccess TramFlagEncoder.java

```
@Override
public EncodingManager.Access getAccess(ReaderWay way) {
    String railwayValue = way.getTag("railway");
    if ((railwayValue != null) && ("tram".equals(railwayValue))) {
            return EncodingManager.Access.WAY;
        }
    return EncodingManager.Access.CAN_SKIP;
}
```

Лістинг Б.6 - Код getSpeed TramFlagEncoder.java

```
protected double getSpeed(ReaderWay way) {
   String railwayValue = way.getTag("railway");
   Integer speed = defaultSpeedMap.get(railwayValue);
   return speed;
}
```

Лістинг Б.7 - Код handleRelationTags TramFlagEncoder.java

```
public long handleRelationTags(long oldRelationFlags, ReaderRelation relation) {
   return oldRelationFlags;
}
```

Лістинг Б.8 - Код handleWayTags TramFlagEncoder.java

```
@Override
public IntsRef handleWayTags(IntsRef edgeFlags, ReaderWay way,
                        EncodingManager.Access accept,
                        long relationFlags) {
   if (accept.canSkip())
      return edgeFlags;
   double speed = getSpeed(way);
   setSpeed(false, edgeFlags, speed);
   setSpeed(true, edgeFlags, speed);
   boolean isRoundabout = roundaboutEnc
         .getBool(false, edgeFlags);
   if (isOneway(way) || isRoundabout) {
      if (isForwardOneway(way))
         accessEnc.setBool(false, edgeFlags, true);
      if (isBackwardOneway(way))
         accessEnc.setBool(true, edgeFlags, true);
   } else {
      accessEnc.setBool(false, edgeFlags, true);
      accessEnc.setBool(true, edgeFlags, true);
   }
   return edgeFlags;
}
```

Лістинг Б.9 - Код isBackwardOneway TramFlagEncoder.java

```
protected boolean isBackwardOneway(ReaderWay way) {
   return way.hasTag("oneway", "-1");
}
```

Лістинг Б.10 - Код isForwardOneway TramFlagEncoder.java

```
protected boolean isForwardOneway(ReaderWay way) {
   return !way.hasTag("oneway", "-1");}
```

Лістинг Б.11 - Код isOneway TramFlagEncoder.java

```
protected boolean isOneway(ReaderWay way) {
   return way.hasTag("oneway", oneways);
}
```

Лістинг Б.12 - Код defineTurnBits TramFlagEncoder.java

```
@Override
public int defineTurnBits(int index, int shift) {
   return shift;
}
```

Лістинг Б.13 - Код isTurnRestricted TramFlagEncoder.java

```
@Override
public boolean isTurnRestricted(long flags) {
   return false;
}
```

Лістинг Б.14 - Код getTurnCost TramFlagEncoder.java

```
@Override
public double getTurnCost(long flag) {
   return 0;
}
```

Лістинг Б.15 - Код getTurnFlags TramFlagEncoder.java

```
@Override
public long getTurnFlags(boolean restricted, double costs) {
   return 0;
}
```

Лістинг Б.16 - Код getVersion TramFlagEncoder.java

```
@Override
public int getVersion() {
   return 1;
}
```

Лістинг Б.17 - Код createEncodedValues TramFlagEncoder.java

```
@Override
public void createEncodedValues(List<EncodedValue>
registerNewEncodedValue, String prefix, int index) {
   super.createEncodedValues(registerNewEncodedValue, prefix, index);
   registerNewEncodedValue.add(speedEncoder = new
FactorizedDecimalEncodedValue(prefix + "average_speed", speedBits,
speedFactor, false));
}
```

Лістинг Б.18 - Код toString TramFlagEncoder.java

```
@Override
public String toString() {
   return "tram";
}
```

Лістинг Б.19 - Код Dockerfile

```
FROM openjdk:8-jdk
ENV JAVA_OPTS "-server -Xconcurrentio -Xmx2g -Xms2g -XX:+UseG1GC -XX:MetaspaceSize=100M -Ddw.server.applicationConnectors[0].bindHost=0.0.0.0 -Ddw.server.applicationConnectors[0].port=8989"
VOLUME [ "/data" ]
RUN mkdir -p /data && mkdir -p /graphhopper
COPY . /graphhopper/
WORKDIR /graphhopper
RUN ./graphhopper.sh build
EXPOSE 8989
ENTRYPOINT [ "./graphhopper.sh", "web" ]
```

Додаток В
Апробація результатів

Додаток Г
Диск